

Singapore Management University

Institutional Knowledge at Singapore Management University

Dissertations and Theses Collection (Open Access)

Dissertations and Theses

6-2020

Online spatio - Temporal demand supply matching

Meghna LOWALEKAR

Singapore Management University, meghna.2015@phdis.smu.edu.sg

Follow this and additional works at: https://ink.library.smu.edu.sg/etd_coll



Part of the [Artificial Intelligence and Robotics Commons](#), and the [Systems Architecture Commons](#)

Citation

LOWALEKAR, Meghna. Online spatio - Temporal demand supply matching. (2020). 1-279.

Available at: https://ink.library.smu.edu.sg/etd_coll/295

This PhD Dissertation is brought to you for free and open access by the Dissertations and Theses at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Dissertations and Theses Collection (Open Access) by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email cherylds@smu.edu.sg.

ONLINE SPATIO-TEMPORAL DEMAND SUPPLY
MATCHING

MEGHNA LOWALEKAR

SINGAPORE MANAGEMENT UNIVERSITY

2020

Online Spatio - Temporal Demand Supply Matching

by

Meghna Lowalekar

Submitted to School of Information Systems in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Computer Science

Dissertation Committee

Pradeep Varakantham (Supervisor/Chair)

Associate Professor

Singapore Management University

Patrick Jaillet (Supervisor)

Professor

Massachusetts Institute of Technology

Akshat Kumar

Assistant Professor

Singapore Management University

Cheng Shih-Fen

Associate Professor

Singapore Management University

Yossiri Adulyasak

Associate Professor

HEC Montreal

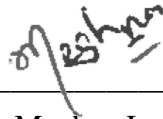
Singapore Management University

2020

Copyright (2020) Meghna Lowalekar

I hereby declare that this PhD dissertation is my original work
and it has been written by me in its entirety.
I have duly acknowledged all the sources of information
which have been used in this dissertation.

This PhD dissertation has also not been submitted for any degree
in any university previously.



Meghna Lowalekar
30 June 2020

Online Spatio - Temporal Demand Supply Matching

Meghna Lowalekar

Abstract

The rapid growth of cities in developing world coupled with the increase in rural to urban migration have led to cities being identified as the key actor for any nation's economy. Shared mobility has become an integral part of life of people in cities as it improves efficiency and enhances transportation accessibility. As a result, the mismatch between the demand and supply of shared mobility resources has a direct impact on people's life. Thus, the goal of my dissertation is to develop solution strategies for these real-time (online) spatio-temporal demand supply matching problems for shared mobility resources which can enhance the service quality by considering expected future demand.

These problems involve a set of customer requests which need to be matched with the available resources (taxis, bikes at stations etc.). The key characteristic that affect the complexity of these problems is the nature of the customer requests involved which in turn affects the matching decisions. We categorize these matching problems into two categories based on the nature of customer requests. In the first category, each customer request is associated with an origin and destination. These customer requests need to be matched with servers/vehicles which pick them from their origin and drop at their destination. The applications include ridesharing, food delivery systems etc. When the servers are of unit-capacity, the matching graph is a bipartite graph with servers on one side and customer requests on the other side. The complexity of the problems increases when multi-capacity servers are used as the underlying matching graph is no longer bipartite but a tripartite graph with servers, requests and request groups (combinations of requests that can travel together).

The second category involves problems where customer requests for resources stored at different warehouses/stations either by walking into the warehouse or from a remote location. In this category, each customer request is either requesting a pick-up or drop-off of resources but not both. For example, in bikesharing sys-

tems, customers either request to pick-up a bike from a station or request to drop-off (return) an earlier picked bike at stations. The multi-capacity servers in these problems provide an assistance in matching these customer requests with required resources by performing a redistribution of these resources across different stations. The matching graph is complex as in addition to matching customer requests with the resources at stations, it also involves matching servers with the decisions of redistributing resources across stations.

The other characteristics which make these matching problems hard, is the scale of the problem (thousands of resources and customer requests), uncertainty in the environment (uncertain customer demand) and the need of making real-time sequential and connected decisions. To tackle these different challenges, I develop online data driven optimization approaches which consider expected future demand. To evaluate the performance of these algorithms, I perform experiments on real world datasets and where possible, I also provide a theoretical bound on the algorithm's performance which is measured using a metric called competitive ratio.

Specifically, for solving the first category of problems, I propose an online data-driven multi-period two-stage stochastic optimization approach and a neural approximate dynamic programming approach which can compute the future effect of current assignments based on the historical data samples. The experimental results on three real world datasets show significant improvement in performance over existing approaches. For a special case, I propose an online adaptive algorithm which can achieve a competitive ratio of γ , where γ is a solution to the equation $(1 - \gamma)^{\kappa+1} = \gamma$ and κ is the maximum capacity of the servers.

For solving the second category of problems, I propose an online data driven multi-period two-stage stochastic optimization approach and a greedy online anticipatory heuristic which use historical demand samples to compute the future effect of current assignments. Experimental results on two real world datasets show that our future demand driven matching approaches provide 20% gain over existing approaches.

Contents

1	Introduction	1
1.1	Motivation and Background	1
1.2	OLYMPIAD - Requests with Pick-up And Drop-off Locations	4
1.3	OLYMPIOD - Requests with Pick-up or Drop-off Locations	7
1.4	Contributions	8
1.5	Results	10
1.6	Overview of the Dissertation	11
2	Background for Decision Making in OLYMPIAD Problems	13
2.1	Model for OLYMPIAD problems	13
2.2	Related Work	16
2.2.1	Online Matching	16
2.2.2	Online MDPs	18
2.2.3	Online Stochastic Optimization	20
2.2.4	Online Multi Vehicle Pick-up and Delivery Problem	21
2.3	Approximate Dynamic Programming (ADP)	27
2.4	Existing Approaches for solving U-OLYMPIAD problems	29
2.4.1	Greedy Algorithm (GD)	30
2.4.2	Randomized Greedy Algorithm (RGD)	30
2.4.3	One-Step Bipartite Matching (OS)	31
2.4.4	Hybrid Multi-Period Two-Stage Stochastic Optimization (HSS)	31
2.4.5	Approximate Dynamic Programming (ADP)	32

2.4.5.1	Value Function Approximation	34
2.5	Existing Approaches for Solving M-OLYMPIAD problems	38
2.5.1	Trip Based Formulation (TBF)	38
2.5.1.1	Heuristics for Online Computation	39
3	Optimization Approaches to Solve U-OLYMPIAD Problems	41
3.1	Optimization Formulation for U-OFLYMPIAD Problems	41
3.1.1	Discussion on $\delta_{ij}^{t,t'}$	44
3.1.2	Example	46
3.2	Optimization formulation for U-OLYMPIAD	48
3.2.1	Example	51
3.2.2	Benders Decomposition	53
3.2.3	Complexity Analysis	56
3.2.4	Competitive Ratio	57
3.3	Relaxing the Assumptions	59
3.3.1	Multi-Period Two-Stage Stochastic (MSS) Optimization For- mulation for U-OLYMPIAD	60
3.3.2	Dummy Requests	62
3.4	Experiments	63
3.4.1	Datasets	64
3.4.1.1	Zone Creation	65
3.4.2	Experimental Settings	66
3.4.3	Main Results	69
3.4.4	Justification for Values of Parameter Settings	75
3.4.5	Synthetic Scenarios Where MSS and BD Do Not Improve Performance	80
3.5	Real World Deployment	83
3.6	Summary	85

4	Neural Approximate Dynamic Programming Approach to Solve M-OLYMPIAD	86
	Problem	86
4.1	NeurADP: Neural Approximate Dynamic Programming	86
4.1.1	Departure From Past Work	88
4.1.2	Approximate Dynamic Programming Model for the M-OLYMPIAD problem	90
4.1.3	Value Function Decomposition	92
4.1.4	Value Function Estimation for NeurADP	94
4.1.5	Overcoming challenges in Neural Network Value Function Estimation	95
4.1.5.1	Improving stability of Bellman updates:	95
4.1.5.2	Addressing the data scarcity:	95
4.1.5.3	Practical simplifications:	96
4.2	Summary	97
5	Optimization Approaches to Solve M-OLYMPIAD Problems	98
5.1	ZAC: A Zone pAth Construction Approach for Solving M-OLYMPIAD Problem	99
5.1.1	Offline: Partial Paths Generation	102
5.1.2	Online	103
5.1.2.1	Generation of the RPS graph	104
5.1.2.2	Finding Optimal match in RPS graph	109
5.2	ZACBenders: A non-myopic approach for solving M-OLYMPIAD .	110
5.2.1	Challenges in solving M-OLYMPIAD with future information	111
5.2.2	ZACBenders Approach	113
5.2.2.1	Two-Stage Stochastic Approximation	113
5.2.2.2	ZACFuture: Optimization formulation to Solve the Two-Stage Stochastic Approximation	116
5.2.2.3	Benders Decomposition to Efficiently Solve ZA- CFuture Optimization Formulation	118

5.3	Summary	123
6	Experimental Results for M-OLYMPIAD Problems	124
6.1	Datasets	125
6.2	Experimental Settings	127
6.3	Results on Real World Datasets	130
6.3.1	Comparison with NeurADP	135
6.4	Results on Synthetic Dataset	137
6.5	Justification for values of algorithmic parameter settings	138
6.5.1	Identification of Right Clustering Method	138
6.5.2	Identification of Right Value of M	139
6.5.3	Number of Samples	140
6.5.4	LookAhead Duration	141
7	Background for Competitive Ratio Analysis for M-OLYMPIAD Problems	142
7.1	Related Work	142
7.2	Background	145
7.2.1	Competitive Ratio in expectation	145
7.2.2	OM-RR-KAD	145
8	Competitive Ratio Analysis for M-OLYMPIAD Problems	148
8.1	Batch Arrival of vertices	149
8.2	Multi-capacity Reusable Servers	153
8.2.1	Model: OPERA	153
8.2.2	Online Algorithm	156
8.3	Experiments	161
9	Background for Decision Making in OLYMPIOD Problems	166
9.1	Model	166
9.2	Related Work	169

9.3	Simulation Model	171
10	Optimization Approaches to Solve OLYMPIOD Problems	173
10.1	Multi-Period Two-Stage Stochastic Optimization	173
10.2	Lagrangian Dual Decomposition	179
10.3	Greedy Online Anticipatory Heuristic	183
10.4	Experiments and Results	186
10.5	Summary	194
11	Conclusion and Future Work	195
11.1	Future Directions	195
A	Supplementary material for the Chapter 3	198
A.1	Proof of Proposition 1	198
A.2	Proof of Proposition 2	202
A.2.1	Proof of Proposition 3	207
A.3	Proof of Proposition 4	209
A.4	Proof of Proposition 5	213
B	Supplementary material for the Chapter 4	217
B.1	Neural Network and Training Specifics	217
B.2	Details of Baseline algorithm	219
B.2.1	Rebalancing	219
C	Supplementary material for the Chapter 8	222
C.1	Proof of Proposition 6	222
C.2	Proof of Proposition 7	225
C.3	Expected Number of times group of type v^g can be formed in round t	229
C.4	Details about the Optimization Formulation LPShare	232
C.5	Proof of Proposition 8	233
C.6	Example showing the groups considered at different steps	237
C.7	Complete proof of Proposition 9	238

C.8 Proof of Corollary 1	243
References	245

List of Figures

1.1	Shared Mobility	2
1.2	Key Characteristics of problems and proposed solution approaches .	3
1.3	Online Spatio-Temporal Demand Supply Matching - Contributions .	9
2.1	Related Work	22
3.1	Cases for server assignment	44
3.2	U-OFLYMPIAD Example	48
3.3	U-OLYMPIAD Example	51
3.4	Zone Creation	65
3.5	Comparison of Revenue earned and Number Of Requests served using different algorithms at each decision epoch. $ \mathcal{L} = 483, \xi^D $ $= 10, \mathcal{N}_i^1 = 2000$ (a) (b) Dataset1 (c) (d) Dataset2	68
3.6	Comparison of Revenue earned and Number Of Requests served using different algorithms at each decision epoch. $ \mathcal{L} = 436, \xi^D $ $= 10, \mathcal{N}_i^1 = 2000$ (a), (b) NYDataset	69
3.7	Comparison of Empirical Competitive Ratio of algorithms during Peak and Non Peak Time, $ \xi^D = 10$, (a) Dataset1 $\mathcal{N}_i^1 = 2000$ (b) Dataset2 $\mathcal{N}_i^1 =$ As observed in data (c) NYDataset $\mathcal{N}_i^1 = 1000$. . .	72
3.8	Comparison of the revenue earned and the number of requests served using different algorithms. NYDataset: In (a) and (b) $ \mathcal{L} = 436, \xi^D $ $= 10$	74

3.9	Comparison of the revenue earned and the number of requests served using different algorithms. (a), (b) Dataset1 and (c), (d) Dataset2. In (a), (b), (c) and (d) $ \mathcal{L} = 483, \xi^D = 10$	75
3.10	Run time comparison of our algorithms for different number of servers(a) $ \mathcal{L} = 483, \xi^D = 10$ Dataset1 (b) $ \mathcal{L} = 483, \xi^D = 10$ Dataset2 (c) $ \mathcal{L} = 436, \xi^D = 10$ NYDataset	76
3.11	Revenue and the run-time comparison of the Offline algorithm on different datasets for different decision epoch length.	76
3.12	Mean and Deviation in the number of requests available at each hour for NYDataset	77
3.13	Comparison of empirical competitive ratio of algorithms for different number of samples. (a)Dataset 1: $ \mathcal{L} = 483, \sum_{i \in \mathcal{L}} \mathcal{N}_i^1 = 2000$, (b) NYDataset : $ \mathcal{L} = 436, \sum_{i \in \mathcal{L}} \mathcal{N}_i^1 = 1000$	78
3.14	Comparison of run time of algorithms for different number of samples.(a)Dataset 1: $ \mathcal{L} = 483, \sum_{i \in \mathcal{L}} \mathcal{N}_i^1 = 2000$, (b) NYDataset : $ \mathcal{L} = 436, \sum_{i \in \mathcal{L}} \mathcal{N}_i^1 = 1000$	79
3.15	(a)Dataset 2: $ \mathcal{L} = 483, \sum_{i \in \mathcal{L}} \mathcal{N}_i^1 = 5000$	80
3.16	Revenue model with no bias for short trips: $ \mathcal{L} = 50, \sum_{i \in \mathcal{L}} \mathcal{N}_i^1 = 2000, \frac{1}{30} \sum_{t=1}^{30} \mathcal{D}^t = 2507$	81
3.17	High proportion of long distance trips with far apart zones: $ \mathcal{L} = 5, \sum_{i \in \mathcal{L}} \mathcal{N}_i^1 = 10, \sum_{t=1}^{30} \mathcal{D}^t = 21$	83

4.1	Schematic outlining our overall approach. We start with a hypothetical \mathcal{G} , $\mathcal{D}_{\nabla}^{\infty}$ and \mathcal{V} in (A) . The grid represents a road network. The blue people and circles correspond to user requests and the nearest street intersection that they're mapped to respectively. The blue dotted lines represent the shortest path between the pick-up and drop-off points of a request. The red and green triangles correspond to existing pick-up/drop-off points for the red and green servers respectively. The dotted lines describe their current trajectory. In (B) we map the requests and their combinations to servers that can serve them under the constraints defined by τ and λ to create feasible actions using the approach presented in (Alonso-Mora, Samaranayake, Wallar, Frazzoli, & Rus, 2017). In (C) , we score each of these feasible actions using our Neural Network Value Function. In (D) , we create a mapping of requests to servers that maximizes the sum of scores generated in (C) using the Integer Linear Program (ILP) in Table 4.1. In (E) , we use this final mapping to update the score function (Section 4.1.4). In (F) , we simulate the motion of servers until the next epoch either based on their current trajectories or a re-balancing strategy. This process then repeats for the next decision epoch.	87
5.1	(a) Representation of RTV graph generated by the model in (Alonso-Mora, Samaranayake, et al., 2017) for capacity 2. (b) Representation of RPS graph generated by ZAC approach.	99
5.2	Example Zone Based Path.	102
5.3	Representation of assignment of server and request to a zone path	107

5.4	Assignment of servers and requests to zone paths over multiple samples of future demand. We use RPS_e^k to denote the RPS graph for decision epoch e in sample k . For the special case of first decision epoch, we denote the graph by RPS_1 . \mathcal{P}_e^k denote the set of paths generated for decision epoch e in sample k . $\xi_e^{D,k}$ denote the set of requests available at decision epoch e in sample k . $V_e^k(RPS_{e-1}^k)$ denote the state of servers at decision epoch e in sample k as a result of assignments obtained by solving the RPS graph at previous decision epoch in the same sample. Therefore, at each decision epoch for each sample, it is a tripartite matching between servers, paths and requests.	112
5.5	Two-Stage stochastic approximation for assignment of servers and requests to zone paths over multiple samples of future demand (For $\kappa = 4$). The zone and decision epoch mentioned in the oval are the zone and decision epoch at which the paths at the first stage in set \mathcal{P} ends. Therefore, servers have dropped all the assigned requests from set \mathcal{D}_r^1 (in first stage), once they reach the zone and decision epoch present in the oval. The number inside diamond represents the number of empty servers present in the zone and decision epoch mentioned in the oval box. At second stage, for each sample, a bipartite matching is performed between empty servers and available requests for all future decision epochs as compared to the tripartite matching between servers paths and requests for each sample in each decision epoch as shown in Figure 5.4.	117
5.6	ZACBenders Approach: Finding Optimal Assignment of requests to paths to servers	122
6.1	Street network for synthetic dataset. Train stations are marked with red.	126

6.2	Comparison of ZACBenders, ZAC and TBF on NYDataset for $\tau=180$ seconds, $\lambda=600$ seconds and $\Delta = 60$ seconds	131
6.3	Comparison of ZACBenders, ZAC and TBF on Dataset1 for $\tau = 180$ seconds, $\lambda = 600$ seconds and $\Delta= 60$ seconds	132
6.4	Comparison of ZACBenders, ZAC and TBF for NYDataset for 1000 servers and varying values of Δ , $\tau = 300$, $\lambda = 600$ seconds	133
6.5	Comparison of ZACBenders, ZAC and TBF for NYDataset for 1000 servers and different time of the day. $\tau = 300$, $\lambda = 600$ seconds . . .	134
6.6	NYDataset - 1000 servers, $\Delta=60$ seconds.	135
6.7	Comparison of service rate on NYDataset. (a) (b) Number of servers =1000 (b)(c) $\tau = 300, \lambda = 600$	136
6.8	Synthetic Dataset with $\tau = 120, \lambda = 240$ and $\Delta = 60$ seconds . . .	138
6.9	Comparison of service rate, runtime and abstraction error with different clustering methods and zone sizes for $M = 1$, number of servers = 1000, capacity = 10, $\tau = 300$, $\lambda = 600$ seconds	138
6.10	Comparison of service rate, runtime and abstraction error with different values of M and zone sizes for NYDataset, number of servers = 1000, capacity 10, $\tau = 300$, $\lambda = 600$ seconds	139
6.11	Comparison of service rate for different number of samples and lookahead duration number of servers = 1000, capacity = 4, $\tau = 300$, $\lambda = 600$ seconds	139
8.1	The Figure depicts the difference in the processing of algorithms in unit-capacity sequential, unit-capacity batch and multi-capacity share case. The online component in each of the algorithms corresponds to the processing in round t for a single instance of arrival of vertices. We only show the detailed flow diagram in the first block for each of the algorithms, rest of the blocks will have similar flow. .	151

8.2	The Figure depicts the tripartite graph used in the OPERA model. It is a combination of 2 bipartite graphs. The goal is to find the matching in the first bipartite graph subject to the constraints enforced due to the edges present in the second bipartite graph. The blue numbers in \mathcal{V} indicate the number of vertices of each type available and blue numbers in \mathcal{V}^g denote the number of groups of each type which can be formed using available vertices in \mathcal{V} . The blue lines indicate a valid assignment of servers in \mathcal{U} to groups in \mathcal{V}^g . Red lines indicate an invalid assignment as the vertex of type v_1 is used 3 times in this assignment but there are only 2 vertices of type v_1 available.	154
8.3	$ \mathcal{U} = 10$, $ \mathcal{V} = 10$, $T = 200$ (a) Varying κ (b) $\kappa = 2$	163
8.4	$ \mathcal{U} = 30$, $ \mathcal{V} = 121$, $T = 240$, Real Dataset (a) 12am (b) 8am	164
10.1	(a) and (d) Performance comparison of MSS and LDD. (b) and (e) Lost demand comparison of LDD for different Δ and Q values. (c) and (f) Lost demand comparison of LDD for different Q values.	189
10.2	Performance comparison of GOAH	190
A.1	Example min cost flow network for U-OFLYMPIAD	201
A.2	Modified Example flow network for U-OFLYMPIAD	202
A.3	Example clause - $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$	207
B.1	Neural Network Architecture	218

List of Tables

2.1	One Step Bipartite Matching	31
2.2	HSS	33
2.3	Formulation using the Linear value function approximation	37
2.4	TBF Formulation	39
3.1	U-OFLYMPIAD formulation	43
3.2	TSS optimization formulation	49
3.3	Master Formulation	54
3.4	Slave Formulation	54
3.5	Dual Slave Formulation	55
3.6	Master Formulation with Optimality Cuts	56
3.7	MSS	59
3.8	MSS formulation with Dummy requests	61
3.9	Experiment section outline	63
3.10	Real World Deployment - DGS results	83
4.1	Optimization Formulation for assignment of servers to feasible actions	92
5.1	Notations	109
5.2	Optimization Formulation for ZAC	110
5.3	Differences between ZAC and ZACBenders	113
5.4	Optimization Formulation for Two-Stage stochastic approximation of M-OLYMPIAD with future samples	118
5.5	Optimization Formulation for Master problem - ZACBenders	120

5.6	Optimization Formulation for Slave problem (Primal)- ZACBenders	120
5.7	Optimization Formulation for Slave problem (Dual) - ZACBenders	121
5.8	Optimization Formulation for Master problem (with optimality cuts) - ZACBenders	122
6.1	Experiment Section Outline	125
6.2	Details for different datasets	126
6.3	Inputs to all algorithms	128
6.4	Algorithm Parameter Settings	129
7.1	Optimization Formulation - Unit-Capacity Sequential Arrival	146
8.1	Optimization Formulation - Multi-capacity	156
10.1	Notation	174
10.2	MSS Formulation	175
10.3	Repositioning Constraints	176
10.4	Routing Constraints	177
10.5	Lost demand reduction	188
10.6	Fuel cost comparison-Hubway 3pm	188
10.7	Number of bikes rented for different trip duration (in minutes)- Hubway 3pm	190
A.1	MinCost Flow and U-OFLYMPIAD	200
B.1	Differences between Baseline and Alonso Approach	220
B.2	Optimization Formulation for Rebalancing unassigned servers	220
C.1	Optimization Formulation - Unit-Capacity Batch Arrival	222
C.2	Optimization Formulation - Batch Arrival - For a fixed sequence a and r	225
C.3	Optimization Formulation - Multi-Capacity	232

C.4 Optimization Formulation - Multi-Capacity - For a fixed sequence
a and *r* 237

Acknowledgement

This dissertation would not have been possible without the help and support of all the people around me.

First and foremost, I would like to thank my advisor Associate Professor Pradeep Varakantham without whose support this dissertation would probably be non-existent. Thank you Pradeep for teaching me everything from doing good research to writing papers and presenting my work. Thank you for supporting me at every stage of my PhD journey and helping me grow as a researcher and a person. I was fortunate to be advised by you and hopefully I will be able to emulate all the things I learned from you in my research career. I would also like to thank my co-advisor Professor Patrick Jaillet for his invaluable reviews, detailed discussions and guidance. Thanks Patrick for always being so supportive and encouraging. Your guidance helped me a lot in learning the correct way to approach different problems especially the theoretical ones. I would also like to thank you for hosting me at MIT and providing me the opportunity to interact with different people at MIT, it definitely helped me a lot.

I thank all my committee members Assistant Professor Akshat Kumar, Associate Professor Shih-Fen Cheng and Associate Professor Yossiri Adulyasak for their valuable comments which helped in improving the dissertation. I would specially like to thank Associate Professor Shih-Fen Cheng for making me a part of DGS project that allowed me to see the impact of my work in the real-world. I thoroughly enjoyed working on the project and learned a lot in the process.

I have been extremely lucky to collaborate with several people during my PhD.

I would like to thank Sanket Shah for working so hard on the NeurADP work. Sanket, it was a great experience to work with you on this project. I enjoyed all the discussions we had on this and other related problems. I was fortunate to work with Supriyo Ghosh on the bikesharing work. Thanks Supriyo for all the discussions and help during the project. I would also like to thank Sanjay Dominik Jena for providing invaluable comments on the bikesharing work. A big thanks goes to Murali and Aayush for creating the demonstration system for the bikesharing work. I also got a chance to collaborate with Shashi Shekhar Jha, Nicholas Wong and Rishikeshan Rajendram for the DGS project. I thank them for the useful discussions and suggestions during the course of the project.

Next, I would like to thank Chew Hong, Pei Huan, Yar Ling and Caroline for all the help with the administrative work. Thanks Yong Fong for organizing different events which made the PhD journey enjoyable. I would like to thank Bingran, Jocelyn, Juana, Janet, and Katherine from SMART for all the positive interactions and for all the help during the PhD. I would also like to thank SMART for providing me the scholarship support.

My PhD journey would have been very different without the support of my friends Tanvi, Rakesh, Rajiv, Nitya, Pritee and Ankit. I want to thank all of you for always being there during all good and bad times in the last few years. I would also like to thank Asrar, Sandhya, Supriyo, Kapil, Murali, Ritesh, Pallavi, Tarun, Kushagra, Srishti, Abhinav, Sanket, Susobhan, Alolika, Chaitanya, Dexun, JiaJing, and James for all the fun interactions and celebrations in lab.

My deepest gratitude goes to my family for their unconditional love and support. My father guided me in shaping up my career from my initial school days till today. My mother always encouraged me and supported me in the choices I made. I want to thank both of them for making repeated visits to Singapore to take care of me. I would like to thank my brother and sister-in-law for being so loving and caring and for always being there to discuss my problems and providing valuable suggestions. I would like to thank my very close friends Kritika, Prashasti, Raina, Shraddha, Swati

Jain and Swati Sharma for their love and support. Last but not the least, I would like to thank my mentors Professor Kamalakar Karlapalem and Late Ritesh Kumar Tiwari for introducing me to research during my IIT days. Ritesh sir, I hope you are proud of me, wherever you are. My special thanks goes to Professor Kamalakar Karlapalem for suggesting me to go to Singapore and work with Pradeep, without which this PhD journey would not have started.

To my parents for their unconditional love and support
To my brother and sister-in-law for always being my side.
To my lovely nephew Shreyas for giving me hope.

and

To my mentors Prof Kamalakar Karlapalem and Late Ritesh Kumar Tiwari.

Chapter 1

Introduction

1.1 Motivation and Background

Today, 55% of the world's population lives in cities or urban areas and this percentage is expected to increase to 68% by 2050 (United Nations, 2018). Therefore, for any nation's growth cities are an important factor. Around the world, many cities are being transformed by using shared mobility resources which improve efficiency, enhance transportation accessibility, provide cost savings and monetize unused resources. The term shared mobility refers to the shared use of a vehicle, bicycle, or any other transportation mode. Figure 1.1 highlights different mobility and delivery applications which fall under the shared mobility umbrella term (Shaheen, Cohen, Zohdy, et al., 2016). The fast paced innovations in internet technologies have enabled shared mobility to become an integral part of life of people in cities. The mismatch between the demand and supply of these shared mobility resources has a direct impact on people's life as apart from causing inconvenience to people it can also contribute towards increasing pollution and congestion on the road by leading to extensive usage of private vehicles. Congestion can cost billions of dollars (Schrank, Eisele, & Lomax, 2012), by measures such as lost time, wasted fuel, and increased cost of doing business. Pollution is also a global concern with World Health Organization attributing seven million premature deaths to air pollution. To

overcome these issues and to make shared mobility resources more accessible, it is essential to develop intelligent decision support systems to better manage the demand and supply of these resources.

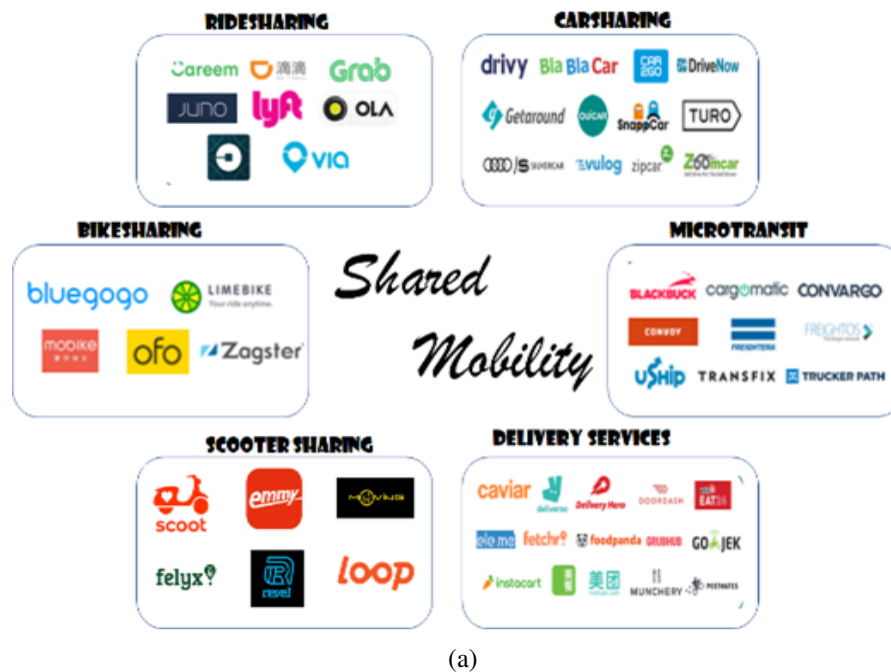


Figure 1.1: Shared Mobility

In all these applications associated with the shared transportation (Agatz, Erera, Savelsbergh, & Wang, 2011; Ghosh, Varakantham, Adulyasak, & Jaillet, 2017; Hosni, Naoum-Sawaya, & Artail, 2014) customers requests need to be matched with servers (e.g., taxis, shuttles, delivery personnel) or resources at stations/warehouses (e.g. bikes located at different docking stations) in an online fashion to optimize the revenue or quality of service. The wide usage of applications such as Uber, Lyft, etc. is a testament to the importance of doing matching well and in quick time.

All these applications have following characteristics

- Involve transportation of resources or people between locations;
- Demand is uncertain and time dependent;
- There is data available about past customer demand;
- Problems are at a societal scale with thousands of customers and servers;
- There is a need to make online (real-time) and sequential (connected) decisions;

- There is a need or an opportunity to optimize number of requests served, revenue or quality of service (e.g., time to pick-up customers or time to drop-off);

Due to the need of making real-time and sequential decision, in this dissertation, I provide approaches which are online (i.e., can continuously react to the incoming demand) and are future demand driven (i.e., current decisions are taken based on the expected future demand).

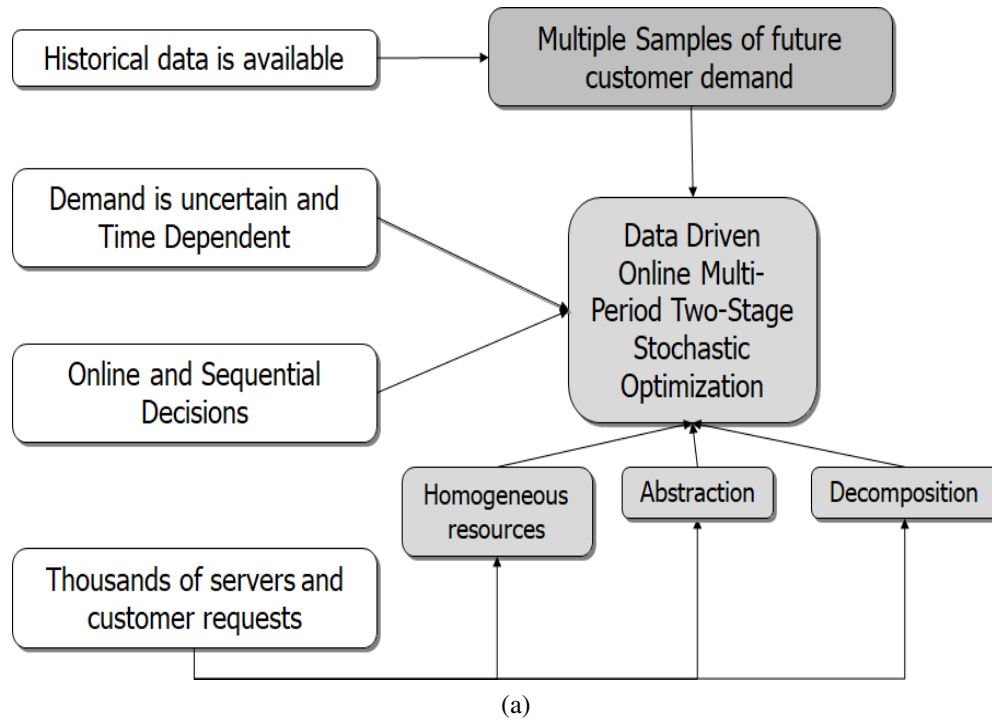


Figure 1.2: Key Characteristics of problems and proposed solution approaches

In the proposed algorithms, to handle the large scale nature of problems, I exploit the homogeneous nature of resources by using abstraction based techniques. The scalability is further improved by using decomposition based approaches which break a large problem into multiple smaller problems which can be solved in parallel. In addition, to make better sequential and connected decisions, I use data driven multi-period two-stage optimization approaches by taking multiple samples of customer requests from the historical data. Figure 1.2 briefly summarizes the how the proposed solution approaches address the key characteristics of the problems in consideration. To evaluate the performance of these algorithms, I perform experiments on the real world datasets. For a special case of last mile transportation

where after serving customers, servers return back to their original location, I also propose an online algorithm which makes decision based on the guidance provided by an offline optimal linear program. As the arrival distribution is known (from historical data), I analyze the algorithm under a known arrival distribution model and provide a bound on the performance of algorithm in comparison to an offline optimal solution (which has complete information).

The Online Spatio-Temporal Demand Supply Matching problems can be categorized into two types ¹ based on the nature of customer requests involved.

1. OLYMPIAD: **OnLine Spatio-Temporal Demand Supply Matching - Requests with Pickup And Drop-off Locations**
2. OLYMPIOD: **OnLine Spatio-Temporal Demand Supply Matching - Requests with Pickup Or Drop-off Locations**

1.2 OLYMPIAD - Requests with Pick-up And Drop-off Locations

In this case, each customer request has a pick-up location and it also includes an associated drop-off location. The example applications include food delivery services, taxi matching services etc. In general, these problems also have a time window associated with pick-up and drop-offs. Servers/vehicles are assigned to customer requests and they pick the customer requests from the pick-up location and drop them at their drop-off location. When the servers have a unit-capacity, we refer to the problem as U-OLYMPIAD. In this case, servers are matched to one request at a time. On the other hand, when servers are multi-capacity, we refer to the problem as M-OLYMPIAD. The problem becomes more complex when multi-capacity servers are used as servers must be matched to groups of requests and not just to individual requests.

¹This is similar to the classification of Static pick-up and delivery problems provided by Parragh *et.al.* (Parragh, Doerner, & Hartl, 2008)

In this dissertation, I consider the domain of taxi matching services for this class of problems. Given the challenging nature of the problems (stochasticity, dynamism, societal scale, online, multi-step, multi-capacity), most existing work on relevant problems (described in Chapter 2) has focused on myopic algorithms (Karp, Vazirani, & Vazirani, 1990a) like greedy and randomized ranking. While these approaches have good competitive ratios in case of online bipartite matching (single step, unit-capacity), they have obvious inefficiencies in handling multi-step and multi-capacity problems due to their myopic nature. To address these issues, there has been research on multi-period two-stage stochastic models and Approximate Dynamic Programming approaches that consider expected demand for the future time steps for unit-capacity servers. However, these approaches have been limited to small scale problems and are applicable in restricted settings (Powell, 1996; Topaloglu & Powell, 2006; Simao et al., 2009; Powell, 2007; Ritzinger, Puchinger, & Hartl, 2016; Katriel, Kenyon-Mathieu, & Upfal, 2008; Zhang, Smilowitz, & Erera, 2011). A recent work by (Alonso-Mora, Samaranayake, et al., 2017) proposes a trip based formulation to solve the M-OLYMPIAD problem but the approach uses heuristic for online computation which can degrade the quality of matching especially for higher capacity vehicles. Moreover, the approach is myopic in nature, i.e., it does not consider future effect of current assignments.

To overcome these drawbacks of existing work, I propose using abstraction and decomposition based approaches which can support thousands of servers and customers. The quality of matching is further improved by using multiple demand samples from historical data. For U-OLYMPIAD problems, I propose a data driven multi-period two-stage optimization formulation to match servers to customer requests while considering multiple future customer demand samples. I exploit the homogeneous nature of servers where instead of considering each server individually, the formulation considers the number of servers present in abstracted locations (i.e., zones). To make the formulation scalable, Benders decomposition is used. For M-OLYMPIAD problems, I propose neural approximate dynamic pro-

gramming approach and a zone path ² based multi-period two-stage optimization approach. These approaches use multiple samples of future demand from the historical data. Due to the complexity involved in M-OLYMPIAD problems, the approximate dynamic programming and two-stage optimization formulations are not exact and approximations are used to make them scalable (more details in chapter 4, 5). To further improve the scalability of two-stage optimization formulation, similar to U-OLYMPIAD problems, I use Benders decomposition to decompose the large optimization problem into multiple smaller problems.

Along with providing solution approaches which perform well empirically, I also focus on designing and analyzing online algorithms which can provide theoretical guarantees on the worst case performance in some special cases. The metric used is competitive ratio which is defined as the worst-case ratio between the objective of the solution found by the online algorithm and the solution found by an offline optimal algorithm which has complete information. In case of U-OLYMPIAD problems, Dickerson *et.al.* (Dickerson, Sankararaman, Srinivasan, & Xu, 2017) were able to provide a $\frac{1}{2}$ bound for the special case when the servers come back to their original location after serving the requests. Unfortunately, this is only applicable for unit-capacity servers and cannot be directly adapted to consider multi-capacity servers due to the change in the structure of underlying matching graph. Another limitation is that the existing theoretical work for U-OLYMPIAD problems has primarily focused on requests arriving sequentially and not in batches which is a desirable property when considering M-OLYMPIAD problems (for instance, last mile services at train stations need to consider that the large number of passengers will arrive and request for last mile transportation to their home at the same time.).

The multi-capacity servers make the problem challenging because servers must be matched to groups of requests and not just to individual requests. As mentioned

²A zone path is a path that connects zones (a zone is an abstraction for multiple individual locations) and therefore it can group multiple requests that have "nearby" or "on the way" pick-ups and drop-offs. The servers and requests are matched to these zone-paths (instead of request groups) to efficiently perform matching in the resulting tripartite graph.

before, unlike U-OLYMPIAD problems, where the underlying graph is bipartite, the M-OLYMPIAD problems have a tripartite graph (Beineke, 1980) with reusable servers (vehicles), request groups (i.e., combinations of passenger requests) and on-line requests. The desired matching between the servers and request groups (combination of requests) is constrained by the edges between requests and request groups (i.e., a request can be part of at most one request group in final assignment) in this tripartite graph. To the best of our knowledge, there has been no research on providing performance guaranteed algorithms for such tripartite graphs. To overcome these limitations of the existing work, we design and analyze an adaptive algorithm which provides a bound on the competitive ratio for U-OLYMPIAD problems in batch arrival model and also provide a performance guaranteed approach for solving M-OLYMPIAD problems.

1.3 OLYMPIOD - Requests with Pick-up or Drop-off Locations

This category involves problems where resources are present at different warehouses/stations and customer requests for these resources either by walking into these warehouses/stations or from a remote location. In this category, each customer request is either requesting a pick-up or drop-off of resources but not both. The multi-capacity servers/vehicles in these problems provide an assistance in matching these customer requests with required resources by performing a redistribution of resources across different warehouses/stations. The underlying matching graph is complex in this case as in addition to matching customer requests with the resources at stations/warehouses, it also involves matching servers with the decisions of redistributing resources across stations/warehouses. The redistribution decisions consist of two types of decisions: (1) Repositioning decision which specifies the number of resources to be picked up or dropped off at stations (2) Routing decisions: which specify the stations where servers should move next.

The resources which are requested/redistributed can be of a single type or can have multiple types. In this dissertation, I consider the case where resources are of single type. This has application in many practical scenarios like food rescue programs which involve collecting extra food from restaurants and redistributing through agencies to people in need, redistributing money between different bank branches, redistributing bikes between different stations in bikesharing systems etc. In this dissertation, we evaluate our approaches for bikesharing systems. Most of the existing work in bikesharing systems has focussed on static repositioning approaches (Schuijbroek, Hampshire, & van Hoes, 2017; Chemla, Meunier, & Calvo, 2013; Raviv, Tzur, & Forma, 2013). These approaches perform repositioning only at the beginning of the day, but they do not consider the mismatch between demand and supply during the day. Ghosh, Varakantham, Adulyasak, and Jaillet (Ghosh et al., 2015, 2017) consider the problem of dynamic repositioning of bikes by providing an offline policy generation approach based on mean demand computed from the historical data. While the offline policy provides significant improvement over static repositioning approaches, it is unable to consider the changing demand scenarios in real-time. To overcome these limitations, I propose using an online repositioning approach which uses future demand samples to improve current decision.

1.4 Contributions

This section describes my contributions towards solving Online Spatio-Temporal Demand Supply Matching problems. The contributions are summarized in Figure 1.3.

My key contributions in solving OLYMPIAD problems are as follows:

1. I first introduce a data driven two-stage stochastic optimization approach to solve the U-OLYMPIAD problems. The approach considers samples of future customer demand (typically obtained from historical data) for finding the assignment of servers to customers. I also provide a multi-period two-stage

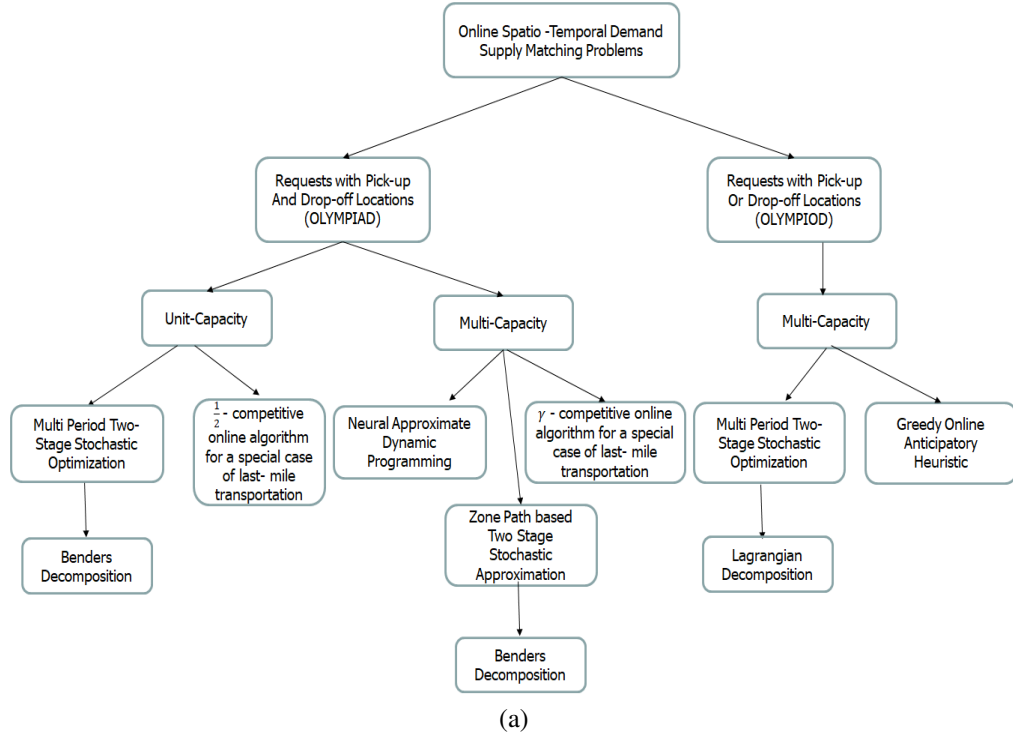


Figure 1.3: Online Spatio-Temporal Demand Supply Matching - Contributions

extension.

2. Given the large scale of problems of interest with thousands of servers and customers, I provide a decomposition of the above formulation to improve parallelism in handling future demand.
3. I provide the theoretical results on the hardness of U-OLYMPIAD problems and provide the theoretical results on competitive ratios for myopic and two-stage algorithms for U-OLYMPIAD problems.
4. I propose Neural Approximate Dynamic Programming approach ³ to solve M-OLYMPIAD problems.
5. I propose a zone path based stochastic optimization formulation to solve M-OLYMPIAD problems. Similar to U-OLYMPIAD case, I provide a decomposition of the formulation to improve parallelism in handling future demand.

³This is a joint work, which was primarily done by Sanket Shah. My contribution lies in modelling it in approximate dynamic programming framework instead of reinforcement learning framework.

6. I propose an adaptive online algorithm which can achieve a competitive ratio of $\frac{1}{2}$ for a special case of U-OLYMPIAD problems where servers return back to their original location after serving a request.
7. I propose an adaptive online algorithm which can achieve a competitive ratio of γ where γ is a solution to the equation $\gamma = (1 + \gamma)^{\kappa+1}$ and κ is the maximum capacity of the servers. Similar to U-OLYMPIAD, these bounds are applicable for a special case of M-OLYMPIAD problems.

My key contributions towards solving OLYMPIOD problems are as follows:

1. I propose a data driven multi-period two-stage stochastic formulation, to consider expected future demand over a set of scenarios to find an efficient repositioning strategy for OLYMPIOD problems.
2. Given the large scale nature of the problem, I provide a Lagrangian decomposition approach that decouples the global problem into routing and repositioning slaves and employs a novel dynamic programming approach to efficiently solve routing slave.
3. I propose a greedy online anticipatory heuristic to solve large scale problems effectively and efficiently.

1.5 Results

1. For U-OLYMPIAD problems, I have evaluated our approaches on the datasets of three major taxi companies. The comparison is performed against myopic algorithms (typically employed by standard taxi matching applications) such as greedy and one-stage bipartite optimal assignment. In addition, we also compare against the Approximate Dynamic Programming (ADP) approach that has been successfully applied in many resource allocation problem and Hybrid Multi-Period Two-Stage Stochastic optimization (HSS) approach used for the truckload assignment problem. We show that our data

driven multi-period two-stage formulation can be solved in times that are competitive to these approaches, while providing 9% gain over other multi-period approaches.

2. A recommendation engine built based on the optimization formulation proposed for U-OLYMPIAD problems is used to provide personalized guidance to taxi drivers in Singapore. The field trial of the driver guidance system (DGS) was conducted with 500 recruited taxi drivers from September 2017-end of 2019. Comparing the vacant roaming times before finding passengers, we discover that by following the DGS recommendations, drivers manage to reduce their vacant roaming times by 27% across all time periods.
3. For M-OLYMPIAD problems, we compare our approaches against the Alonso *et al.*'s (Alonso-Mora, Samaranayake, et al., 2017) on two real world datasets and a synthetic dataset and show that our approaches provide upto 14.7% improvement. In special case of first and last mile transportation on synthetic dataset, our approach provides up to 20% improvement.
4. For OLYMPIOD problems, we conduct experiments on real world bikesharing datasets and obtain 20% reduction in lost demand over existing approaches.

1.6 Overview of the Dissertation

The rest of the dissertation is organized as follows: Chapter 2 describes the model, related work and existing solution approaches for OLYMPIAD (Online Spatio-Temporal Demand Supply Matching - Requests with Pick-up and Drop-off locations) problems. Chapter 3 presents our approaches for solving the U-OLYMPIAD problems. Chapter 4 explains the Neural Approximate Dynamic Programming approach to solve M-OLYMPIAD problems. Chapter 5 describes the Zone Path Construction based approaches to solve M-OLYMPIAD problems. Chapter 6 describe the experimental results for M-OLYMPIAD problems. Chapter 7 describes the

background and related research for competitive ratio analysis for M-OLYMPIAD problems. Chapter 8 describes the adaptive algorithm and competitive ratio analysis for the special case of M-OLYMPIAD problems. Chapter 9 describes the background, models and related work for decision making in OLYMPIOD (Online Spatio-Temporal Demand Supply Matching - Requests with Pick-up or Drop-off location) problems. Chapter 10 describes our approaches to provide online repositioning and routing decisions for OLYMPIOD problems. I finally conclude with the future research plans in Chapter 11. Appendix includes the detailed proofs of propositions present in the dissertation.

Chapter 2

Background for Decision Making in OLYMPIAD Problems

In this chapter, we provide a model to describe OLYMPIAD problems and provide the details of the existing approaches used to solve the U-OLYMPIAD and M-OLYMPIAD problems. In our experimental results in chapter 3 and 6, we compare our solutions with these approaches.

2.1 Model for OLYMPIAD problems

We assume that the time is divided into discrete blocks of duration Δ . Given a set of customer requests available at current decision epoch and the initial location of servers, the goal in OLYMPIAD problems is to find a matching between servers and customer requests such that the objective function (e.g., revenue, number of requests) of the matching is optimized. The key constraint is that the acceptance/rejection decision for requests should be taken at the decision epoch at which they become available. If a request is not assigned a server at which it becomes available, it is considered rejected. Considering potential future samples can help in making better matching decisions (Powell, 1996; Zhang et al., 2011; Godfrey & Powell, 2002). Therefore, we use multiple potential samples of demand requests at future decision epochs. One way of obtaining the potential scenarios is to consider

the scenarios observed in the past data. We consider the future demand samples for ρ duration or over next Q decision epochs. ρ is termed as lookahead duration and $Q = \lceil \frac{\rho}{\Delta} \rceil - 1$. For ease of notation, in the formulation, we use decision epoch 1 to denote the current decision epoch. Therefore, decision epoch t corresponds to the current decision epoch + $(t - 1)$.

Formally an OLYMPIAD problem is defined using following tuple:

$$\langle \mathcal{G}, \mathcal{V}, \mathcal{D}, \xi^D, \mathcal{C}, \mathcal{T}, \mathcal{S}_p, \mu, \kappa, \tau, \lambda \rangle$$

- $\mathcal{G} = (\mathcal{L}, \mathcal{E})$ is a graph with the vertices (\mathcal{L}) as the set of all valid locations ¹. \mathcal{E} defines the adjacency of the locations in set \mathcal{L} .
- \mathcal{V} denotes the set of servers/vehicles ². q_v denotes the set of customer requests present in the vehicle, where each element of q_v is represented by the tuple, $\langle o_j, d_j, t_j \rangle$, where $o_j, d_j \in \mathcal{L}$ denote the origin and destination location of the requests and t_j denote the decision epoch at which the request was assigned to the vehicle.
- \mathcal{D} represents the demand or the set of customer requests for servers. We use \mathcal{D}^1 to denote the set of customer requests at current decision epoch. Each element $j \in \mathcal{D}^1$ is characterized by a tuple $\langle o_j, d_j, R_j^1 \rangle$ where $o_j, d_j \in \mathcal{L}$ denote the origin and destination location of the requests and R_j^1 denotes the number of requests having origin at location o_j and destination at location d_j at current decision epoch. In the context of taxis, this would correspond to the set of customer requests that travel between certain starting and ending locations. The set \mathcal{D}_r^1 is derived from \mathcal{D}^1 with each element $j \in \mathcal{D}^1$ is repeated R_j^1 times, i.e., \mathcal{D}_r^1 contains each request separately.
- ξ^D is the set of customer request samples for the future decision epoch, where $\xi_t^{D,k}$, $t > 1$, represents the set of customer requests at $t - 1$ decision epochs in the future in sample k . Each element $j \in \xi_t^{D,k}$ is characterized by a tuple

¹For example the set of valid locations can be all the nodes in the street network of a city or the set of zones which are used to divide the city.

²We use servers and vehicles interchangeably in the dissertation

$\langle o_j, d_j, R_j^{t,k} \rangle$ where $o_j, d_j \in \mathcal{L}$ denote the origin and destination location of requests and $R_j^{t,k}$ denotes the number of requests having origin at location o_j and destination at location d_j at decision epoch t for sample k .

- \mathcal{C} represents the objective (e.g., revenue, number of requests served, negative of waiting time), with $\mathcal{C}_{v,o_j,d_j}^t$ denoting the objective value obtained by matching a server $v \in V$ to a single customer request with origin and destination locations given by $\langle o_j, d_j \rangle$ at decision epoch t .
- $\mathcal{T}(l, l', t)$ gives the time taken by a server to move from location l to l' at decision epoch t .
- S_p denotes the set of shortest duration paths between all location pairs, with $S_p(l, l', t)$ denoting the shortest path between location l and l' at decision epoch t . Shortest paths are computed based on the $\mathcal{T}(l, l', t)$ values.
- μ denotes the initial distribution of servers at locations. $\mu_v^t(l)$ is set to 1 if server v becomes available at location l at decision epoch t and is 0 otherwise.
- κ denotes the maximum capacity of servers.
- τ denotes the maximum allowed waiting time (while pick-up).
- λ denotes the total maximum allowed delay (including pick-up and drop-off).

Please note that the demand variables (D^1 and ξ) are exogenous variables, i.e., they are not affected by the matching decisions taken by online algorithms. For U-OLYMPIAD problems, $\kappa = 1$. Also, in U-OLYMPIAD, there is no additional delay allowed during drop-off, therefore, $\lambda = \tau$.

Justification for considering deterministic travel time durations: The model considers that the travel time is deterministic and known for the lookahead duration (ρ). This assumption is justified as the typical value of ρ is 30 minutes during which travel times do not change drastically. Moreover, travel time values can be updated after Δ duration (decision epoch duration). The value of Δ is much smaller than ρ (generally between 10 seconds and 5 minutes), so even if there is a change in the conditions, the algorithms can use updated travel time values while making

decisions.

2.2 Related Work

In this section, we describe the multiple threads of research in online sequential decision making that are of relevance to OLYMPIAD problems. Online algorithms typically consider requests that are revealed incrementally over time and algorithms must make decisions based on the requests that are revealed (Borodin & El-Yaniv, 1998). The key threads of research that are of relevance are: online matching, online MDPs, online stochastic optimization algorithms and online multi-vehicle pick-up and delivery problems.

2.2.1 Online Matching

A matching, M in a graph $G(V, E)$ is defined as the set of edges $M \subset E$ such that for every $v \in V$ there is at most one edge in M incident on v . In classical online bipartite matching problem (Karp et al., 1990a), one side of the vertices is known, and the other side of the vertices arrive online. This is formally defined as a graph $G(U, V, E)$ where vertices in the set U are known and vertices in the set V appear online. The goal is to maximize the size of the matching M .

A simple generalization of the online bipartite matching is the weighted case where vertices or edges have weights associated with them and the goal is to maximize the total weight of the matching. This problem has applications in online advertising employed by Yahoo, Google etc. Specifically, in such applications, the goal is to optimize the allocation of a fixed advertising space to incoming advertisers who typically arrive at different times.

We describe the commonly used approaches for solving online bipartite matching problems. In the experimental section, we compare our proposed approach with the following approaches.

1. **Greedy:** Greedy algorithm matches the incoming vertex with the best avail-

able choice. In case of weighted models, it matches with the maximum weighted vertex or edge. Greedy algorithm is shown to have a competitive ratio³ of $\frac{1}{2}$ (Karp et al., 1990a).

2. **Randomized Greedy:** The randomized greedy algorithm perturbs the value of matching by multiplying it by a random number w between 0 and 1. It then greedily matches the servers to customer by using perturbed value. Goel *et.al.* (Aggarwal, Goel, Karande, & Mehta, 2011) show that the randomized greedy algorithm achieves a ratio of $1-\frac{1}{e}$ for the vertex weighted bipartite matching.

In classical online bipartite matching, vertices appear one by one. In our model, discussed in the introduction, a group of requests (possibly more than one) arrive simultaneously at each decision epoch, therefore we can apply standard bipartite matching algorithm on the currently available partial graph. We call this One-Stage Algorithm as it only considers requests available at one stage (the current decision epoch). One-Stage Optimal finds the maximum weighted bipartite matching between available vertices at every decision epoch by solving a linear program. Although most work in online bipartite matching addresses the problem where one side is fixed (Jaillet & Lu, 2013; Mehta et al., 2013; Manshadi, Gharan, & Saberi, 2012), results in (Blum, Sandholm, & Zinkevich, 2006) show that greedy algorithm achieves a competitive ratio of $1/2$ when both sides of vertices appear online. Recently in Wang *et al.* (Wang & Wong, 2015) an algorithm based on water-filling algorithm has been proposed which achieves a competitive ratio of 0.526 in case both sides of vertices appear online.

While there are similarities, there are multiple differences in our work from research in online matching:

- We consider a multi-period two-stage problem, where there are multiple connected rounds of bipartite matching. Therefore, unlike in online bipartite match-

³Competitive ratio of an algorithm is defined as the ratio of the worst case solution obtained by the algorithm and the optimal solution.

ing that assigns one service to only one customer, in this dissertation, we match one service to multiple customers over time. A recent work by Dickerson *et.al.* (Dickerson, Sankararaman, Srinivasan, & Xu, 2018) proposes a new model for Online Matching with (offline) Reusable resources in which resources on one side are reusable, i.e., resources are matched multiple times over time but their model assumes that each resource has a fixed position and comes back to its original position before it can be matched again. This assumption is valid for the special case of last mile transportation which we study in chapters 8 but for the general problems considered in this dissertation where position of resource (server) depends on the previous match (assignment) this assumption is not applicable.

- In addition, the spatio-temporal aspect of requests (not present in traditional online matching problems) adds further computational complexity to the matching problem.
- Finally, in terms of approaches, unlike work in online matching which has primarily considered myopic approaches, we pursue multi-period two-stage approaches that consider potential future requests.

2.2.2 Online MDPs

Another relevant thread of research is Online Markov Decision Processes (MDPs). In the online setting, rewards and transition functions of MDP for future timesteps are unknown. To learn these unknown reward and transition functions, following two cases are considered.

- The more popular sub-thread has focused on the cases where the reward and the transition function are assumed to be stochastically stationary and the instances of the reward and the transition function are revealed depending on the action. In this case, Reinforcement Learning (Szepesvári, 2010) has presented numerous techniques for learning the policies, which are guaranteed to maximize the expected value.

- In the second sub-thread (Even-Dar, Kakade, & Mansour, 2009; McMahan, Gordon, & Blum, 2003; Abbasi, Bartlett, Kanade, Seldin, & Szepesvári, 2013) the revealed reward and transition functions are adversarial to the executed action.

While it is possible to represent our Online Demand Supply Matching Problems as an online MDP, the number of states and actions are exponential in the number of agents. Since the number of agents is in the thousands especially for OLYMPIAD problems, it is even difficult to specify the model. For the large scale MDPs, existing works have used state aggregation (Li, Walsh, & Littman, 2006) and Approximate Dynamic Programming methods (Powell, 2007) to compute policies. As described in Section 2.2.4, Approximate Dynamic Programming methods have been widely applied for the resource allocation problems. We used linear and piecewise linear value function approximation which are shown to work well for truckload assignment and fleet management problem (Godfrey & Powell, 2002; Simao et al., 2009). We provide a comparison with this approach in our experiment section for unit-capacity OLYMPIAD problems. In summary, following are the differentiating factors of the work presented in this dissertation:

- While it is possible to represent our problem as an online MDP, the number of states and actions are exponential in the number of agents. We provide an experimental comparison of our model with the Approximate Dynamic Programming approach which is generally used to solve the large scale MDPs.
- We neither associate adversarial behaviour with the nature nor do we assume stationarity with respect to reward and transition functions. Instead, we assume that the behaviour is similar to what has been observed in the training demand settings.

Given that both Online MDPs and our work are focussed on online sequential decision making under uncertainty, existing work in online MDPs can benefit by adapting the following two key ideas mentioned in this dissertation in order to scale to problems with multiple agents:

- Exploiting anonymity (lack of identity) and homogeneity of agents to address the exponential complexity associated with increasing the number of agents.
- Exploiting decomposability across multiple samples of the future state space evolution.

2.2.3 Online Stochastic Optimization

Stochastic programming is used to model the optimization problems that involve uncertainty. These models take advantage of the fact that the probability distributions which represent the data are known or can be estimated (Shapiro, Dentcheva, & Ruszczyński, 2009). To represent the future uncertainty, multiple samples (scenarios) from the known or estimated probability distribution are considered.

The complexity of these multi-period two-stage stochastic programs increases with increasing the number of timesteps and sample scenarios (Shapiro, 2006). Online anticipatory algorithms (Mercier & Van Hentenryck, 2007; R. Bent & Van Hentenryck, 2004; Mercier & Van Hentenryck, 2011; Ghiani, Manni, & Thomas, 2012) are generally used to solve large scale stochastic integer programs when the set of feasible decisions at each stage is finite (Mercier & Van Hentenryck, 2007). The assumption is that there exists an offline deterministic algorithm for the application. Online anticipatory algorithms relax the non anticipativity constraints in the stochastic program and make decisions online at a time t in three steps:

- Sample the distribution to obtain a subset of future scenarios.
- Optimize each scenario for each possible decision.
- Select the best decision over all scenarios.

The above algorithm requires evaluating each decision for each possible sample which is computationally expensive. Hence, in general, the following two approximations are used.

- *Consensus*: Instead of optimizing each sample for each decision, consensus optimizes each sample once. Only decisions which are optimal for a sample

receive positive score, other decisions are given zero score. The decision having highest score is executed at time t .

- *Regrets*: Regrets algorithm assumes the availability of an application specific regret function which gives fast approximation on the regret value of any decision. Now similar to consensus, regrets algorithm optimizes each sample once but unlike consensus, it assigns score to all decision using the application specific regret function.

Online Anticipatory algorithms have been used in the applications like online vehicle routing, packet scheduling, reservation systems but these applications typically have a small set of feasible decisions at each stage (50-100 requests) (R. Bent & Van Hentenryck, 2004; R. W. Bent & Van Hentenryck, 2004; Mercier & Van Hentenryck, 2007).

Unfortunately, in our case, the number of feasible decisions are in millions (tens of thousands of servers and hundreds/thousands of requests). Furthermore, if we are to optimize for each sample separately, we can only handle very few samples within the online time constraints. Thus, we provide approaches (relaxation, Benders Decomposition, Lagrangian decomposition on top of the linear optimization) that improve scalability considerably to handle millions of decisions. Though Benders Decomposition (Rebennack, 2016; Legrain & Jaillet, 2016; Murphy, 2013) and Lagrangian decomposition (Fisher, 1985; Kumar, Wu, & Zilberstein, 2012; Ghosh et al., 2015, 2017) is typically used to solve the large scale stochastic optimization problems, the novelty is to use this for our updated formulation to obtain competitive results.

2.2.4 Online Multi Vehicle Pick-up and Delivery Problem

This thread of research is closely related to OLYMPIAD problems considered in this dissertation. Online Multi Vehicle Pick-up and Delivery problems typically represent problems where there are multi-capacity servers that transport multiple resources/loads from their origins to destinations. When servers are used to move peo-

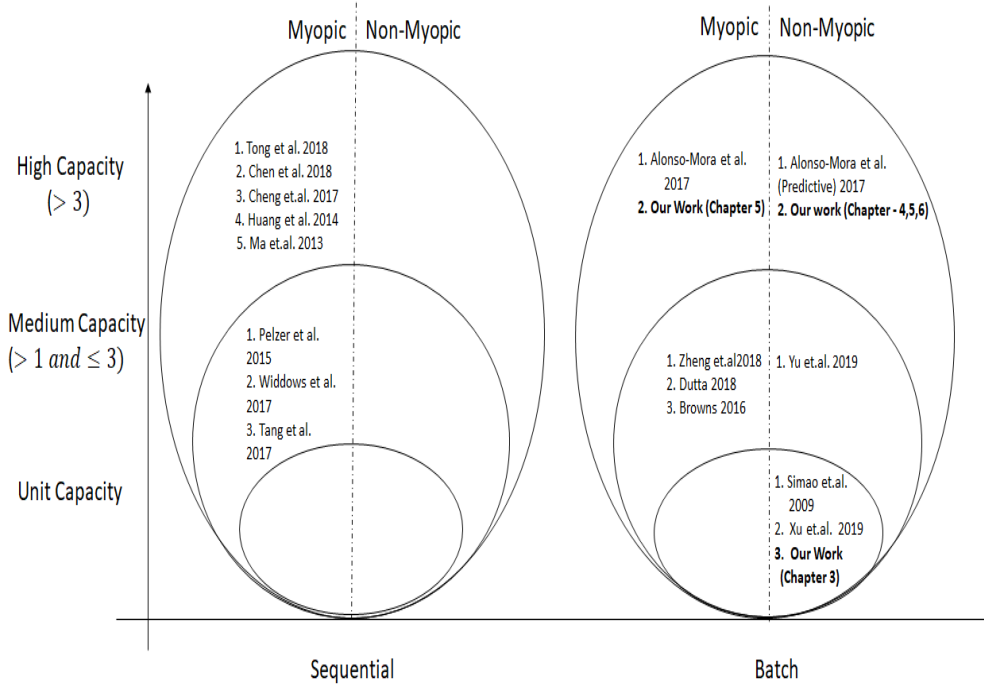


Figure 2.1: Related Work

ple instead of resources, the problem is referred to as dial-a-ride problem (Feuerstein & Stougie, 2001; Lipmann, Lu, de Paepe, Sitters, & Stougie, 2002; Bonifaci, Lipmann, Stougie, et al., 2006) and when all the origins or all the destinations are located at a depot, the problem is referred to as vehicle routing problem (Ritzinger et al., 2016).

The general representation of the dial-a-ride problems is ideally suited to represent problems faced by companies such as super shuttle (transports people from an airport to different locations in the city), uber pooling (transports customers from near by start locations to near by destination locations). These problems are hard to solve and the traditional approaches for these problems can solve only very small instances of 96 requests and 8 vehicles (Ropke, Cordeau, & Laporte, 2007).

The integer programming formulation, without any spatial or temporal aggregation (Ropke et al., 2007), is difficult to solve and is not scalable to large scale problems and online decision making even for unit-capacity. Therefore, in recent times, many heuristic approaches have been proposed to solve the OLYMPIAD problem. As shown in Figure 2.1, the existing work on OLYMPIAD problems can

be categorized along three dimensions of capacity, consideration of requests and the nature of assignment (whether it is myopic or takes future demand into account for making current assignments).

In case of U-OLYMPIAD problems, servers need to be assigned to atmost one request at a time. Greedy and randomized ranking (Karp, Vazirani, & Vazirani, 1990b) algorithms have been used in the literature to compute myopic matching when requests are considered sequentially. The myopic matching for the batch case is also trivial in this case and can be achieved by performing a bipartite matching between vehicles and requests (Agatz et al., 2011). To improve the performance of these myopic algorithms in the batch case, there has been research on providing multi-period two-stage approaches for U-OLYMPIAD problems.

Multi-period approaches for U-OLYMPIAD problems considered in literature: There has been some research on multi-period models for unit-capacity servers (Powell, 1996; Spivey & Powell, 2004; Simao et al., 2009). Similar to approaches considered in this dissertation for U-OLYMPIAD problems, they consider matching supply and demand at the level of zones, cities or areas (abstraction of locations) and not at the level of individual locations. That is to say, all demand and/or supply of a specific type within an area are deemed equivalent to improve scalability. There are multiple key differences in assumptions/constraints made in existing work on truckload assignment problem (and other similar ones) as compared to the problem of interest in this dissertation (e.g. taxi matching). These differences in assumptions are significant as the computational complexity class changes due to these assumptions/constraints.

- **Zone assignment constraints:** In the existing models used for truckload assignment problem, at any timestep, truck can only be assigned to load in the same zone ⁴. This is a strong assumption which makes the formulation a pure

⁴Some works (Godfrey & Powell, 2002) consider in the evaluation (not in the formulation) that the loads (tasks/requests) if not served will be available at the future timesteps. But, in the cases where waiting time is few minutes, i.e., demands are available only for one timestep and disappear if not assigned (for example in taxi case), using existing formulations, requests can not be served from the neighboring zones

network flow problem that can be solved in polynomial time. The complexity in truckload assignment comes due to other generalizations (such as returning trucks to their base locations, each truck serving multiple loads etc.). On the other hand, due to requirement of smaller duration (about 5 minutes) for picking up customers, zones cannot be very large and a server from a neighboring zone can be assigned to the customer in a zone (if it is the nearest server). These assumptions/constraints of our problem result in NP-hardness and thereby increase the complexity.

- **Time to compute dispatch strategies:** Since truckload tasks typically take anywhere from 1 day to 4 days, the time available to compute a dispatch strategy is in the order of tens of minutes or even an hour. In the problem of interest of this dissertation, decisions on dispatch must be decided in real-time (in less than a minute) and the number of servers is in the order of thousands and demand is in the order of at least 300-400 customers per minute.
- **Representation of current and future assignments in formulation:** The differences in the assumptions/constraints also introduce a key change in the formulations for modelling the problems. Unlike our approach, the existing work on truckload (and other similar domains) employ pure network flow formulations (i.e., only assigning demand from the same zone as supply) for both current and future time steps, or only for the future time step (Powell, 1996). This is the reason for better approximations provided by our formulation, as shown in the non-trivial improvements in results (on an average 9%) provided by our approach over Hybrid Multi-Period Two-Stage Stochastic optimization (HSS) (Powell, 1996) on three different taxi datasets.
- **Linear or piecewise linear approximation of value function:** Existing work on the Approximate Dynamic Programming in solving fleet optimization MDPs (Spivey & Powell, 2004; Simao et al., 2009; Godfrey & Powell, 2002) approximates the future value using the linear or piece-wise linear functions. While this seems to have provided good results in the truck fleet optimization prob-

lems, it is not a good approximation in the taxi fleet optimization. The value function approximation considered for fleet management problem is separable over zones but as in our case servers can be assigned from the nearby regions, the value of having one extra server in a zone will depend on the number of servers present in the nearby zones. Experimentally, our approaches provide on an average 9% improvement over ADP as well.

For M-OLYMPIAD problems, due to the complexity of finding a myopic batch assignment, most of the existing works consider sequential (i.e., one by one) assignment. Widdows *et al.* (Widdows, Lucas, Tang, & Wu, 2017) and Tang *et al.* (Tang *et al.*, 2017) propose an approach which allows 2 passengers to travel in the vehicle at the same time. It takes one request at a time and generates all feasible driver paths by inserting the pick-up and drop-off of the request in the existing driver paths. Pelzer *et al.* (Pelzer *et al.*, 2015) also allow 2 passengers to share the ride. They divide the road network into multiple partitions and limit the search space within the partition to find the match for the incoming request. Ma *et al.* (S. Ma, Zheng, & Wolfson, 2013) propose a myopic sequential matching algorithm for high capacity OLYMPIAD problems. They propose a taxi searching algorithm which uses a spatio-temporal index to quickly retrieve candidate taxis. It then uses a scheduling algorithm which after comparing the current request with each candidate taxi, insert into the schedule of taxi which minimizes additional incurred distance for the request. Other works (Y. Huang, Bastani, Jin, & Wang, 2014; Tong *et al.*, 2018; Chen *et al.*, 2018; Cheng, Xin, & Chen, 2017) also provide approaches where insertion operation is widely utilized, i.e., for each request, they find the best place to insert in a taxi's path.

While the sequential solution is faster to compute, the quality of solution obtained is typically poor, therefore, there have been works on finding a myopic batch solution. Most of the works in this case have focussed on low capacity vehicles. Zheng *et al.* (Zheng, Chen, & Ye, 2018) consider batch assignment but they only consider grouping at most two requests in a vehicle. They propose different approx-

imation and apply matching and optimization based approaches to assign vehicles to the combination of two requests. Dutta (Dutta, 2018) use a locally sensitive hashing technique to efficient group two requests together but they do not consider assignment of vehicles to requests. Brown *et al.* (Brown, 2016) propose exhaustively generating the combinations of atmost three requests from all the available requests. For capacity two vehicles, Yu *et al.* (Yu & Shen, 2019) propose an approximate dynamic programming approach which is non-myopic. They use a linear value function approximation to approximate the future effect of assignment and use spatial and temporal aggregation to group different parts of road network into a small number of regions. Their approach is not scalable to large number of locations and higher capacity vehicles.

A leading approach for M-OLYMPIAD problems was provided by Alonso *et al.* (Alonso-Mora, Samaranayake, et al., 2017). The approach is divided into two parts, where the first part constructs an RTV (Request Trip Vehicle) graph. A trip in an RTV graph corresponds to a combination of requests that is feasible (with respect to allowed delay). There is an edge between request and trip if the request is a part of the trip and there is an edge between trip and vehicle if the vehicle can serve all the requests in trip. From all allowable allocations of vehicles to trips, the second part computes an optimal allocation of vehicles to trips that minimizes delay or maximizes the number of requests served. As can be expected, this approach is limited in scalability since the set of possible trips increases exponentially with increase in number of requests and capacity of vehicles. For online execution, time limits are used for different steps.

The non-myopic approach by Alonso *et al.* (Alonso-Mora, Wallar, & Rus, 2017) is a minor extension of their myopic approach where they randomly sample 200 or 400 requests for next 30 minutes and then use those requests along with currently available requests to generate the assignments. Typically, there are 300 requests per minute so randomly sampling 200 or 400 requests for next 30 minutes does not help in improving the quality of solution. This is reflected in their results as well,

where the service rate remains approximately same as the service rate of the myopic approach. But they observe a minor decrease in the average delay experienced by the passengers. The sampled requests also increase the computational complexity of the approach and the runtime of the approach after adding sampled requests is more than the time available for assignment (duration over which requests are batched). As a result, it is not possible to use the approach for real-time assignments.

The zone path construction based approaches proposed in this work for M-OLYMPIAD problems, overcome these limitations of existing work by providing an offline-online method to generate request combinations efficiently by employing zone-paths. The future value of assignment to these zone paths is computed by considering multiple samples of future demand.

2.3 Approximate Dynamic Programming (ADP)

ADP is a framework based on the Markov Decision Problem (MDP) model for tackling large multi-period stochastic fleet optimization problems (Powell, 2007). The problem is formulated using the tuple $\langle S, A, \xi, T, O \rangle$:

S : denotes the system state with s_t denoting the state of system at decision epoch t .

A : denotes the set of all possible actions ⁵ (which satisfy the constraints on the action space) with A_t denoting the set of possible actions at decision epoch t . $a_t \in A_t$ is used to denote an action at decision epoch t .

ξ : denotes the exogenous information – the source of randomness in the system. For instance, this would correspond to demand samples defined for OLYMPIAD problems. ξ_t denotes the exogenous information (e.g., demand) at time t .

T : denotes the transition function which describes how the system state evolves over time.

O : denotes the objective function with $o_t(s_t, a_t)$ denoting the value obtained on

⁵We use action and decision interchangeably in the chapter.

applying action a_t on state s_t .

In an MDP, system evolution happens as $(s_0, a_0, s_1, a_1, s_2, \dots)$. However, in an ADP, the evolution happens as $(s_0, a_0, s_0^a, \xi_1, s_1, a_1, s_1^a, \dots, s_t, a_t, s_t^a, \dots)$, where s_t denotes the pre-decision state at decision epoch t and s_t^a ⁶ denotes the post-decision state (Powell, 2007). The transition from state s_t to s_{t+1} depends on the action vector a_t and the exogenous information ξ_{t+1} . Therefore,

$$s_{t+1} = T(s_t, a_t, \xi_{t+1})$$

Using post-decision state, this transition can be written as

$$s_t^a = T^a(s_t, a_t); s_{t+1} = T^\xi(s_t^a, \xi_{t+1})$$

Let $V_t(s_t)$ denotes the value of being in state s_t at decision epoch t , then using Bellman equation we get

$$V_t(s_t) = \max_{a_t \in A_t} (O(s_t, a_t) + \gamma \mathbb{E}[V_t(s_{t+1}) | s_t, a_t, \xi_{t+1}])$$

where γ is the discount factor. Using post-decision state, this expression can be broken down into two parts:

$$V_t(s_t) = \max_{a_t \in A_t} (O(s_t, a_t) + \gamma V_t^a(s_t^a)) \quad (2.1)$$

$$V_t^a(s_t^a) = \mathbb{E}[V_t(s_{t+1}) | s_t^a, \xi_{t+1}] \quad (2.2)$$

The advantage of this decomposition is that Equation 2.1 can be solved using an LP in fleet optimization problems. The basic idea in any ADP algorithm is to define a value function approximation around post-decision state, $V_t^a(s_t^a)$ and to update it by stepping forward through time using sample realizations of exogenous information (i.e. demand in fleet optimization that is typically observed in data).

⁶Here a is just used to indicate that it is post decision state and it does not correspond to any specific action.

For any iteration n , suppose we are in state s_t^n , the optimization problem at t can be written as

$$\hat{v}_t^n = \max_{a_t \in A_t^n} (O(s_t^n, a_t) + \gamma V_t^{a, n-1}(s_t^{a, n})) \quad (2.3)$$

Due to deterministic transitions in an iteration, the post decision state can be directly computed. \hat{v}_t^n is a sample realization of the value of being in state s_t^n , so the value function approximation is updated using

$$V_t^n(s_t^n) = (1 - \alpha_{n-1}) \cdot V_t^{n-1}(s_t^n) + \alpha_{n-1} \cdot \hat{v}_t^n \quad (2.4)$$

The above equation will give an estimate of being in pre-decision state s_t but when we are solving for decision epoch t we would use $V_{t+1}^{n-1}(s_{t+1})$ but since s_{t+1} is a random variable (depends on the exogenous information), we have to approximate the expectation. An effective alternative is to use \hat{v}_t^n to update the value of being in post decision state.

$$V_{t-1}^{a, n}(s_t^{a, n}) = (1 - \alpha_{n-1}) \cdot V_{t-1}^{n-1}(s_{t-1}^{a, n}) + \alpha_{n-1} \cdot \hat{v}_t^n \quad (2.5)$$

Please refer to Powell (Powell, 2007) for more details. Please note that if the state s_t also includes time, we can drop the subscript t from the value function, i.e., value function is only defined for states not for each time period. We use this in the NeurADP formulation in Chapter 4.

2.4 Existing Approaches for solving U-OLYMPIAD problems

In this section, we describe the existing approaches for solving OLYMPIAD for unit-capacity servers.

Algorithm 1 GreedyAlgorithm()

```
1: Initialize: assignments={},  $V' = \mathcal{V}$ ,  $D' = \mathcal{D}_r^1$ 
2: while  $V' \neq \emptyset \parallel D' \neq \emptyset$  do
3:   maxval = 0
4:    $v_1 = -1$ 
5:    $r_1 = -1$ 
6:   for each server  $v \in V'$  do
7:     if  $\sum_{l \in \mathcal{L}} \mu_v^0(l) == 1$  then
8:        $\mu_v^0(l) == 1 \Rightarrow l_v = l$ 
9:       for each request  $r \in D'$  do
10:        if  $\mathcal{T}(l_v, o_r, 0) \leq \tau \ \&\& \ C_{v, o_r, d_r} > \text{maxval}$  then
11:          maxval =  $C_{v, o_r, d_r}$ 
12:           $v_1 = v$ 
13:           $r_1 = r$ 
14:   if  $v_1 \neq -1 \ \&\& \ r_1 \neq -1$  then
15:     assignments.put( $v_1, r_1$ )
16:      $V' = V' - v_1$ 
17:      $D' = D' - r_1$ 
18:   else
19:     break
```

2.4.1 Greedy Algorithm (GD)

In each step, the greedy Algorithm assigns the demand element to an available server which provides the maximum marginal gain. The algorithm 1 provides the steps for greedy algorithm.

2.4.2 Randomized Greedy Algorithm (RGD)

Randomized greedy algorithm perturbs the value of matching by multiplying it by a random number w between 0 and 1. It then greedily matches the servers to the customer request by using the perturbed value. The step 10 and 11 in the greedy algorithm is modified as follows:

$$w = \text{random}(0, 1)$$

$$\text{if}(\mathcal{T}(l_v, o_r, 0) \leq \tau \ \&\& \ w \cdot C_{v, o_r, d_r} > \text{maxval}) \Rightarrow \text{maxval} = w \cdot C_{v, o_r, d_r}$$

OS:

$$\max \sum_{v \in \mathcal{V}} \sum_{r \in \mathcal{D}_r^1} C_{v,o_r,d_r} \cdot x_{vr} \quad (2.6)$$

$$\sum_{r \in \mathcal{D}_r^1} x_{vr} \leq 1 \quad \forall v \in V \quad (2.7)$$

$$\sum_v x_{vr} \leq 1 \quad \forall r \in \mathcal{D}_r^1 \quad (2.8)$$

$$x_{vr} \in \{0, 1\} \quad (2.9)$$

Table 2.1: One Step Bipartite Matching

2.4.3 One-Step Bipartite Matching (OS)

One Step Bipartite matching matches the available servers with the demand elements using standard bipartite matching optimization formulation as shown in table 2.1. x_{vr} denotes that the server v is assigned to the request r . Constraints in the formulation ensure that each server is assigned to at most one request and each request is assigned to at most one server.

2.4.4 Hybrid Multi-Period Two-Stage Stochastic Optimization (HSS)

In this section, we provide the details and the formulation of the model used by Powell *et.al.* (Powell, 1996). This formulation solves the assignment problem at first stage and a pure network flow approximation at the future stages. As for the truckload problem, the typical value of decision epoch is one day, therefore, even after having an assignment problem at first stage, this formulation will be a pure network for the truckload assignment using a single sample of future demand. As we described in related work (section 2.2), in our case, due to smaller decision epochs and specifically as the completion decision epoch of the customer request depends on the server assigned, this formulation will not be a pure network. But using this formulation for our case, allows us to measure the impact of using more accurate information for future demands. As opposed to the formulation and experimental

evaluation in (Powell, 1996), we use multiple samples of future demand for this approach.

Table 2.2 presents the Hybrid Multi-Period Two-Stage Stochastic optimization formulation for our problem. The variable x_{vr} denotes that a server v is assigned to the demand element r . We set x_{vr} to 0 if the server v can not reach the pick-up location of the request r within τ duration. u_v^1 denotes that the server v is held at its initial location.

HSS uses large zones or regions for the future timesteps. We abstract the locations in set \mathcal{L} into a set of large zones \mathcal{Z}_l . Let $z_{mn}^{t,k}$ denotes the number of servers moving from the large zone m to the large zone n at decision epoch t in the sample k and $y_i^{t,k}$ denotes the number of servers held at the large zone i at decision epoch t in sample k . $\delta_{ij}^{t,t'}$ is a binary constant which is 1 if the server starting at the decision epoch t from the large zone i reaches larger zone j exactly at the decision epoch t' .

We use o_v^z to denote the initial large zone of server v and o_j^z and d_j^z to denote the origin and destination large zones of the demand element $j \in \mathcal{D}_r^1$. We use $\xi_t^{D,k,z}$ to denote the future demand samples where each demand element has the origin and destination as a large zone.

Constraints (2.11) ensure that either the server is assigned to at most one request or it is held at its initial position. Constraint (2.12) ensures that a request is assigned to at most one server. Constraints (2.13) ensure that for any sample at any decision epoch, the number of servers moving between large zones is equal to the number of available requests between those pair of large zones. Constraints (2.14)-(2.15) ensure that the number of servers held at any large zone at any decision epoch is the difference between the number of available servers in the large zone and the number of assigned servers from the large zone.

2.4.5 Approximate Dynamic Programming (ADP)

In this section, we provide the Approximate Dynamic Programming formulation for U-OLYMPIAD problems. The modeling is similar to the stochastic dynamic re-

HSS:

$$\max \sum_{v \in \mathcal{V}} \sum_{j \in \mathcal{D}_r^1} \mathcal{C}_{v,o_j,d_j}^1 \cdot x_{vj}^1 + \frac{1}{|\xi^D|} \sum_{k \leq |\xi^D|} \sum_{t=2}^{Q+1} \sum_{j \in \xi_t^{D,k,z}} \mathcal{C}_{o_j,o_j,d_j}^t \cdot z_{o_j,d_j}^{t,k} \quad (2.10)$$

$$\mathbf{s.t.} \quad \sum_{j \in \mathcal{D}_r^1} x_{vj}^1 + u_v^1 = 1 \quad \forall v \in \mathcal{V} \quad (2.11)$$

$$\sum_{v \in \mathcal{V}} x_{vj}^1 \leq 1 \quad \forall j \in \mathcal{D}_r^1 \quad (2.12)$$

$$z_{o_j,d_j}^{t,k} \leq R_j^{t,k} \quad \forall k \leq |\xi^D|, j \in \xi_t^{D,k,z}, \forall t > 1 \quad (2.13)$$

$$\begin{aligned} y_i^{t,k} &= y_i^{t-1,k} + \sum_{t'=2}^{t-1} \sum_{j \in \xi_{t'}^{D,k,z}, d_j = i} z_{o_j,i}^{t',k} \cdot \delta_{o_j,i}^{t',t} + \mathcal{N}_i^t + \sum_{v \in \mathcal{V}} \sum_{\substack{j \in \mathcal{D}_r^1, \\ d_j^z = i}} x_{vj}^1 \cdot \delta_{o_v^z,i}^{1,t} \\ &\quad - \sum_{\substack{j \in \xi_t^{D,k,z}, \\ o_j = i}} z_{i,d_j}^{t,k} \quad \forall i \in \mathcal{L}, k \leq |\xi^D|, \forall t > 2 \end{aligned} \quad (2.14)$$

$$\begin{aligned} y_i^{2,k} &= \sum_{v \in \mathcal{V}, o_v = i} u_v^1 + \mathcal{N}_i^1 + \sum_{v \in \mathcal{V}} \sum_{\substack{j \in \mathcal{D}_r^1, \\ d_j^z = i}} x_{vj}^1 \cdot \delta_{o_v^z,i}^{1,2} \\ &\quad - \sum_{\substack{j \in \xi_t^{D,k,z}, \\ o_j = i}} z_{i,d_j}^{2,k} \quad \forall i \in \mathcal{L}, k \leq |\xi^D| \end{aligned} \quad (2.15)$$

$$x_{vj}^1 \in \{0, 1\} \quad \forall v \in \mathcal{V}, j \in \mathcal{D}_r^1 \quad (2.16)$$

$$u_v^1 \in \{0, 1\} \quad \forall v \in \mathcal{V} \quad (2.17)$$

$$z_{o_j,d_j}^{t,k} \quad \text{non - negative integer} \quad \forall k \leq |\xi^D|, j \in \xi_t^{D,k,z}, \forall t > 1 \quad (2.18)$$

$$y_i^{t,k} \quad \text{non - negative integer} \quad \forall i \in \mathcal{L}, \forall k \leq |\xi^D|, \forall t > 1 \quad (2.19)$$

Table 2.2: HSS

source allocation formulation presented by Godfrey *et.al.* (Godfrey & Powell, 2002) for travel time spanning multiple timesteps. Due to large state space and as the value function approximations are updated using tabular methods, using offline methods to learn the value of each possible state is not possible and it leaves most of the states with 0 values. Therefore, we also update the value function approximations using multiple samples (ξ^D in OLYMPIAD model) online. So the states which are reachable based on requests in D_r^1 get updated. Therefore, in iteration k we use the demand elements from $\xi_t^{D,k}$ to populate the D_t element defined below.

We first define some basic elements.

B_t = the total number of servers that we know about at decision epoch t .

$B_{tt'}$ = the total number of servers that we know about at decision epoch t but will become available at decision epoch t' , with $B_{tt'}^i$ denoting the number of servers in the location i ⁷.

D_t = the customer demand at decision epoch t .

Therefore, the system state is given by

$$s_t = (B_t, D_t)$$

x_{ij}^t are the decision variables denoting the number of servers assigned to the demand element j at decision epoch t . y_i^t are the intermediate variables denoting the number of servers in the location i at decision epoch t .

2.4.5.1 Value Function Approximation

The value function defined in section 2.3 is replaced by a suitable approximation. In general, the value function approximation is defined in terms of the servers, i.e., the post decision state is considered to only depend on B_t and not on D_t . It works well when the requests should be served as soon as possible and in case the requests are not served they will not be available at the next decision epoch. This holds true

⁷We can relate this with the variables in OLYMPIAD definition, if t represent current decision epoch then $B_{tt'}^i = \sum_v \mu_v^{t'}(i)$

for OLYMPIAD problems where if the demand is unserved, it is removed from the system. Therefore, the value function approximation in terms of post decision state can be written as $V_t^a(B_t)$

The problem is solved by executing a forward pass through time, determining the set of decisions for a single sample at a time. Godfrey *et.al.* (Godfrey & Powell, 2002) proposed a separable approximation, for the travel time spanning multiple timesteps, of the form

$$V_t^a(B_t) = \sum_{i \in \mathcal{L}} \sum_{t'=1}^{Q+1} \left[V_{t,t+t'}^{a,i}(B_{t,t+t'}^i) \right]$$

As the travel times span multiple timesteps, some servers are next available for assignment at decision epoch $t + 1$, some at decision epoch $t + 2$ and so on. The function $V_{t,t+t'}^{a,i}(B_{t,t+t'}^i)$ estimates the expected value of only those servers which are next available for assignment at decision epoch $t + t'$. After taking the sum over all possible travel times and all locations, we get the estimated future contribution of all the servers.

Here, we would like to highlight that in our case, as the servers can be assigned from nearby locations, the value function may not remain separable across locations as the value of having an extra resource in location i will depend on the number of resources in nearby locations.

But it is still possible to assume the separation and apply these approximations for U-OLYMPIAD. Linear and piecewise linear value function approximations are typically used to solve these problems as the subproblem has network structure but as we show in the following subsections, in our case the linear and piecewise value function approximation do not have a network subproblem.

Linear Value Function Approximation Linear value function approximation generally works well when the value of function is linear in terms of the number of resources or the number of resources in each state can be either 0 or 1. But even when this is not the case, sometimes linear approximations can work reasonably well. $V_t(B_t) = \sum_{i \in \mathcal{L}} \sum_{t'=1}^{Q+1} \left[V_{t,t+t'}^{a,i}(B_{t,t+t'}^i) \right]$ where each of $V_{t,t+t'}^{a,i}(B_{t,t+t'}^i)$ is given

by $v_{t,t+t'}^{i,k} \cdot B_{t,t+t'}^i$ where $v_{t,t+t'}^{i,k}$ is the slope of the approximation at k^{th} iteration.

The formulation which is solved at each decision epoch for each sample is provided in the Table 2.3. $\delta_{ij}^{t,t'}$ are binary constants denoting that the location i server assigned to demand element j at decision epoch t will become available again for assignment at decision epoch t' .

As we can see, the formulation does not have a network structure due to the presence of binary constants δ in equations (2.23) and (2.24).

Updating the value function

The slopes are updated in each forward simulation using the standard update equation as follows

$$v_{t,t+t'}^{i,k} = (1 - \alpha_{k-1}) \cdot v_{t,t+t'}^{i,k-1} + \alpha_{k-1} \cdot \pi_{t,t+t'}^i$$

where $\pi_{t,t+t'}^i$ are the dual variable corresponding to equations 2.23 and 2.24. We use the dual values obtained after solving the linear relaxation. α_{k-1} is the step size. We use the stepsize as $\frac{2}{4+k}$ as mentioned in (Godfrey & Powell, 2002). For the case when travel time spans multiple timesteps, we perform adjustment to the dual values using dual next as proposed in (Godfrey & Powell, 2002).

The complete algorithm is presented in the Algorithm 2.

Algorithm 2 ADPLinear()

- 1: Initialize: Set $v_{t,t+t'}^i, \forall t' = 1, \dots, Q + 1 \ \forall i, \forall t$
 - 2: **Forward Simulation:**
 - 3: **for** each sample s **do**
 - 4: **for** $t = 1, \dots, Q + 1$ **do**
 - 5: Solve the optimization in the Table 2.3
 - 6: Store dual vectors $\pi_{t,t+t'}^i, \forall t'$
 - 7: **for** $t = 1, \dots, Q + 1$ **do**
 - 8: **for** $t' = 1, \dots, Q + 1$ **do**
 - 9: Compute adjusted marginal contribution $\pi_{t,t'}^i$ using DualNext.
 - 10: Update $v_{t+1,t'}^{i,n} = (1 - \alpha_{n-1}) \cdot v_{t+1,t'}^{i,n-1} + \alpha_{n-1} \cdot \pi_{t+1,t'}^{i,n}$
-

Piecewise Linear Value Function Approximation When we need to estimate the value of having a quantity of some resource and the function is piecewise linear, i.e.,

$\max \sum_{i \in \mathcal{L}} \sum_{j \in D_i} c_{i,o_j,d_j}^t \cdot x_{ij}^t + \sum_{t'=t+1}^{Q+1} v_i^{t+1,t'} \cdot y_i^{t'} \quad (2.20)$
$\sum_i x_{ij}^t \leq R_j^t \quad \forall j \in D^{t,+} \quad (2.21)$
$\sum_j x_{ij}^t \leq B_{tt}^i \quad \forall i, t \quad (2.22)$
$y_i^{t+1} = B_{t't}^i + B_{tt}^i - \sum_j x_{ij}^t + \sum_m \sum_{j,d_j=i} x_{mj}^t \cdot \delta_{mj}^{t,t'} \quad \forall i, t' > t \quad (2.23)$
$y_i^{t'} = B_{t't}^i + \sum_m \sum_{j,d_j=i} x_{mj}^t \cdot \delta_{mj}^{t,t'} \quad \forall i, t' > t + 1 \quad (2.24)$
$x_{ij}^t \quad \text{non - negative integer} \quad \forall i, j \quad (2.25)$
$y_i^{t'} \quad \text{non - negative integer} \quad \forall i', t' > t \quad (2.26)$

Table 2.3: Formulation using the Linear value function approximation

the slopes of the function are monotonically increasing or decreasing, piecewise linear value function approximations are used. This is more suitable approximation for U-OLYMPIAD as the value of function increases linearly on having more resources but after reaching a threshold (when the number of servers is sufficient to serve the demand), the value remains constant. Therefore, the slope of the value function is monotonic decreasing, i.e., we have a piecewise linear concave function.

Due to the presence of large number of servers ($|N|$ varies from 1000 to 8000 in our experiments) and the need to execute the formulation multiple times (multiple decision epochs and multiple samples), it is not possible to store all the value function values by learning offline or to execute this approximation in real-time. We also tried using the alternate formulation which defines one variable for each component of the piecewise approximation instead of having the number of variables equal to the number of servers but as we need the dual values to update the value function approximation, we need to solve the linear relaxation of the formulation. The linear relaxation of this modified formulation provides worse result than the linear approximation presented previously. Therefore, we only compared against linear value function approximation.

2.5 Existing Approaches for Solving M-OLYMPIAD problems

In this section we describe a leading approach for taxi ridesharing provided by Alonso *et al.* (Alonso-Mora, Samaranayake, et al., 2017). We call the approach as Trip Based Formulation as the approach generates trips and assigns the servers to the trips.

2.5.1 Trip Based Formulation (TBF)

Following are the steps taken in this approach to perform assignment of trips to servers.

1. Construction of RV Graph: RV graph constructs a graph with nodes as the current available requests in \mathcal{D}_r^1 and servers/vehicles in the set \mathcal{V} . This graph represents which requests and servers might be pairwise-shared. Two requests $r_1 \in \mathcal{D}_r^1$ and $r_2 \in \mathcal{D}_r^1$ are connected if an empty virtual server starting at the origin location of one of the request can pick-up and drop-off both requests subject to the values of τ and λ . Similarly, a server v and a request r are connected if the request can be served by the server following the time window constraints.
2. Construction of RTV Graph: The second step of the method is to explore the regions of the RV-graph for which its induced subgraph is complete to find feasible trips. A trip is a set of requests, for example, $(r_1, r_2, r_3, \dots, r_n)$ denotes the trip of size n . A trip is feasible if all the requests in the trip can be combined, picked-up and dropped-off by some server while satisfying the value of τ and λ for all the requests. A request can be a part of several feasible trips of varying size, and a trip might be connected to multiple servers. The RTV-graph contains edges between a request and a trip and between a server and a trip. The algorithm to compute the feasible trips and edges proceeds

TBF:

$$\min \sum_j y_j \quad (2.27)$$

$$\sum_{p \in Tr_v} x_{vp} \leq 1 \quad \forall v \in V \quad (2.28)$$

$$\sum_v \sum_{p \in Tr_j \cap Tr_v} x_{vp} + y_j = 1 \quad \forall j \in \mathcal{D}_r^1 \quad (2.29)$$

$$x_{vp} \in \{0, 1\}, y_j \in \{0, 1\} \quad (2.30)$$

Table 2.4: TBF Formulation

incrementally in trip size for each server. This step is parallelized for each server. Let Tr denotes the set of trips generated with Tr_v denoting the set of trips which can be assigned to server v . Similarly, Tr_j denotes the set of trips which can be assigned to demand element \mathcal{D}_{rj}^1 .

3. Integer Programming formulation for assignment of Trips to servers: Once the RTV graph is constructed the integer programming formulation in table 2.4 is used to perform the assignment of trips to servers. The variable x_{vp} denotes if server v is assigned to the trip p and y_j is set to 1 if the request j is not served. We use the objective of minimizing the number of unserved requests.
4. Rebalancing of empty vehicles/servers: After the servers are assigned to the demand elements in the previous step, the unassigned servers which are not carrying any passenger are rebalanced to move towards unassigned requests. This is due to the assumption that the ignored passengers may request again, and it is likely that more requests will come from the same location in future. Given a set of empty servers and unassigned requests, the servers are assigned to requests while minimizing the total travel time of all the servers.

2.5.1.1 Heuristics for Online Computation

Following are the heuristics employed in TBF for Online Computation

1. Limit on the edges in RV Graph: After generating the complete RV graph, for each request, only 30 server edges are kept in the graph based on the time required by server to reach the origin location of request.
2. TimeLimit for each server in computation of RTV graph: In the computation of the RTV-graph, to explore potential trips, a maximum amount of time (0.2 seconds) per server is used.
3. Timelimit in solving Integer Programming Formulation: Integer Programming Formulation is solved with an optimality gap of 0.1% and a maximum run time of 15 seconds.

Chapter 3

Optimization Approaches to Solve U-OLYMPIAD Problems

In this chapter, we provide the optimization approaches to solve U-OLYMPIAD problems. We first provide the optimization formulation for the offline case. We use U-OFLYMPIAD to denote the offline problem. In section 3.2 we provide a two-stage formulation to solve the U-OLYMPIAD. We also provide a multi-period two-stage extension. To improve the scalability, we propose using the Benders decomposition approach. We then provide the key assumptions made in this work and the ways to relax them. We also provide detailed experimental results on three real world datasets. Finally, we also present the results of the real world deployment of a driver guidance system built based on the work in this chapter.

3.1 Optimization Formulation for U-OFLYMPIAD Problems

In the offline setting demand for all the timesteps is known. Therefore, we use \mathcal{D} element in OLYMPIAD model to represent demand for all timesteps with \mathcal{D}^t denoting the demand at timestep t . In this chapter, we consider that the server can be assigned to a new request, once it finishes serving the currently assigned requests.

Therefore, a server i is considered available for assignment if and only if $q_i = \phi$. In chapters 4 and 5, we relax this assumption.

We provide the integer linear optimization formulation for U-OFLYMPIAD problems in Table 3.1. Intuitively, the goal of the formulation is to set values for the assignment (of servers to demand) variables while ensuring the flow of servers between locations and the corresponding assignment to demand are valid. Depending on whether the Markovian property for resource allocation is satisfied, the complexity of this formulation can either be polynomial time or NP-Hard. We provide a detailed discussion on this aspect in Section 3.1.1.

The notations used in the formulation (apart from the OLYMPIAD model elements described in previous chapter) are as follows:

1. As for unit-capacity case, all servers present in the same location are homogeneous, in the objective function we can replace the server with the location, i.e., \mathcal{C}_{i,o_j,d_j} denotes the objective value obtained on assigning location i server to demand element j .
2. \mathcal{N}_i^t indicates the number of servers which first become available at location l at decision epoch t , i.e., $\mathcal{N}_i^t = \sum_v \mu_v^t(i)$. For the offline case all servers are available initially, therefore, $\mathcal{N}_i^t = 0, \forall i, t > 1$.
3. $f(l, t)$ gives the list of locations which can be reached from l within a duration τ at decision epoch t , i.e., $l' \in f(l, t)$ if and only if $\mathcal{T}(l', l, t) \leq \tau$. This ensures that waiting time for any request is less than τ .
4. $g(l, t)$ provides the list of originating locations for requests that can be assigned to a server in location l at decision epoch t , i.e., $l' \in g(l, t) \implies l \in f(l', t)$.
5. M is the total number of decision epochs.
6. $\delta_{ij}^{t,t'}$ indicates whether location i server assigned to a single request belonging to the element j of \mathcal{D}^t at decision epoch t completes its trip exactly at decision

U-OFLYMPIAD($M, \mathcal{L}, \mathcal{D}, \mathcal{N}, \mathcal{C}, f, g, T, \delta$):

$$\max \sum_{t=1}^M \sum_{i \in \mathcal{L}} \sum_{\substack{j \in \mathcal{D}^t, \\ o_j \in g(i,t)}} C_{i,o_j,d_j}^t \cdot x_{ij}^t \quad (3.2)$$

$$\mathbf{s.t.} \sum_{\substack{j \in \mathcal{D}^1, \\ o_j \in g(i,1)}} x_{ij}^1 \leq \mathcal{N}_i^1 \quad \forall i \quad (3.3)$$

$$\sum_{i \in f(o_j,t)} x_{ij}^t \leq R_j^t \quad \forall t, j \in \mathcal{D}^t \quad (3.4)$$

$$\begin{aligned} \sum_{\substack{j \in \mathcal{D}^t, \\ o_j \in g(i,t)}} x_{ij}^t &\leq \mathcal{N}_i^1 - \sum_{t'=1}^{t-1} \sum_{\substack{j \in \mathcal{D}^{t'}, \\ o_j \in g(i,t')}} x_{ij}^{t'} \\ &+ \sum_{t'=1}^{t-1} \sum_{t''=t'+1}^t \sum_{\substack{j \in \mathcal{D}^{t'}, \\ d_j=i}} \sum_{i' \in f(o_j,t')} \delta_{i',j}^{t',t''} \cdot x_{i'j}^{t''} \quad \forall i, t > 1 \end{aligned} \quad (3.5)$$

$$x_{ij}^t \quad \text{non - negative integer} \quad \forall i, j, t \quad (3.6)$$

Table 3.1: U-OFLYMPIAD formulation

epoch t' . The formal definition is as follows:

$$\delta_{ij}^{t,t'} = \begin{cases} 1 & \mathbf{if} \ t + \lfloor \frac{\mathcal{T}(i,o_j,t) + \mathcal{T}(o_j,d_j,t)}{\Delta} \rfloor + 1 = t', \\ 0 & \mathbf{otherwise} \end{cases} \quad (3.1)$$

7. x_{ij}^t is the integer variable which denotes the number of location i servers assigned to the element $j \in \mathcal{D}^t$ at decision epoch t .

We also provide an example in Section 3.1.2 to explain these notations. The complexity of the formulation is determined by the assumptions on the underlying binary constants $\delta_{ij}^{t,t'}$, $1 \leq t \leq t' \leq M$ (refer to Section 3.1.1).

In the optimization formulation, the number of servers available in any location, at any decision epoch depends on assignments at previous decision epochs. Constraints (3.3) and (3.5) ensure that at any decision epoch t , the number of servers assigned from a location is less than the number of available servers in the location at decision epoch t . Constraint (3.4) ensures that for each element in \mathcal{D}^t , the number

of requests assigned to servers is less than the number of available requests between origin and destination location pairs corresponding to element j of \mathcal{D}^t .

3.1.1 Discussion on $\delta_{ij}^{t,t'}$

We can observe in equation (3.1) that the values of binary constants $\delta_{ij}^{t,t'}$ depends on the total time taken by a location i server to reach destination location of element j of \mathcal{D}^t . Therefore, as shown in the Figure 3.1, we have two cases

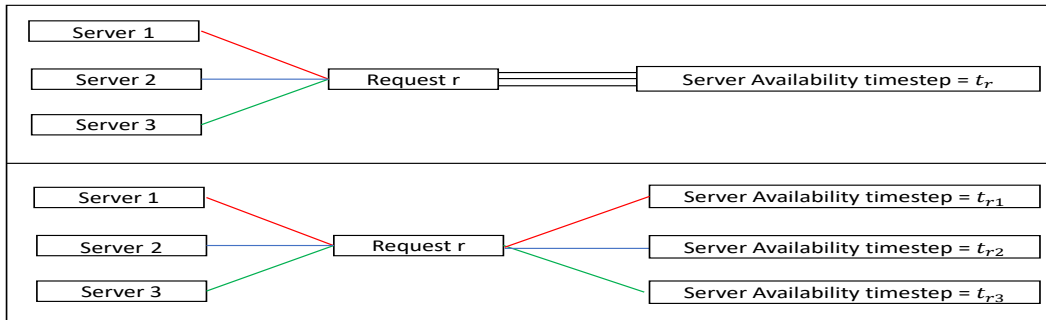


Figure 3.1: Cases for server assignment

- The time at which server becomes available again after serving a request is independent of the assigned server.
- The time at which server becomes available again after serving a request is dependent of the assigned server. For example, suppose server A assigned to a request r at $t = 1$ becomes available again for assignment at $t=4$. Now if request r is served by another server B at $t=1$, then it is possible that B will still be serving the request r at $t=4$ and becomes available again at $t = 6$. One of the cases where this is possible is when the time taken by different servers to reach pick-up location of request is different. In the above example A reaches pick-up location of r at $t = 2$ and B reaches pick-up location of r at $t = 4$.

Case 1 is similar to the Markov property defined in the context of resource allocation problem¹ and makes the problem polynomial time solvable as the formulation

¹please refer to chapter 13 of the book by Powell (Powell, 2007), which states that attributes of a transition resulting from a decision acting on a resource with attribute r is independent of r (i.e., the transition is memoryless). In the context of resource allocation problem, it is mentioned that when system satisfies the Markov property, then a network formulation is obtained otherwise the formu-

is equivalent to a minimum cost network flow problem. Formally stating the above discussion, for the special case when $\delta_{ij}^{t,t'} = \delta_{i'j}^{t,t'}$, $\forall j, t, t', i, i'$ (i.e., a request completion decision epoch is same irrespective of the server assigned to it), we show the reduction of the U-OFLYMPIAD to a min cost flow problem with integer capacities (Proposition 1). As the min cost flow problem is polynomial time solvable, the U-OFLYMPIAD is also polynomial time solvable for this special case.

Proposition 1. *If $\forall j, i, i', t, t' \delta_{ij}^{t,t'} = \delta_{i'j}^{t,t'}$, then the U-OFLYMPIAD is reducible to a min cost flow problem.*

Proof: See A.1. ■

For case 2 (i.e., for at least one setting of j, i, i', t, t' if $\delta_{ij}^{t,t'} \neq \delta_{i'j}^{t,t'}$), we show that the U-OFLYMPIAD is NP-hard by reducing the well known 3-SAT problem to it (Proposition 2).

Proposition 2. *If $\exists j, i, i', t, t'$ s.t. $\delta_{ij}^{t,t'} \neq \delta_{i'j}^{t,t'}$, then the U-OFLYMPIAD is NP-hard.*

Proof: See A.2. ■

Relaxation of Optimization Formulation As shown in Proposition 1, if special condition on δ values holds, the problem is reducible to min cost flow. Therefore, we will get the integer optimal solution even if we relax the integrality constraints in the integer program for the U-OFLYMPIAD.

For the general case, we do not have any theoretical guarantees on the linear relaxation of the integer program. But in our experiments, we observed that the linear relaxation of the U-OFLYMPIAD integer program is tight and the difference between the optimal value of the integer program and the relaxed linear program is always less than 1%. Therefore, the objective value obtained on solving the relaxation of the U-OFLYMPIAD, provides a tight upper bound on the value of the

relaxation is not integral but there is no theoretical proof provided. We also obtain a similar result and provide theoretical proofs for both cases. In (Powell, 2007) authors suggest to use aggregation over attributes to restore the Markov property (i.e., network structure) but as we show in our experimental results, this does not give good results in our case.

Shape	Meaning	Notation	In the Shape
Circle	Location	i	Location number
Black Hexagon	Initial servers	\mathcal{N}_i^0	Number of servers associated with the location next to the hexagon at the decision epoch 1
Black Diamond	Request	j	Request index for the rectangle below the diamond
Black Rectangle	Demand	$\langle o_j, d_j, R_j^t \rangle$	\langle source location, destination location, # of requests from source to destination \rangle
Black Line	Assignment	x_{ij}^t	Line between location i and demand j at decision epoch t indicates the possible assignment of location i server to the demand j .
Green Numbers	Revenue	C_{i,o_j,d_j}^t	Revenue obtained on performing the assignment indicated by the line below the number.
Black Dashed Line	Binary Constants	$\delta_{ij}^{t,t'}$	Dashed line from the line between location i and demand j at decision epoch t to the destination location d_j at the decision epoch t' indicates that $\delta_{ij}^{t,t'} = 1$. Absence of the dashed line indicates $\delta_{ij}^{t,t'} = 0$.

optimal solution. In addition, a major part of the solution has integer values. A simple heuristic of taking the integer part of the solution satisfying all constraints provides a solution which is nearly 95% of the optimal integer solution. In Section 3.2, we also provide a heuristic to extract an integral solution from the LP relaxation solution.

3.1.2 Example

In this section, we show an example that explains the different parameters and elements of U-OFLYMPIAD model.

Example 1. *We consider a 3 location problem and show request assignment over 3 decision epochs (i.e., $M = 3$) in Figure 3.2. The following table provides a mapping between formal notation and the visual depiction of the U-OFLYMPIAD problem. For ease of explanation, we assume that the travel times between locations are same at all decision epochs, so we drop index t from the definition of T , f and g . Travel time between each pair of location is mentioned at the bottom of the Figure 3.2. This represents the set T . The value of τ and Δ is taken as 5 (min). As mentioned in the model, the sets f and g are populated based on the travel time and τ values. Therefore, $f(2)$ will contain all locations which are reachable from 1 in less than 5 minutes, $f(2) = 1, 2, 3$. Similarly $g(2)$ contains all locations from which it takes less than 5 minutes to reach location 2, therefore, $g(2) = 1, 2$. All other values are*

populated in the same way.

The server availability at decision epochs 2 and 3 will depend on the assignments at the decision epoch 1. The server, if assigned, will be available in the destination location of the requests at the decision epoch where the dashed line from the corresponding assignment line ends. Unassigned servers will be available in the same location at next decision epoch.

We can also observe from this example, the importance of considering multi-period two-stage matching. For this, we compare the revenue earned by two kinds of decisions

1. Single-stage decision, i.e., decisions which do not consider future requests:

In this case, at decision epoch 1 location 1 server will be assigned to the demand element 2 and location 2 server will be assigned to the demand element 1, i.e., $x_{12}^1 = 1, x_{21}^1 = 1$ and rest all variables as 0. This will give a total revenue of 20+14. As none of the servers will be available at decision epoch 2, no other requests can be served.

2. Multi-period two-stage decision, i.e., decisions which consider future requests:

In this case, at decision epoch 1, location 1 server will be assigned to the demand element 1 and location 2 server will not serve any request, i.e., $x_{11}^1 = 1$ and rest all variables for decision epoch 1 are set to 0. At decision epoch 2 both servers will be available in the location 2 and can serve both requests, i.e., $x_{21}^2 = 1$ and $x_{22}^2 = 1$. This will give a total revenue of 15+15+10 = 40.

We have presented the model for the case where the exact demand information is available for all decision epochs. In the next section, we extend this model to consider uncertain customer demand which is revealed in an online fashion.

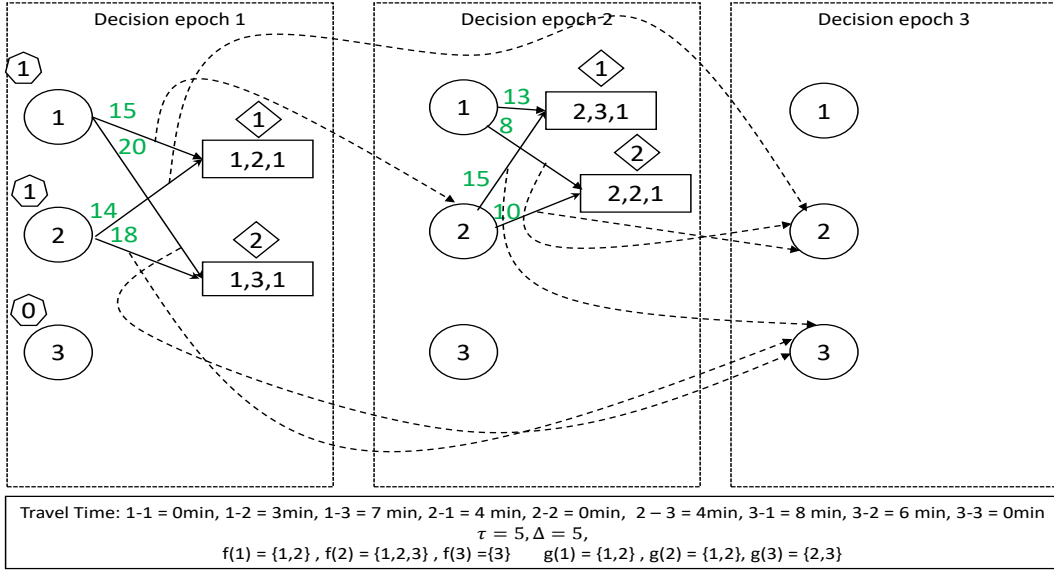


Figure 3.2: U-OLYMPIAD Example

3.2 Optimization formulation for U-OLYMPIAD

We now extend the optimization formulation described in previous section for the online scenario. In an online setting requests are only available for current decision epoch. The goal is to make matching decisions at each decision epoch so as to optimize the overall objective.

Similar to the binary constants δ defined in the optimization formulation for U-OLYMPIAD problem, we will have binary constants $\delta_{ij}^{t,t',k}$ ($t > 1$), which are set to 1 if location i server assigned to a single request belonging to the element j of $\xi^{D,k,t}$ at decision epoch t completes its trip exactly at decision epoch t' . For ease of explanation, we first describe the formulation for a two-stage model ($Q = 1$, one timestep in future), where we only consider requests at the next decision epoch in making matching decisions at the current decision epoch.

Our goal is to identify the assignment of requests to the servers so as to maximize the sum of objective values for the current decision epoch and the expected objective value for the next decision epoch. An integer linear optimization formulation is provided in the Table 3.2 for computing the best match at the current decision epoch while considering multiple samples of potential requests in the next decision epoch. We refer to this as **TSS()**. Integer variables x_{ij}^1 denote the number of location

TSS($\mathcal{L}, \mathcal{D}, \mathcal{N}, \mathcal{C}, \xi^D, f, g, T, \delta$):

$$\max \sum_{i \in \mathcal{L}} \sum_{\substack{j \in \mathcal{D}^1, \\ o_j \in g(i,1)}} c_{i,o_j,d_j}^1 \cdot x_{ij}^1 + \frac{1}{|\xi^D|} \sum_{k \leq |\xi^D|} \sum_{i \in \mathcal{L}} \sum_{\substack{j \in \xi_2^{D,k}, \\ o_j \in g(i,2)}} c_{i,o_j,d_j}^2 \cdot x_{ij}^{2,k} \quad (3.7)$$

$$\text{s.t.} \quad \sum_{\substack{j \in \mathcal{D}^1, \\ o_j \in g(i,1)}} x_{ij}^1 \leq \mathcal{N}_i^1 \quad \forall i \in \mathcal{L} \quad (3.8)$$

$$\sum_{i \in f(o_j,1)} x_{ij}^1 \leq R_j^1 \quad \forall j \in \mathcal{D}^1 \quad (3.9)$$

$$\sum_{\substack{j \in \xi_2^{D,k}, \\ o_j \in g(i,2)}} x_{ij}^{2,k} \leq \mathcal{N}_i^1 - \sum_{j \in \mathcal{D}^1} x_{ij}^1 + \mathcal{N}_i^2 + \sum_{j \in \mathcal{D}^1, d_j=i} \sum_{i' \in f(o_j,1)} \delta_{i'i}^{1,2} \cdot x_{i'j}^1 \quad (3.10)$$

$\forall i \in \mathcal{L}, k \leq |\xi^D|$

$$\sum_{i \in f(o_j,2)} x_{ij}^{2,k} \leq R_j^{2,k} \quad \forall k \leq |\xi^D|, j \in \xi_2^{D,k} \quad (3.11)$$

$$x_{ij}^1 \quad \text{non-negative integer} \quad \forall i \in \mathcal{L}, j \in \mathcal{D}^1 \quad (3.12)$$

$$x_{ij}^{2,k} \quad \text{non-negative integer} \quad \forall i \in \mathcal{L}, \forall k \leq |\xi^D|, j \in \xi_2^{D,k} \quad (3.13)$$

Table 3.2: TSS optimization formulation

i servers assigned to element $j \in \mathcal{D}^1$. Similarly, integer variables $x_{ij}^{2,k}$ denote the number of location i servers assigned to the element $j \in \xi_2^{D,k}$.

While the first component of the objective value corresponds to the current decision epoch, the second component computes the expected value associated with the future requests (provided in ξ^D). Constraints (3.8) and (3.10) ensure that at any decision epoch, the number of assigned servers from the location i is less than the number of available servers. In Constraint (3.10) the number of servers available at decision epoch ‘‘2’’ in location i is calculated by considering the remaining servers in location i after doing assignments for decision epoch ‘‘1’’. Constraints (3.9) and (3.11) ensure that at any decision epoch, between any location pair, the number of requests assigned to servers is less than the number of available requests between the origin and destination location pair. For a single sample, the above integer program is equivalent to an U-OLYMPIAD integer program with $M=2$. Therefore, as shown in Proposition 1, if $\delta_{i'i}^{1,2} = \delta_{ij}^{1,2} \quad \forall i, i', j$, the relaxation of TSS, for a single sample, will have an integer optimal solution.

For a general case with multiple samples, we show in Proposition 3 that **TSS** is NP-hard.

Proposition 3. *Solving **TSS** for more than one sample is an NP-hard problem irrespective of δ values.*

Proof: See A.2.1. ■

Relaxation of the **TSS** Optimization Formulation

For a general case with multiple samples, the linear relaxation of the **TSS** integer program can yield fractional solutions. But in our experiments on synthetic domains and two real world datasets, we observe that the linear relaxation of **TSS** is tight and the difference between the optimal value of the integer program and the relaxed linear program is always less than 1%. We also observe that most of the time, the solution is integral and even if the solution is not integral, major part of the solution has integer values. Therefore, our approach is to solve the relaxed version of the problem and in case the solution is not integral, we round it to an integer solution as described below.

While converting a fractional solution to an integer, only the parts of the solution that are fractional are modified. From the fractional part, variables x_{ij}^1 are rounded in such a way that the number of servers assigned from each location at the first decision epoch remain close to the fractional optimal solution. This ensures that the servers which were left unassigned by the **TSS**, remain unassigned and the assignments at the second stage are least affected. We denote F_i as $\lceil \sum_j x_{ij}^1 - \sum_j \lfloor x_{ij}^1 \rfloor \rceil$, i.e., the sum of fractional assigned servers from location i rounded up to the nearest integer. We denote G_j as $\lceil \sum_i x_{ij}^1 - \sum_i \lfloor x_{ij}^1 \rfloor \rceil$, i.e., the number of fractionally assigned requests rounded up to the nearest integer. We further divide the set G_j into two parts with the set G_{1j} containing requests which complete on or before decision epoch 2 and the set G_{2j} containing remaining requests. We greedily assign the servers available in $F_i, \forall i$ to the requests available in the set $G_j, \forall j$ in two rounds. In the first round, we only consider the requests in the set G_{1j} , i.e., we give priority

to the requests which can be completed by decision epoch 2. If after first round $\exists i, F_i > 0$, we greedily assign them to the requests in G_{2j} .

3.2.1 Example

In this section, we show an example that explains the different parameters and elements of the U-OLYMPIAD model.

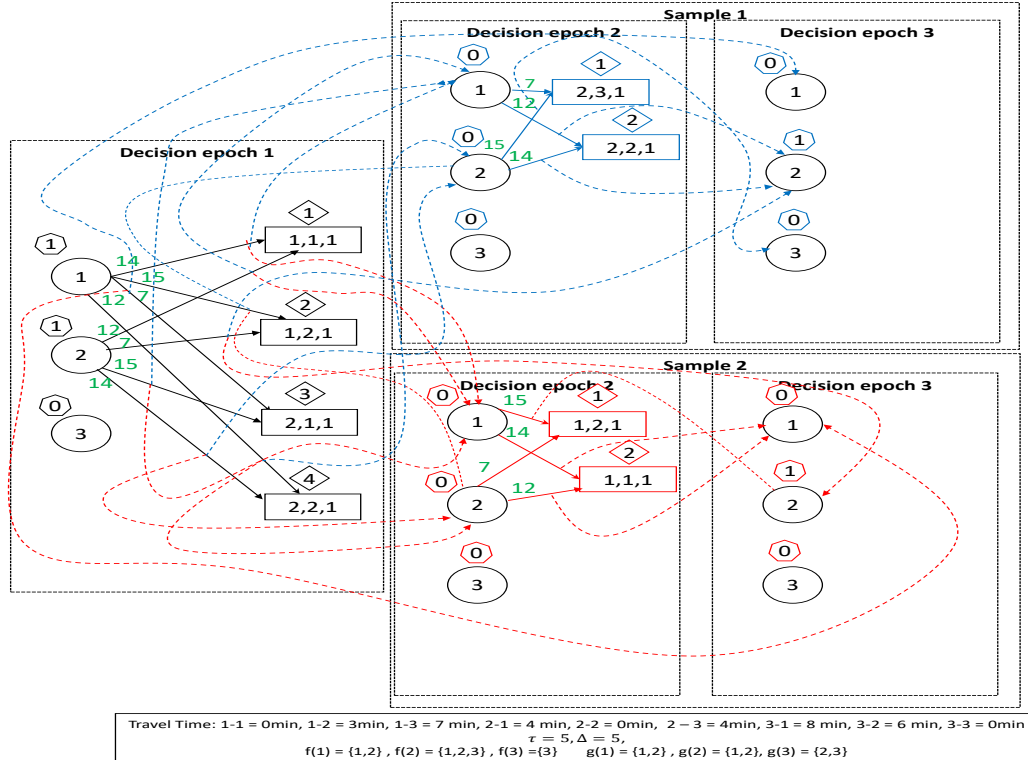


Figure 3.3: U-OLYMPIAD Example

Example 2. Similar to U-OFLYMPIAD, we consider a 3 location problem as shown in the Figure 3.3. The following table provides a mapping between the formal notation and the visual depiction of the U-OLYMPIAD problem. We only define the new elements as compared to the U-OFLYMPIAD example.

For decision epoch 1, the assignment decisions are independent of samples, but server which will become available after serving the request, will be available for all samples. Therefore, in the graphical representation, we represent it by creating

Shape	Meaning	Notation	In the Shape
Colored Hexagon	Servers	\mathcal{N}_i^t	Servers which become available in the location next to the colored hexagon at decision epoch t due to the completion of previously assigned requests. The value will be same for all the samples.
Colored Rectangle	Sampled Demand	$\langle o_j, d_j, R_j^{t,k} \rangle$	\langle source location, destination location, # of requests from the source to the destination in the sample \rangle
Colored Diamond	Sampled Request	j	Request index for the colored rectangle below the colored diamond
Colored Line	Assignment in samples	$x_{ij}^{t,k}$	Line between location i and demand element j in sample k at decision epoch t indicates possible assignment of the location i server to the demand element j in sample k .
Colored Dashed Line	Binary Constants for sample	$\delta_{ij}^{t,t',k}$	Colored Dashed line from the line between location i and the demand element j in the sample k at decision epoch t to the destination location d_j at decision epoch t' indicates that $\delta_{ij}^{t,t',k} = 1$. Absence of the dashed line indicates $\delta_{ij}^{t,t',k} = 0$.

copies of first stage δ constants for each sample and represent them with colored dash lines even for first decision epoch. All these copies will have identical values. In the formulation, this is shown by having $\delta_{ij}^{1,t'}$ and x_{ij}^1 in constraint (3.10).

The value of τ and Δ is taken as 5. Travel time T and sets f and g are populated as explained in the U-OFLYMPIAD example.

We can also observe from this example, that the optimal decision for individual samples will not remain optimal when all the samples are considered together. We show the decisions and revenue computation in following three cases:

- 1. **Only sample 1 is considered:** If only sample 1 is considered, it will be beneficial to move both servers to location 2 at decision epoch 2. Therefore, the optimal decision will be, $x_{12}^1 = 1, x_{24}^1 = 1, x_{21}^{2,1} = 1, x_{22}^{2,1} = 1$ and rest all variables as 0. This will give a total revenue of $15+14+15+14=58$.*
- 2. **Only sample 2 is considered:** Similar to above case, if only sample 2 is considered, the optimal decision will be $x_{11}^1 = 1, x_{23}^1 = 1, x_{11}^{2,1} = 1, x_{11}^{2,1} = 1$ and rest all variables as 0. This will give a total revenue of $15+14+15+14=58$.*
- 3. **Both samples are considered:** On the other hand if both samples are considered, the optimal decision will be $x_{12}^1 = 1, x_{21}^1 = 1, x_{ij}^{2,1} = 1, x_{ij}^{2,1} = 1$ and rest all variables as 0. This will give a total revenue of $14+14+ \frac{((15+12)+(15+12))}{2}$*

= 55. In this case, individual sample's optimal decision will give a revenue of $15+14+ \frac{((15+14)+(7+12))}{2} = 53$.

3.2.2 Benders Decomposition

Given the scale of the problems of interest in this dissertation (i.e., thousands of taxis serving thousands of customers spread across hundreds of locations), we reduce the complexity associated with increasing the number of samples by exploiting the following observation:

Observation 1. *In TSS, once the assignment at the first decision epoch, $\{x_{ij}^1\}$ is given, the optimization models for computing the assignment at the second decision epoch, $\{x_{ij}^{2,k}\}$ for each of the samples k , are independent of each other.*

We exploit Observation 1 by using the Benders Decomposition (Benders, 1962) method, a master slave decomposition technique where **the Master Problem** is responsible for obtaining the solutions for the “difficult” variables; and **the Slave problem(s)** is (are) responsible for finding the solutions to other variables, given a fixed assignment of values to “difficult” variables (from the master). The Slave problem(s) also generate Benders cuts, which are added to the master problem and the master problem is solved with these cuts to obtain an improved solution. This process continues until no more cuts can be added to the master problem.

Based on Observation 1, $\{x_{ij}^1\}$ are the difficult variables as they impact the values assigned to all the other variables. Therefore, the master is responsible for obtaining the assignments for the $\{x_{ij}^1\}$ variables and the slave(s) are responsible for obtaining the assignments to the $\{x_{ij}^{2,k}\}$. For the master (Table 3.3), in the optimization provided in TSS, we replace the part of the objective dealing with future variables, $\{x_{ij}^{2,k}\}$ by the recourse function $\mathcal{Q}(\{x_{ij}^1\}_{i \in \mathcal{L}, (j \in \mathcal{D}^1, o_j \in g(i,1))}, k)$ which becomes the objective function in the slave problems. The recourse function $\mathcal{Q}()$ needs to be computed for each value of x_{ij}^1 . In the slaves (Table 3.4), we consider the fixed values of x_{ij}^1 and to avoid confusion, we refer to them using the capital letter notation, X_{ij}^1 .

Master:

$$\max \sum_{i \in \mathcal{L}} \sum_{\substack{j \in \mathcal{D}^1, \\ o_j \in g(i,1)}} c_{i,o_j,d_j}^1 \cdot x_{ij}^1 + \frac{1}{|\xi^D|} \sum_{k \leq |\xi^D|} \mathcal{Q}(\{x_{ij}^1\}_{i \in \mathcal{L}, (j \in \mathcal{D}^1, o_j \in g(i,1))}, k) \quad (3.14)$$

$$\text{s.t.} \quad \sum_{\substack{j \in \mathcal{D}^1, \\ o_j \in g(i,1)}} x_{ij}^1 \leq \mathcal{N}_i^1 \quad \forall i \in \mathcal{L} \quad (3.15)$$

$$\sum_{i \in f(o_j,1)} x_{ij}^1 \leq R_j^1 \quad \forall j \in \mathcal{D}^1 \quad (3.16)$$

$$x_{ij}^1 \quad \text{non - negative integer} \quad \forall i \in \mathcal{L}, j \in \mathcal{D}^1 \quad (3.17)$$

Table 3.3: Master Formulation

Slave $\mathcal{Q}(\{X_{ij}^1\}_{i \in \mathcal{L}, j \in \mathcal{D}^1}, k)$:

$$\max \sum_{i \in \mathcal{L}} \sum_{\substack{j \in \xi_2^{D,k}, \\ o_j \in g(i,2)}} c_{i,o_j,d_j}^2 \cdot x_{ij}^{2,k} \quad (3.18)$$

$$\sum_{\substack{j \in \xi_2^{D,k}, \\ o_j \in g(i,2)}} x_{ij}^{2,k} \leq \mathcal{N}_i^1 - \sum_{\substack{j \in \mathcal{D}^1, \\ o_j \in g(i,1)}} X_{ij}^1 + \mathcal{N}_i^2 + \sum_{\substack{j \in \mathcal{D}^1, \\ d_j=i}} \sum_{i' \in f(o_j,1)} \delta_{i'j}^{1,2} \cdot X_{i'j}^1 \quad \forall i \in \mathcal{L} \quad (3.19)$$

$$\sum_{i \in f(o_j,2)} x_{ij}^{2,k} \leq R_j^{2,k} \quad \forall j \in \xi_2^{D,k} \quad (3.20)$$

$$x_{ij}^{2,k} \geq 0 \quad \forall i \in \mathcal{L}, \forall k \leq |\xi^D|, j \in \xi_2^{D,k} \quad (3.21)$$

Table 3.4: Slave Formulation

The dual² of the primal slave problems are provided in Table 3.5, where α variables are the dual variables corresponding to the constraints (3.19) and β variables are the dual variables corresponding to the constraints (3.20).

²The idea of taking a dual of a linear program is an important concept in linear programming. The linear program is typically called the 'primal' linear program. For each linear program, there is an associated linear program called its 'dual'. The dual of a linear program is obtained by creating one variable for each constraint of the primal, and having one constraint for each variable of the primal. The maximization problem is changed to minimization and the roles of the coefficients of the objective function and of the right-hand sides of the inequalities are switched. In addition, transpose of the matrix of coefficients of the left-hand side of the inequalities is taken.

That is to say for a primal problem,

$$\max_x C^T x \quad \text{s.t.} \quad Ax = B$$

we have the associated dual given by

$$\max_x B^T y \quad \text{s.t.} \quad A^T y \geq C$$

The *weak duality theorem* (Bertsimas & Tsitsiklis, 1997) states that the solution to a maximization primal problem is always less than or equal to the solution of the corresponding dual problem. Therefore, using the concept of weak duality we can say that by taking the dual of the slave problems, we can find an upper bound on the value of the recourse function ($\mathcal{Q}()$)(objective of primal slave problem), in terms of the master problem variables x_{ij}^1 . These can then be added as optimality cuts to the master problem (Murphy, 2013) for generating better first stage assignments³. Let

<p>Dual Slave ($\{X_{ij}^1\}_{i \in \mathcal{L}, j \in \mathcal{D}^1}, k$) :</p> $\min \sum_{i \in \mathcal{L}} \alpha_i^k \cdot (\mathcal{N}_i^1 - \sum_{\substack{j \in \mathcal{D}^1, \\ o_j \in g(i,1)}} X_{ij}^1 + \mathcal{N}_i^2 + \sum_{\substack{j \in \mathcal{D}^1, \\ d_j = i}} \sum_{i' \in f(o_j,1)} \delta_{i'j}^{1,2} \cdot X_{i'j}^1) + \sum_{j \in \xi_2^{D,k}} \beta_j^k \cdot R_j^{2,k} \quad (3.22)$ <p>s.t. $\alpha_i^k + \beta_j^k - \mathcal{C}_{i,o_j,d_j}^2 \geq 0 \quad \forall i \in \mathcal{L}, j \in \xi_2^{D,k} \quad (3.23)$</p> $\alpha_i^k \geq 0 \quad \forall i \in \mathcal{L} \quad (3.24)$ $\beta_j^k \geq 0 \quad \forall j \in \xi_2^{D,k} \quad (3.25)$
--

Table 3.5: Dual Slave Formulation

θ^k be the approximation of $\mathcal{Q}()$ function then the master problem with optimality cuts is provided in the Table 3.6. It should be noted that we are using x_{ij}^1 variables in the “master with optimality cuts” and not the fixed values, X_{ij}^1 . In each iteration we solve the master problem and the computed x_{ij}^1 variable values are passed to the dual slave problems. After solving the dual slave problems, optimality cuts are generated. If the current values of $\theta^k (\forall k)$ satisfy the optimality cut conditions then we have obtained an optimal solution, else cuts are added to the master problem and the master problem is solved again. As we can see in the “Dual Slave” linear programs, the slave problems are independent of each other and are only connected by the choice of the master variables (“difficult” variables). Therefore, once the master variables are fixed, the slave problems can be solved in a parallel fashion.

³As the slave problems are always feasible for any value of the master variables we only need to add optimality cuts to the master problem.

Master Formulation with Optimality Cuts:

$$\max \sum_{i \in \mathcal{L}} \sum_{\substack{j \in \mathcal{D}^1, \\ o_j \in g(i,1)}} C_{i,o_j,d_j}^1 \cdot x_{ij}^1 + \frac{1}{|\xi^{\mathcal{D}}|} \sum_{k \leq |\xi^{\mathcal{D}}|} \theta^k \quad (3.26)$$

$$\begin{aligned} \text{s.t. } \theta^k \leq & \sum_{i \in \mathcal{L}} \alpha_i^k \cdot (\mathcal{N}_i^1 - \sum_{\substack{j \in \mathcal{D}^2, \\ o_j \in g(i,1)}} x_{ij}^1 + \mathcal{N}_i^2 + \sum_{\substack{j \in \mathcal{D}^1, \\ d_j=i}} \sum_{i' \in f(o_j,1)} \delta_{i'j}^{1,2} \cdot x_{i'j}^1) \\ & + \sum_{j \in \xi_2^{D,k}} \beta_j^k \cdot R_j^{2,k} \end{aligned} \quad (3.27)$$

$$\sum_{\substack{j \in \mathcal{D}^1, \\ o_j \in g(i,1)}} x_{ij}^1 \leq \mathcal{N}_i^1 \quad \forall i \in \mathcal{L} \quad (3.28)$$

$$\sum_{i \in f(o_j,1)} x_{ij}^1 \leq R_j^1 \quad \forall j \in \mathcal{D}^1 \quad (3.29)$$

$$x_{ij}^1 \text{ non - negative integer} \quad \forall i \in \mathcal{L}, j \in \mathcal{D}^1 \quad (3.30)$$

Table 3.6: Master Formulation with Optimality Cuts

3.2.3 Complexity Analysis

The complexity of a linear program depends on the number of variables and constraints in the formulation. The maximum number of variables in the TSS formulation are $|\mathcal{L}| \cdot |\mathcal{D}^1| + \sum_k |\mathcal{L}| \cdot |\xi_2^{D,k}|$. This maximum value will be obtained when any request can be served using server from any location. But in practice depending on the value of τ , f and g function will restrict the number of variables. For $\tau = 0$, request can only be served using the server present at its origin location, therefore, the number of variables are $|\mathcal{D}^1| + \sum_k |\xi_2^{D,k}|$. As the value of τ increases, the number of variables increases. The number of constraints are $|\mathcal{L}| + |\mathcal{D}^1| + |\xi^{\mathcal{D}}| \cdot |\mathcal{L}| + \sum_k \xi_2^{D,k}$.

Therefore, on increasing the number of locations and the number of samples, both variables and constraints increase. The size of \mathcal{D}^1 and $\xi_2^{D,k}$ increases with the value of Δ . This is because Δ is the duration between 2 decision epochs, so when Δ is small, the number of requests between two decision epochs will be less. Therefore, on increasing the value of Δ , the number of variables and constraints both increase.

In addition to τ , the travel time between locations (T) also affect the number of variables. For a constant τ , if travel time between locations increases, there will

be less locations from where servers can be used to serve any requests. Hence the number of variables decrease on increasing travel time keeping the value τ constant.

3.2.4 Competitive Ratio

Competitive analysis is typically employed for evaluating the quality of online algorithms. The metric used is called as the competitive ratio (Borodin & El-Yaniv, 1998) and it compares a solution produced by an online algorithm with the best possible solution. Specifically, the competitive ratio of an online algorithm is defined as the worst-case ratio between the objective of the solution found by the algorithm and the objective of an optimal solution, which assumes all uncertain information is known beforehand. For deterministic input model and maximization problems, an online algorithm is called c -competitive if for any instance of the problem

$$\frac{ALG(I)}{OPT(I)} \geq c \quad \forall I \quad (3.31)$$

where $ALG(I)$ is the value of any given online algorithm and $OPT(I)$ is the value of the offline optimal algorithm for instance I . Equivalently, this can be written as

$$c = \inf_I \frac{ALG(I)}{OPT(I)} \quad (3.32)$$

In stochastic environments, we calculate empirical competitive ratio (competitive ratio in expectation), c_μ^D that is defined as

$$c_\mu^D = \frac{E_D[ALG(I)]}{E_D[OPT(I)]} \quad (3.33)$$

For stochastic input models where multiple distributions of uncertainty are provided, instead of computing competitive ratio in expectation, we can also calculate **expected competitive ratio** (Please refer Section 2.2.1 in (Mehta et al., 2013)). It

is computed in expectation over the randomness in the input:

$$c_\mu = \inf_D E_D \left[\frac{ALG(I)}{OPT(I)} \right] \quad (3.34)$$

where D is the distribution over instances I from which input is drawn and expected competitive ratio is the expectation over the ratio achieved by the algorithm and the optimum for that distribution.

Proposition 4. *In U-OLYMPIAD without sample information and adversarial behavior from environment⁴, when maximizing the number of requests satisfied for a fixed number of servers N , the competitive ratio, c for any deterministic b -stage algorithm (i.e., with information available up to the b^{th} decision epoch) in a M -decision epoch ($M \geq b$) problem is*

$$c \leq \frac{1}{M - b + 1}$$

Proof: See A.3. ■

Proposition 5. *In U-OLYMPIAD with sample information and stochastic behavior from environment according to the samples, when maximizing the number of requests satisfied, the expected competitive ratio, c_μ , of the TSS algorithm is*

$$c_\mu \leq \frac{3}{4 \cdot (M - 1)} + \frac{3}{4 \cdot M}$$

where M is the number of decision epochs ($M \geq 3$).

Proof: See A.4. ■

The expected competitive ratio value is typically overly pessimistic as it measures the worst case. However, the algorithms can have good average case performance, c_μ^D as demonstrated in our experimental results. We will compute this for our formulations and other one-stage approaches, by comparing the solution values obtained with the M -stage optimal offline solution. We solve the U-OFLYMPIAD

⁴Environment generates demand that creates worst case performance for algorithms decisions

integer program defined in Section 3.1 to compute the M-Stage optimal offline solution.

MSS ($\mathcal{L}, \mathcal{D}, \mathcal{N}, \mathcal{C}, \xi^D, f, g, T, \delta, Q$):	
$\max \sum_{i \in \mathcal{L}} \sum_{\substack{j \in \mathcal{D}^1, \\ o_j \in g(i,1)}} \mathcal{C}_{i,o_j,d_j}^1 \cdot x_{ij}^1 + \frac{1}{ \xi^D } \sum_{k \leq \xi^D } \sum_{t=2}^{Q+1} \sum_{i \in \mathcal{L}} \sum_{\substack{j \in \xi_t^{D,k}, \\ o_j \in g(i,t)}} \mathcal{C}_{i,o_j,d_j}^t \cdot x_{ij}^{t,k} \quad (3.35)$	$\text{s.t.} \quad \sum_{\substack{j \in \mathcal{D}^1, \\ o_j \in g(i,1)}} x_{ij}^1 \leq \mathcal{N}_i^1 \quad \forall i \in \mathcal{L} \quad (3.36)$
$\sum_{i \in f(o_j,1)} x_{ij}^1 \leq R_j^1 \quad \forall j \in \mathcal{D}^1 \quad (3.37)$	$\sum_{\substack{j \in \xi_t^{D,k}, \\ o_j \in g(i,t)}} x_{ij}^{t,k} \leq \mathcal{N}_i^1 - \sum_{\substack{j \in \mathcal{D}^1, \\ o_j \in g(i,1)}} x_{ij}^1 - \sum_{t'=2}^{t-1} \sum_{\substack{j \in \xi_{t'}^{D,k}, \\ o_j \in g(i,t')}} x_{ij}^{t',k} + \mathcal{N}_i^t$ $+ \sum_{t'=2}^t \sum_{\substack{j \in \mathcal{D}^1, \\ d_j=i}} \sum_{i' \in f(o_j,1)} \delta_{i'j}^{1,t'} \cdot x_{i'j}^1 + \sum_{t'=2}^{t-1} \sum_{t''=t'+1}^t \sum_{\substack{j \in \xi_{t'}^{D,k}, \\ d_j=i}} \sum_{i' \in f(o_j,t'')} \delta_{i'j}^{t',t'',k} \cdot x_{i'j}^{t',k}$ $\quad \forall i \in \mathcal{L}, k \leq \xi^D , \forall t > 1 \quad (3.38)$
$\sum_{i \in f(o_j,t)} x_{ij}^{t,k} \leq R_j^{t,k} \quad \forall k \leq \xi^D , j \in \xi_t^{D,k}, \forall t > 1 \quad (3.39)$	$x_{ij}^1 \quad \text{non - negative integer} \quad \forall i \in \mathcal{L}, j \in \mathcal{D}^1 \quad (3.40)$
$x_{ij}^{t,k} \quad \text{non - negative integer} \quad \forall i \in \mathcal{L}, \forall k \leq \xi^D , j \in \xi_t^{D,k}, \forall t > 1 \quad (3.41)$	

Table 3.7: MSS

3.3 Relaxing the Assumptions

In our optimization models for the U-OFLYMPIAD and U-OLYMPIAD, we make a few assumptions. In this section, we describe the assumptions and provide the ways of relaxing them:

1. The time is divided into discrete blocks (each with duration Δ) and we consider only the next decision epoch in **TSS**. Such an approach is limiting as different requests have different travel time. We relax this assumption by providing a multi-period two-stage model, which allows for smaller values of Δ

and also considers multiple discrete blocks at once. More details about the multi-period two-stage model is provided in the Section 3.3.1.

2. The server moves from one place to another if and only if it is assigned to a request. It is fairly trivial to relax this assumption by introducing the notion of dummy requests. We provide a detailed description in the Section 3.3.2.
3. Customers are impatient and if they are not assigned a server in one decision epoch, they will not wait for the next decision epoch for a server to be assigned. However, we can easily relax this assumption by representing a customer staying across multiple decision epochs as a customer who leaves and arrives as a new customer at the next decision epoch.
4. The server will start moving as soon as it is assigned to a request. The time taken to compute the assignment is ignored. Therefore, a server assigned to a request at decision epoch t will start moving towards request at the time corresponding to decision epoch t . This is a reasonable assumption, as we are able to obtain solutions in less than a minute (as shown in our experiments).

3.3.1 Multi-Period Two-Stage Stochastic (MSS) Optimization Formulation for U-OLYMPIAD

In **TSS**, we only consider the future requests for the next decision epoch. Given that different requests may have different travel time, we extend the **TSS** model to consider the samples for multiple decision epochs to improve the performance for small Δ values. The extended model is referred to as **MSS**.

The optimization model for **MSS** is shown in the Table 3.7. The number of variables and constraints in the formulation increase with the value of Q . Therefore, the complexity of **MSS** formulation increases as the value of Q increases.

The variable x_{ij}^1 denotes the number of location i servers assigned to the j^{th} element of D^1 at the current decision epoch. The variable $x_{ij}^{t,k}$, $t > 1$ denotes the

number of location i servers assigned to element j of $\xi_t^{D,k}$.

Constraints (3.36) and (3.38) ensure that at any decision epoch, the number of servers assigned from location i is less than the number of available servers. Constraints (3.37) and (3.39) ensure that at any decision epoch, between any location pair, the number of requests assigned to servers is less than the number of available requests between the origin and destination location pair.

Given the similarity in the formulations, we can again employ Benders Decomposition to reduce the complexity with increasing the number of samples. Difficult variables will still be the stage 1 variables, i.e., x_{ij}^1 and all other variables, $x_{ij}^{t,k}$ ($t > 1$) would be the slave variables with a slave for each sample of customer requests.

3.3.2 Dummy Requests

We can remove the assumption that servers will only move when they are assigned to a request by introducing dummy requests. Dummy requests have zero revenue and have a destination in a given location. We introduce $u_{ij'}^1$ as an integer variable denoting the number of location i servers assigned to move to location j' at the current decision epoch, i.e., the number of location i servers assigned to dummy requests with destination in location j' at the current decision epoch. Similarly, $u_{ij'}^{t,k}$ denote the number of location i servers assigned to move to location j' in sample k at decision epoch t , $t > 1$. We modify the **MSS** formulation to include these variables. There will be a cost associated with the movement of server to serve dummy requests which is included in the objective function. The modified **MSS** formulation is shown in Table 3.8 ⁵. $Cost_{i,j'}^t$ denotes the cost of moving a server from location i to location j' at decision epoch t .

⁵We abuse the notation a bit and also use $\delta_{j'i}^{t,t'}$ as a binary constant which is 1 if server starting at decision epoch t from location j' reaches location i exactly at decision epoch t'

Section	Description	Key Content
3.4.1	Datasets	Details on the datasets and different data fields used from the datasets.
3.4.2	Experimental Settings	Details on the different inputs, parameters and evaluation settings used.
3.4.1.1	Zone Creation	Describes the process of creating multiple zones for any dataset.
3.4.3	Main Results	Describes our key results and shows the comparison of seven algorithms at different times on the three metrics of revenue, number of requests and run-time.
3.4.4	Justification for values of parameter settings	Justifies the parameters used in Section 3.4.3 by comparing the performance of our algorithms for different parameter values.
3.4.5	Synthetic Scenarios where MSS and BD do not Improve Performance	Describes the specially created scenarios where our algorithms do not provide good results.

Table 3.9: Experiment section outline

3.4 Experiments

In this section, we will compare the performance of seven approaches Multi-Period Two-Stage Stochastic optimization (**MSS**)⁶ and Benders Decomposition (BD), Greedy (GD) algorithm, Randomized Greedy Algorithm (RGD), One-Stage optimization (OS), Approximate Dynamic Programming formulation (ADP) and Hybrid Multi-Period Two-Stage Stochastic optimization approach (HSS) used in (Powell, 1996). We employ $\mathbf{MSS}(\Delta = x, Q = y)$ and $\mathbf{BD}(\Delta = x, Q = y)$ to refer to our approaches when the time interval Δ is set to x and the look ahead decision epochs Q is set to y . We compare different approaches with respect to (i) Revenue earned by servers; (ii) Number of requests satisfied; and (iii) Run-time to compute the assignment.

Table 3.9 provides the outline for this section. We will show two main results that demonstrate the significant utility of our approaches:

- **MSS** and **BD** consistently outperform myopic (GD, RGD and OS) and multi-step approaches (ADP and HSS). While the improvement varies, on an average (across datasets and other settings) there was a 20% improvement over GD, RGD and OS and 9% improvement over HSS and ADP.
- **BD** provides the best trade off between run-time and solution quality. It can solve the problems in quickly while achieving roughly the same solution quality as **MSS**.

⁶**MSS** is a generalization of **TSS**, so we only focus on **MSS**. **TSS** is equivalent to **MSS** for $Q=1$

3.4.1 Datasets

We conducted our experiments by taking the demand distribution from three real world datasets of major taxi companies. The first dataset is the publicly available New York Yellow Taxi Dataset (NYYellowTaxi), henceforth referred to as NY-Dataset. The names of the other two real world datasets can not be revealed due to confidentiality agreements. They are referred to as Dataset1 and Dataset2. These datasets contain the data of past customer requests for servers at different times of the day and for different days of the week. From these datasets, we take the following fields:

- Pick-up and Drop-off Location (Latitude and Longitude coordinates): These locations are mapped to the zones as mentioned in the Section 3.4.1.1.
- Pick-up Time: This time is converted to the appropriate timestep based on the value of Δ . The pick-up time $hh : mm$ is mapped to the timestep/decision epoch $\lceil \frac{hh \cdot 60 + mm}{\Delta} \rceil$. For example, for $\Delta = 5$, 08:04 AM is mapped to the timestep $\lceil \frac{8 \cdot 60 + 4}{5} \rceil = 97$, i.e., all the requests between time 08:00 AM and 08:05 AM are mapped to the the time 08:05 AM.

The distance of a trip and the revenue earned from a trip are computed using the underlying model typically employed by the taxi companies (an initial base fare + a quantum that accrues with the distance travelled/time taken). We describe it in the Section 3.4.2.

Since our approaches perform well on these three real world datasets, we also provide the specially created scenarios where our approaches may not work as well. The details of these specially created synthetic scenarios are described in the Section 3.4.5.

For the experimental evaluation, we divide the city into multiple zones. Therefore, set of locations \mathcal{L} corresponds to the zones for the experimental evaluation.

In the next subsection, we describe the zone creation process using the publicly available NYDataset. The zone creation process is important because we perform matching at the level of zones. The zone level matching ensures scalability while

providing a good approximation.

3.4.1.1 Zone Creation

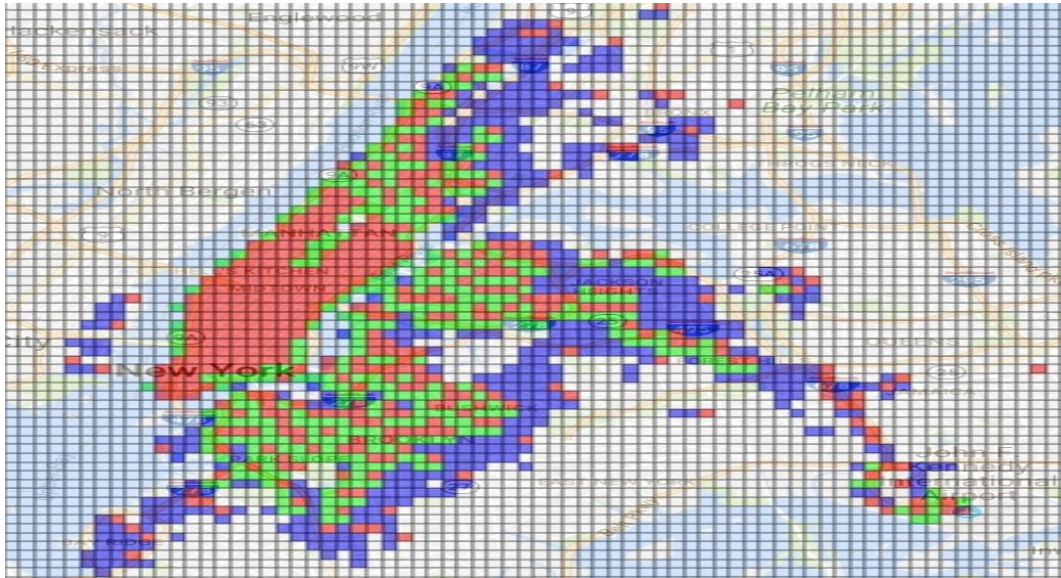


Figure 3.4: Zone Creation

We employ past data of pick-ups and drop-offs to divide a city into zones. Using past data helps in providing the right tradeoff between having few zones and the error introduced due to consideration of zones. There are four key steps to the zone creation process for all three datasets:

- We divide the entire city area into small grids (0.5 KM x 0.5 KM).
- We consider the minimum number of grids that make up most of the pick-ups and drop-offs (> 90%). These are the first set of zones. This will ensure that the error is negligible for most of the requests.
- We then add a minimum number of zones, so that 9% of the remaining pick-ups and drop-offs are within 1 KM x 1 KM of a zone.
- We add a minimum number of zones, so that the remaining 1 % of the pick-ups and drop-offs are within 2 KM x 2 KM of zone.

Figure 3.4 provides the spatial configuration of zones in case of New York city. All the red grids represent zones. Green grids represent the areas that are within 1 KM x 1 KM of a zone and account for 9% of the pick-ups/drop-offs. Blue grids

represent the areas that are within 2 KM x 2 KM of a zone and account for 1% of past pick-ups/drop-offs. We used the trip data for 6 months from Jan 2016-June 2016 (total 136830072 pick-ups/drop-offs) to create these zones. To assign zone to any new location, we check the distance of the location from the centers of these created zones and assign the zone with minimum distance. The percentage values can be changed depending on the need and dataset, while ensuring that for majority of locations error remains negligible.

3.4.2 Experimental Settings

There are three different categories of settings that have an impact on the performance of algorithms:

1. Inputs provided to all algorithms: These include:

- τ : It represents the maximum time within which the server should reach the pick-up point. We take the value of τ as 5 minutes, i.e., at any decision epoch a server can be assigned to a request if and only if the time taken by the server to reach the request origin zone is less than or equal to 5 minutes.
- Revenue model: We employed the following model for experimental evaluation that is a well accepted standard (*Singapore Taxi Fare*). The model calculates revenue as follows:

$$C_{i,o_j,d_j}^t = \mathbf{B} + \mathbf{R} \cdot \text{dist}(o_j, d_j) - \mathbf{P} \cdot (\text{dist}(i, o_j) + \text{dist}(o_j, d_j)) \quad \forall t$$

where $\text{dist}(o_j, d_j)$ is the distance between the centers of zones o_j and d_j . For different cities, the values of \mathbf{B} , \mathbf{R} and \mathbf{P} are different⁷

- Travel time between zones: For the experimental results, the time taken to travel

⁷For dataset1, dataset2, we use 3.8, 0.5 and 0.1 for \mathbf{B} , \mathbf{R} and \mathbf{P} respectively. For NYDataset, we use 2.5, 2.5, 0.1 for \mathbf{B} , \mathbf{R} and \mathbf{P} respectively.

between zones⁸ is also same irrespective of the time of day and is calculated as

$$\mathcal{T}(z, z', t) = \frac{dist(z, z')}{v_{server}} \quad \dots \forall t$$

where v_{server} is the speed of server which is taken as 40km per hour.

- Number of servers (\mathcal{N}_i^1): The number of servers used is dependent on the fleet size of the taxi company. At the start of the experiment servers in different zones are distributed either uniformly if server locations are not observed (NYDataset and Dataset 1) or as observed in the data (Dataset2). Based on the assignment obtained by algorithms at any decision epoch availability of servers at the next decision epoch is updated. In the results section, we vary the number of servers to show the performance of all algorithms for different number of servers.
- Number of locations/zones ($|\mathcal{L}|$): We described the zone creation process in the Section 3.4.1.1. Using our zone creation process we obtained 483 zones for the 2 confidential datasets and 436 zones for the NYDataset. For a different city and dataset, the number of zones will be different.

2. Parameters of algorithms: The parameters required by our algorithms (MSS and BD) are:

- Decision epoch length (Δ): This parameter determines how often, the algorithm should be executed and assignment decisions are made. We identify the right value of Δ through experiments as described in results section.
- Look ahead timesteps (Q): The value of Q is taken such that $\Delta \cdot (Q + 1) = 30$ minutes, i.e., if Δ is 1 minute, Q is taken as 29 and if Δ is 15, Q is taken as 1. We choose a fix value of 30 minutes because more than 90% of the requests complete within 30 minutes with average travel time between 5-10 minutes. So 30 minutes provides a good look ahead. But this value can be changed depending on the dataset.

⁸We can update it to more realistic values from data without impacting the overall results.

- Number of samples (ξ^D): While computing an assignment at decision epoch t , our approaches (MSS and BD) and existing multi-step approaches (HSS and ADP) require samples of customer requests at decision epochs $t+1, t+2, \dots, t+Q$ from past data (at the same decision epoch on a weekday/weekend depending on whether the decision epochs are on a weekday/weekend). We identify the right value for the number of samples through experiments as described in results section.

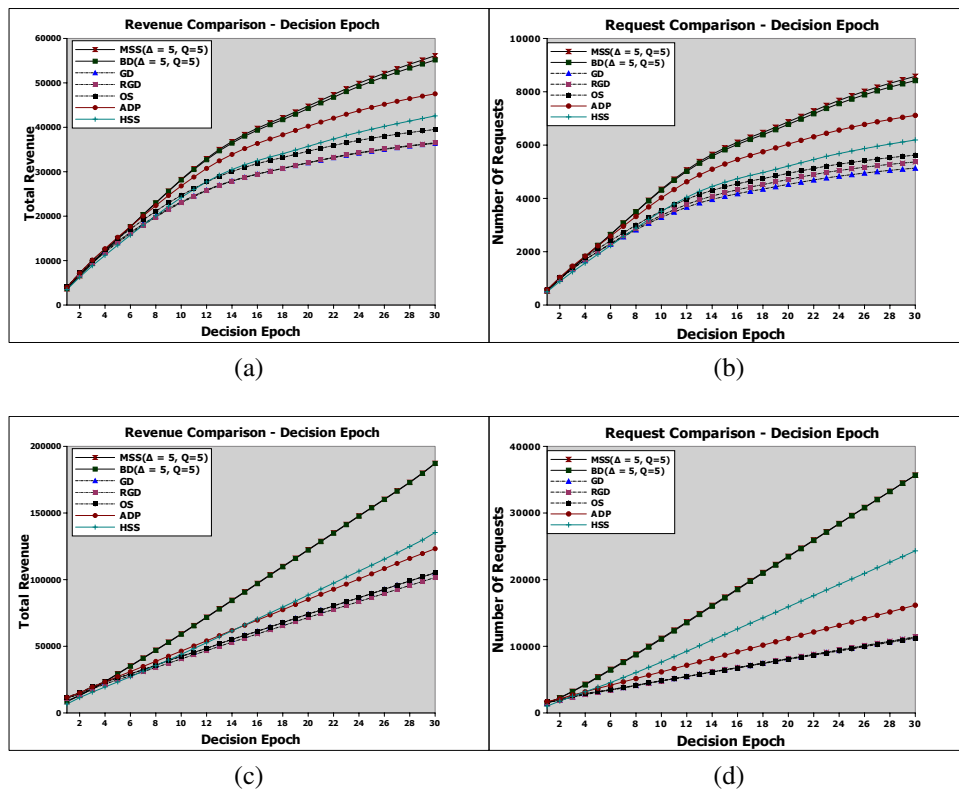


Figure 3.5: Comparison of Revenue earned and Number Of Requests served using different algorithms at each decision epoch. $|\mathcal{L}| = 483$, $|\xi^D| = 10$, $\mathcal{N}_i^1 = 2000$ (a) (b) Dataset1 (c) (d) Dataset2

3. Evaluation settings: For each algorithm, once the assignment is computed at each decision epoch, we evaluate the assignment on the realized requests (which are samples from the past data that are not considered while computing the assignment). The objective for all the algorithms is to maximize revenue. To provide the right trade-off between run-time and solution quality, we considered 3 iterations in BD.

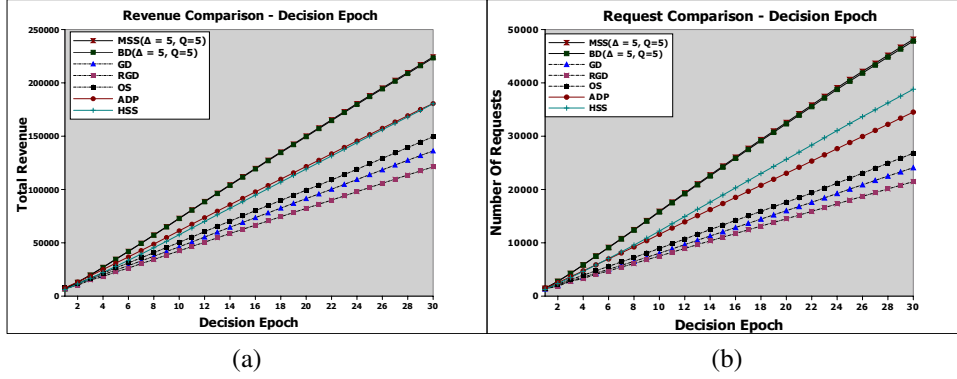


Figure 3.6: Comparison of Revenue earned and Number Of Requests served using different algorithms at each decision epoch. $|\mathcal{L}| = 436$, $|\xi^D| = 10$, $\mathcal{N}_i^1 = 2000$ (a), (b) NYDataset

The evaluation settings include the following :

- Number of decision epochs (M): Typically transitions in the server demand happen approximately every 3 hours (morning and evening peak hours, lunch), so we choose the value of 2.5 hours, i.e., $M = \frac{2.5 \cdot 60}{\Delta}$. However, we did try other values for M (lower and higher than 2.5 hrs) and the results were similar.
- Number of days and the time of evaluation: We performed experiments with requests at various times of the day, 8:00 AM, 3:00 PM, 6:00 PM and on different days. We evaluated the approaches by running them on 45 different days and taking the average values over 45 days for Dataset1 and Dataset2. For NYDataset, we evaluated the approaches on 15 weekdays between 4th April 2016-22nd April 2016.

We conducted experiments with all the combinations of settings and inputs mentioned in this section. To avoid repeating similar results over and over again, we provide the representative results.

3.4.3 Main Results

In this section, we compare the revenue and the number of requests served by all seven algorithms. We also compare the run-time of all the approaches by using different number of servers.

We conducted experiments by considering the demand distributions from all the datasets. We choose the best configuration for parameters of all algorithms (justification provided in the next section). All approaches are executed for $\Delta = 5$. For our approaches, we use 10 samples and the look ahead timesteps (Q) as 5. For HSS and ADP as well, we use the number of samples as 10. For HSS similar to MSS, we use the look ahead timesteps as 5. For ADP, we use the look ahead timesteps as 10. ADP algorithm as described in the 2.4.5 involves solving an optimization problem for each sample and for each time step in the planning horizon. As opposed to other multi-step approaches, ADP algorithm considers one sample at a time, therefore, we use a higher value of Q for ADP. But it is not possible to run ADP algorithm for more than 10 time steps for sufficient number of samples in real-time. Moreover, we checked on few instances by using all the $M(30)$ time steps as planning horizon and did not see any significant improvement in the performance of ADP algorithm. Therefore, we report the results for ADP algorithm with a fixed value of Q as 10. We first compare the revenue obtained by all the algorithms at each time step for 2000 servers starting from 8AM. We have similar results while starting from 3:00 PM and 6:00 PM. Figure 3.5 and 3.6 shows the comparison of the total revenue obtained and the total number of requests served by different algorithms at each decision epoch for all the datasets. Here are the key observations:

- For the first decision epoch, the value obtained by the myopic approaches is higher than **MSS** and **BD**. This is because **MSS** and **BD** take sub-optimal decisions at the current decision epoch to obtain high revenue over future decision epochs. After initial decision epochs, **MSS** and **BD** starts outperforming other approaches.
- With respect to the revenue, on NYDataset, **MSS** and **BD** provided close to \$37 improvement per server as compared to the myopic approaches and close to \$22 improvement per server as compared to ADP and HSS. For Dataset2, **MSS** and **BD** provided nearly \$41 improvement per server as compared to the myopic approaches and \$26 improvement as compared to ADP and HSS. Similarly, on

Dataset1, we get \$8 improvement as compared to the myopic approaches and \$4 improvement as compared to ADP and HSS.

- With respect to the number of requests served, on NYDataset, **MSS** and **BD** serve 21381 additional requests as compared to the myopic approaches and they serve 9378 additional requests as compared to ADP and HSS. For Dataset2, **MSS** and **BD** serve 24000 additional requests as compared to myopic approaches and it serves additional 11000 requests as compared to ADP and HSS. Similarly, on Dataset1, we serve 3000 additional requests as compared to the myopic approaches and 1400 additional requests as compared to ADP and HSS. This is a significant result, because in most cities at rush hours, the number of servers is almost always lower than the actual demand available.
- On NYDataset, ADP and HSS have identical results (requests served by HSS is more but revenue is almost the same). On Dataset2, HSS slightly outperforms ADP and on Dataset1 ADP outperforms HSS. Dataset1 has higher variance in requests as compared to the other 2 datasets at 8AM. Therefore, for NYDataset and Dataset2, ADP gets almost similar samples in each iteration but with Dataset1 it gets different samples which could be the reason for better performance of ADP on Dataset1.

The reason for **MSS** and **BD** providing better results as compared to **HSS** is that **HSS** works with larger zones at future stages, so **MSS** and **BD** have better approximation of future as compared to **HSS**. One of the reasons for **ADP** not performing as well as **MSS** is that the decision epoch at which the request completes depends on the assigned server (as described in the Section 3.1) which breaks the Markov property. In addition, as described in 2.4.5, possible assignment of the servers from nearby zones makes the value function non separable across zones, but **ADP** approximation assumes this separation. Due to these reasons, the value function approximation updated using the dual variables of a relaxed linear program is not accurate.

The value function approximation used for **ADP** is linear. With piece-wise lin-

ear value function approximation, the time taken to run each optimization increases, so it is not possible to perform significant number of iterations using piece-wise linear value function approximation. The results obtained using the piece-wise linear value function approximation with smaller number of servers are similar to the linear value function approximation. Therefore, we use linear value function approximation for reporting the results.

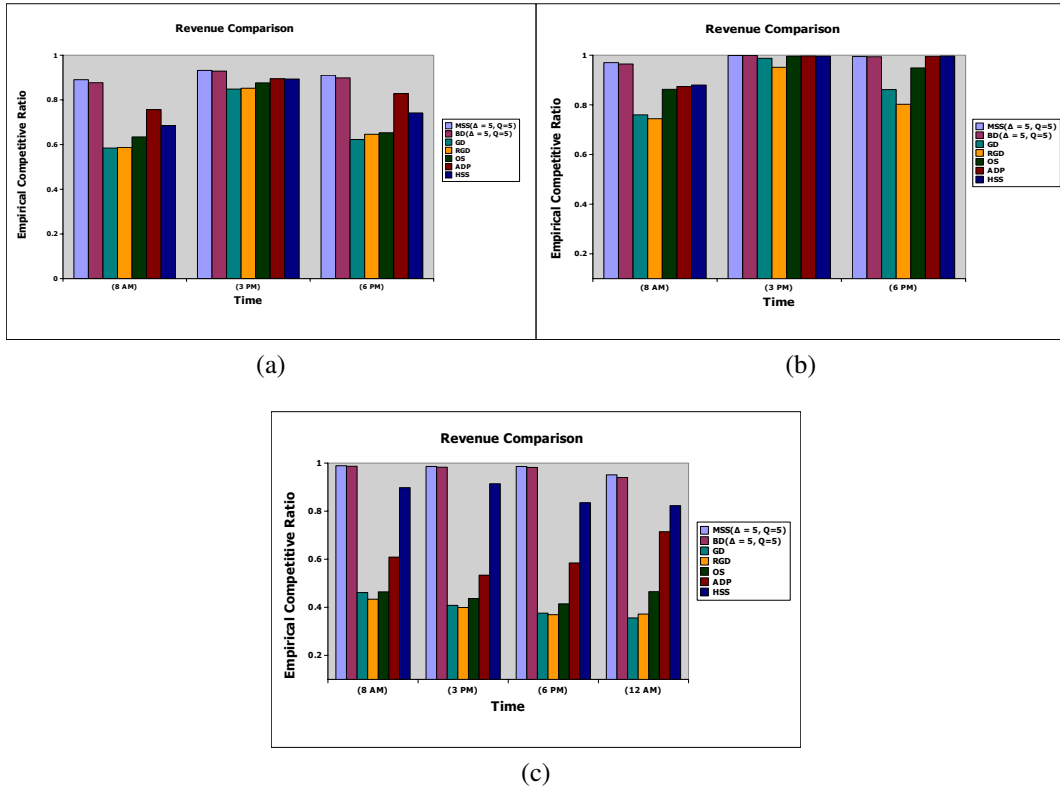


Figure 3.7: Comparison of Empirical Competitive Ratio of algorithms during Peak and Non Peak Time, $|\xi^D| = 10$, (a) Dataset1 $\mathcal{N}_i^1 = 2000$ (b) Dataset2 $\mathcal{N}_i^1 = \text{As observed in data}$ (c) NYDataset $\mathcal{N}_i^1 = 1000$

Peak and Non-Peak Time: To evaluate the sensitivity of performance of algorithms with respect to the time of the day, we now compare the competitive ratio of algorithms in the morning (8:00AM), evening (06:00PM) and afternoon (03:00PM) for all datasets. For NYDataset, as shown in the next section, the distribution of requests remains almost same throughout the day and the number of requests were low only at the night time. Therefore, for NYDataset, we also compare the competitive ratio at 12AM.

For Dataset1, the average number of requests at each decision epoch is 700 at 08:00AM, 200 at 03:00PM and 350 at 06:00PM. We can observe in the Figure 3.7 that at 03:00PM, when there are more servers as compared to the number of requests, the gap between the competitive ratio of **MSS**, **BD** and myopic approaches is less than 10% but at 08:00AM and 06:00PM, on an average the gap between the competitive ratio of **MSS**,**BD** and myopic approaches is more than 25%, the gap between **MSS**,**BD** and **ADP** is nearly 12% and the gap with **HSS** is nearly 19%.

Similar results are observed on Dataset2. For Dataset2, the number of available servers is taken as observed in the data. The average number of requests at each decision epoch is 2000 at 08:00AM, 1700 at 03:00PM and 1800 at 06:00PM. We can observe that at 3PM the gap between the competitive ratio of **MSS**, **BD** and other approaches is less than 1% due to fewer requests and more available servers (nearly 8000 servers at 3PM as compared to 5500 servers at 08:00AM). At 08:00AM the gap is nearly 10% from all the approaches.

For NYDataset, the average number of requests at each decision epoch is 1941.8 at 08:00AM, 1726.1 at 03:00PM, 2407.617 at 06:00PM and 712.88 at 12:00AM. Figure 3.7 shows the comparison of the competitive ratio of algorithms at different times. In NYdataset, we did not observe the competitive ratio of myopic algorithm improve with fixed number of servers at different time but the competitive ratio of **ADP** improves at 12:00AM. This improvement is more likely due to variance in the samples provided. As shown in the Section 3.4.4, NYDataset has low mean and high variance in the number of requests at night time. The gap between the competitive ratio of **MSS**, **BD** and myopic approaches is more than 40% at all times. The maximum gap between **MSS**, **BD** and **ADP** is 40% and minimum gap is 20%. The maximum gap between **MSS**, **BD** and **HSS** is 12% and minimum gap is 7%.

Number of Servers: Finally, we also compare all the algorithms by experimenting with different number of servers (Figures 3.8 and 3.9). We observe that as the number of servers increase, the gap between **MSS**, **BD** and other approaches decreases.

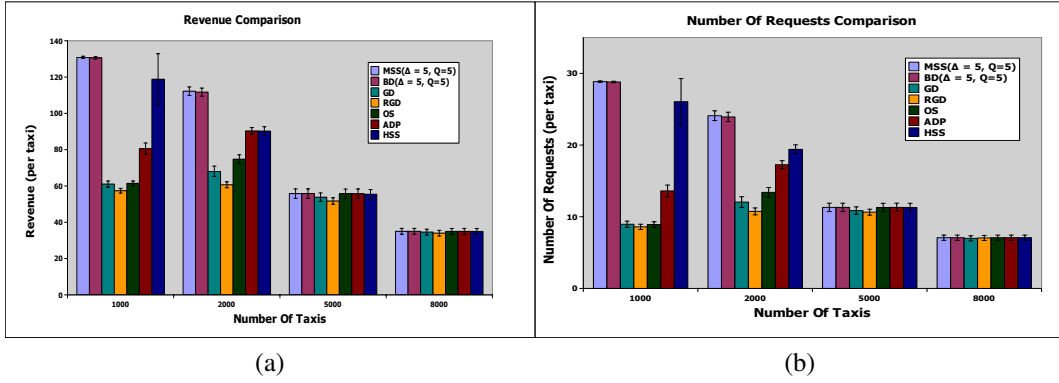


Figure 3.8: Comparison of the revenue earned and the number of requests served using different algorithms. NYDataset: In (a) and (b) $|\mathcal{L}| = 436, |\xi^D| = 10$

This is because when more servers are available, servers will be free even after executing assignments at the current decision epoch, so future demands can be met irrespective of the current assignment. Furthermore, when there are significantly more servers than the demand, sophisticated matching approaches are not required.

We also compare the run-time of algorithms with different number of servers. Figure 3.10 shows the run-time of **MSS** and **BD** with different number of servers. We observed that when the number of servers are much less or servers are available in excess, run-time of **MSS** is lower as compared to the case when the number of servers is comparable to the available requests. For Dataset1, **MSS** has maximum run-time at 2000 servers and for the other 2 datasets it has maximum run-time for 5000 servers. Though run-time of **MSS** changes with the number of servers, run-time of **BD(P)** remains almost same irrespective of the number of servers⁹. Moreover we conducted our experiments on academic systems, on commercial servers this will take much less time.

⁹Since, the reported run-time for **BD** is based on sequential solving of slaves, the actual run-time when slaves are run in parallel on multiple cores is lower. We show the expected time which will be taken by **BD**, if slaves are solved in parallel (denoted by **BD(P)** in the figure). We calculated this time by considering the time taken to solve slaves in parallel as the maximum time taken by any slave in each iteration.

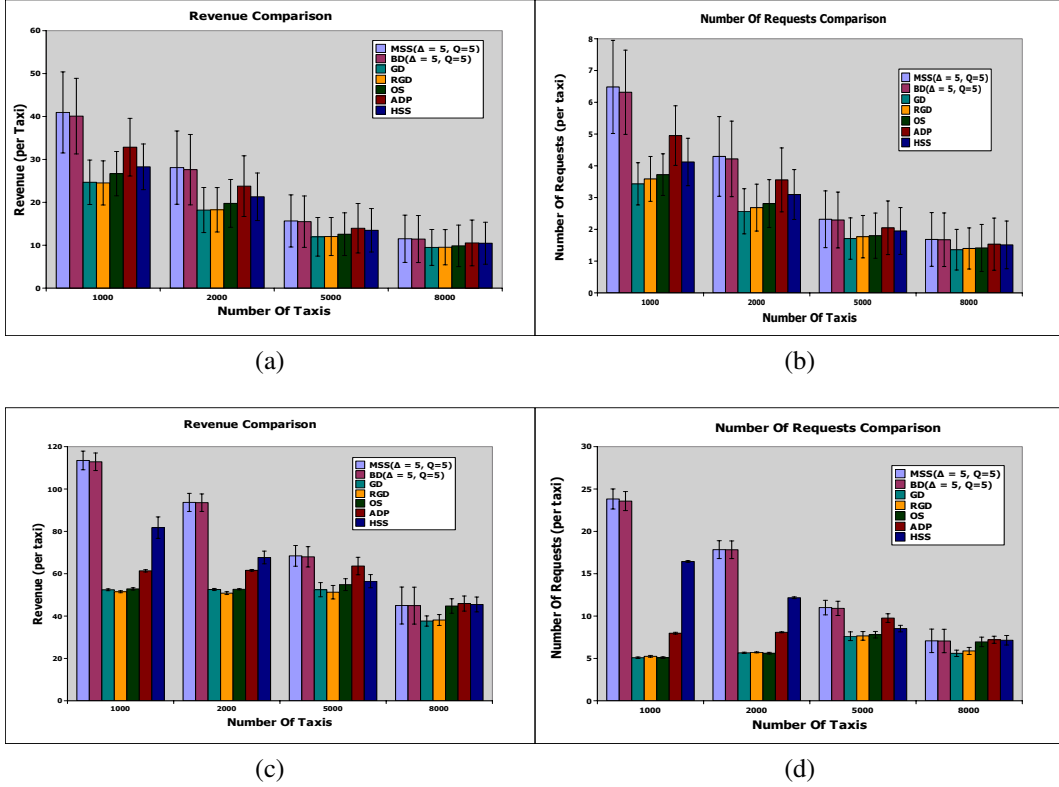


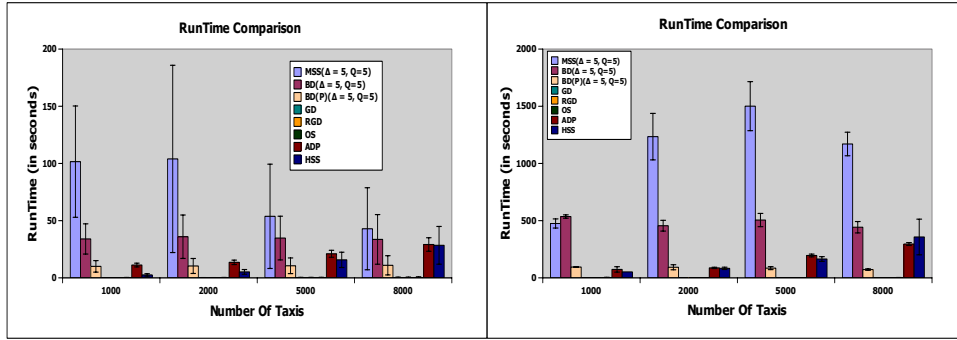
Figure 3.9: Comparison of the revenue earned and the number of requests served using different algorithms. (a), (b) Dataset1 and (c), (d) Dataset2. In (a), (b), (c) and (d) $|\mathcal{L}| = 483, |\xi^D| = 10$

3.4.4 Justification for Values of Parameter Settings

In this section, we show the reason for using the Δ value as 5 and the number of samples as 10 in the previous section.

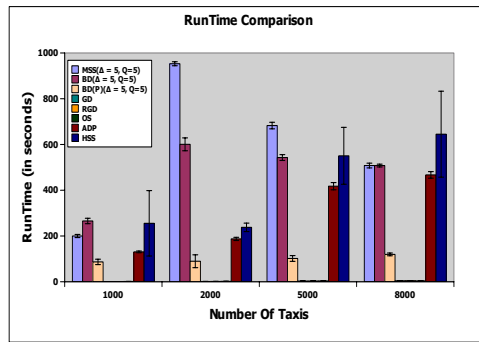
Decision epoch length (Δ): We identify the appropriate value of Δ which provides the right trade-off between solution quality and run-time. The solution quality increases with decreasing the value of Δ . This is because a server is assigned to only one request in each decision epoch (Section 3.3). Therefore, for large Δ values, even if a server can serve more than one request in Δ duration, it will be serving only one request. To observe the change in solution quality on decreasing the value of Δ , we compare the objective value of U-OLYMPIAD over 2.5 hours for different value of Δ on both datasets.

Figure 3.11 shows the comparison of total revenue earned and run-time of U-



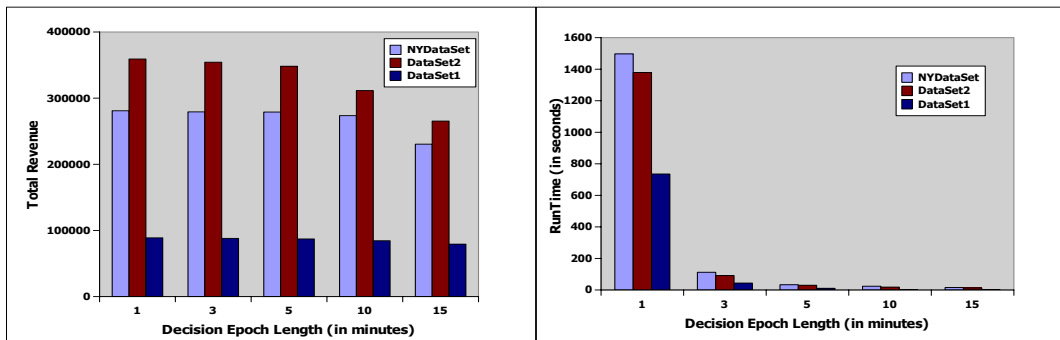
(a)

(b)



(c)

Figure 3.10: Run time comparison of our algorithms for different number of servers (a) $|\mathcal{L}| = 483, |\xi^D| = 10$ Dataset1 (b) $|\mathcal{L}| = 483, |\xi^D| = 10$ Dataset2 (c) $|\mathcal{L}| = 436, |\xi^D| = 10$ NYDataset



(a)

(b)

Figure 3.11: Revenue and the run-time comparison of the Offline algorithm on different datasets for different decision epoch length.

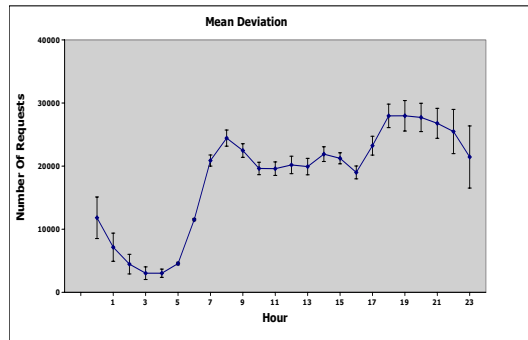
OLYMPIAD for different Δ values (in minutes) on all the datasets ¹⁰.

As mentioned in the Section 3.4.2, the value of Q is taken such that $\Delta \cdot (Q + 1) = 30$ minutes, i.e., if Δ is 1, Q is taken as 29 and if Δ is 15, Q is taken as 1.

Figure 3.11a shows the total revenue obtained on executing U-OLYMPIAD. We observe that on all the datasets, there is a major increase in revenue on decreasing the value of Δ from 15 to 10 and 10 to 5. But the increase in revenue on decreasing the value of Δ from 5 to 3 and 3 to 1 is comparatively less. Figure 3.11b shows the maximum time taken to compute a single assignment with 1 sample of future demand. We observe that the time taken to compute an assignment, drastically increases on decreasing the value of Δ from 3 to 1. This is due to the large increase in the number of constraints in the optimization formulation. Even for $\Delta = 3$, the time taken to compute a single assignment with one sample is 100 seconds on Dataset2.

As the time taken is high for $\Delta < 5$ and quality of solution at $\Delta = 5$ is comparable to the solution quality at $\Delta = 1$, we take the value of Δ as 5 for our experiments.

Number of samples ($|\xi^D|$): Our next set of experiments measure the performance



(a)

Figure 3.12: Mean and Deviation in the number of requests available at each hour for NYDataset

of our approaches with respect to change in the number of samples. Figure 3.12a

¹⁰The average number of requests in Dataset 2 and NYDataset is more than twice the average number of requests in Dataset1 (nearly 60000 requests in Dataset2 and NYDataset as compared to 30000 requests in Dataset1). Therefore, there is a significant difference between total revenue earned and time taken to compute an assignment.

shows the mean and deviation in the number of requests available at each hour over 15 weekdays in NYDataset. From the figure, we can observe that there is a high variance in the number of requests at night between 10:00PM to 02:00AM, but during the day, the variance is low. Therefore, during the day, even with a single sample, we can obtain good quality results. Due to this reason, to observe the effect of using different number of samples for NYDataset, we provide the results at 12:00AM (as there is low mean and high variance at 12:00AM). On the other hand, Dataset1 has high variance in the number of requests in the morning at 8:00AM on weekday (minimum: 6000, maximum: 60000 over 2.5 hours). Therefore, for Dataset1, we provide the results at 08:00AM. The average number of requests at each decision epoch is 700 for Dataset1 and 712.88 for NYDataset. The total number of servers is set to 1000 for NYDataset and 2000 for Dataset1.

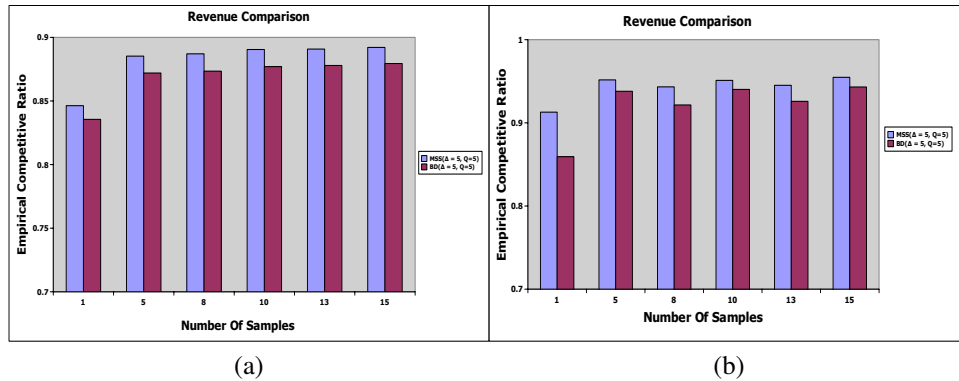


Figure 3.13: Comparison of empirical competitive ratio of algorithms for different number of samples. (a)Dataset 1: $|\mathcal{L}| = 483$, $\sum_{i \in \mathcal{L}} \mathcal{N}_i^1 = 2000$, (b) NYDataset : $|\mathcal{L}| = 436$, $\sum_{i \in \mathcal{L}} \mathcal{N}_i^1 = 1000$

Figure 3.13 provides the results for the average value of empirical competitive ratio for Dataset1 and NYDataset. In Figure 3.13a and 3.13b, X-axis denotes the number of samples considered while computing the assignment and Y-axis denotes the empirical competitive ratio (from Section 3.2.4). The key observations are as follows:

- (1) The most significant result is that **MSS** and **BD** are able to achieve more than 88% of the optimal revenue with 10 samples on Dataset1 and more than 95% on NYDataset. Even though both datasets have high variance in the number of

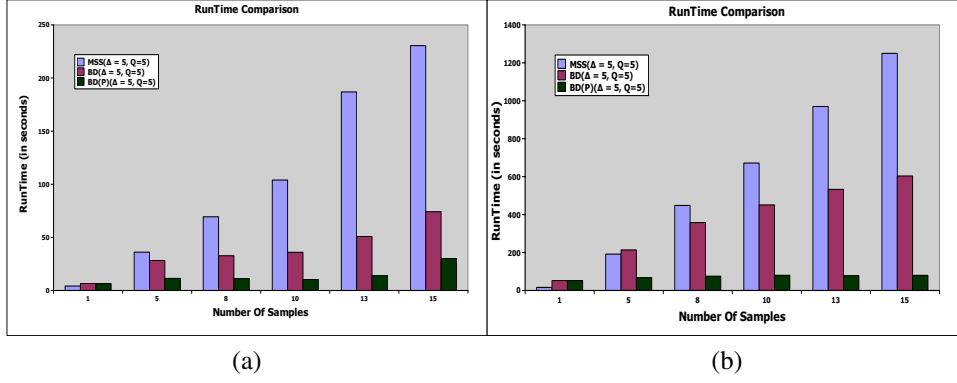


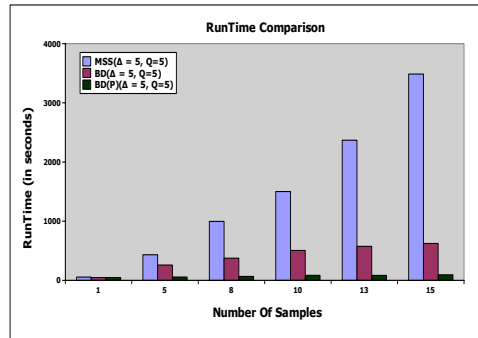
Figure 3.14: Comparison of run time of algorithms for different number of samples.(a)Dataset 1: $|\mathcal{L}| = 483$, $\sum_{i \in \mathcal{L}} \mathcal{N}_i^1 = 2000$, (b) NYDataset : $|\mathcal{L}| = 436$, $\sum_{i \in \mathcal{L}} \mathcal{N}_i^1 = 1000$

requests, Dataset1 has higher variance. Moreover, Dataset1 has more requests within the same pair of zones which makes a single zone more prominent than the others. So in case of Dataset1, the important zones for different samples are different and can be completely different from the test day. On the other hand, requests are more distributed in NYDataset, therefore, NYDataset has higher empirical competitive ratio as compared to the Dataset1.

- (2) Even with 1 sample, on an average **MSS** and **BD** obtain $\tilde{84}\%$ of the optimal revenue on the Dataset1 and more than 85% of the optimal revenue on NYDataset.
- (3) With respect to the revenue, on Dataset1, on an average, **MSS** provides 1.5% additional improvement over **BD**, while on NYDataset, with higher number of samples, **MSS** provides 1% improvement over **BD**.
- (4) **BD** provides the right trade-off between run-time and solution quality. While **MSS** provides high quality solutions (especially for Dataset1), it can take significantly more time (up to 1500 seconds when making online decisions).
- (5) **BD(P)** on an average takes 24 seconds with 10 samples to compute an assignment for Dataset1 and $\tilde{83}$ seconds with 10 Samples for NYDataset.

On the other hand, there is very low variance over the requests on different days in Dataset2 (minimum:56000 ,maximum:60000 over 2.5 hours). Therefore, the empirical competitive ratio is more than 95% even on taking only one sample and does

not improve much on adding more samples. The run-time results on Dataset2 are similar to NYDataset. NYDataset and Dataset2 take more time to compute an assignment as compared to Dataset1 because as discussed before Dataset1 has more requests between same zone pairs while requests are more distributed in other 2 datasets. Therefore, the number of zone pairs which have requests between them is much lower for Dataset1. Due to this, the overall size of linear program is larger for other two datasets resulting in higher run-time.



(a)

Figure 3.15: (a)Dataset 2: $|\mathcal{L}| = 483$, $\sum_{i \in \mathcal{L}} \mathcal{N}_i^1 = 5000$

On all the datasets, MSS and BD provide high value of competitive ratio with very few samples. One of the major reasons for this is that the samples need not be exactly similar to the test data to provide a gain in the competitive ratio. As mentioned before, a server can be assigned to a request if it can reach the pick-up location of request within 5 (τ) minutes and the cost of traveling to reach the request pick-up location is negligible as compared to the revenue obtained. Therefore, unless samples are drastically different, they will contribute a lot in increasing the competitive ratio.

3.4.5 Synthetic Scenarios Where MSS and BD Do Not Improve Performance

To better understand the settings where our approaches work well and where it does not, we also performed experiments on the synthetic datasets. We investigate the

following cases, where our approaches can potentially yield bad results as compared to the myopic approaches:

1. Requests between zones generated using uniform, binomial and Poisson distributions: For this setting, we obtained similar results (as on the real world datasets) with **MSS** and **BD** outperforming myopic approaches. On synthetic datasets with uniform distribution, **MSS** and **BD** do not get the advantage of making servers available in the locations with more demand (based on samples) as requests are available in all the zones. The reason behind **MSS** and **BD** still outperforming myopic approaches in this scenario is the revenue model which rewards shorter trips over longer trips.

2. Revenue model with no bias for short trips (over long trips): As myopic approaches consider the revenue for only the current decision epoch, they will assign servers to a trip having long distance. On the other hand, **MSS** and **BD** will prefer serving multiple short distance trips. When we change the revenue model in equation (3.4.2) such that the revenue is directly proportional to the distance, **MSS** and **BD** do not get the advantage by serving multiple short distance trips. On synthetic dataset with uniform distribution of requests and the modified revenue model, all algorithms have comparable value of the competitive ratio, as shown in Figure 3.16.

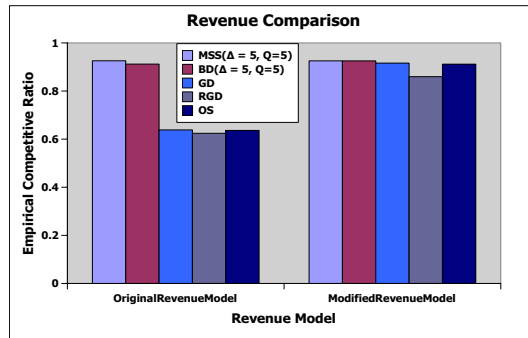


Figure 3.16: Revenue model with no bias for short trips: $|\mathcal{L}| = 50$, $\sum_{i \in \mathcal{L}} \mathcal{N}_i^1 = 2000$, $\frac{1}{30} \sum_{t=1}^{30} \mathcal{D}^t = 2507$

3. High proportion of long distance trips with far apart zones: To simulate this scenario, we had to create a specific setting that is hard to replicate in real world datasets. We took 5 zones and no request has an origin and destination in the same

zone. The time taken to travel between zones is taken such that the servers need to be in exactly the same zone to serve the requests. In the 5 zones considered, we take the minimum time taken to travel between any pair of zones as 15 minutes and maximum time as 40 minutes. Out of 20 possible ($5 \cdot 4$) zone pair combinations, time taken to travel between 6 zone pairs is taken as more than 30 minutes (revenue 12.5), between 6 other zone pairs as 15 minutes (revenue 7.2), 4 other zone pair as 20 minutes (revenue 8.2) and between remaining 4 zone pairs as 25 minutes (revenue 10.5). Requests are generated such that the requests with travel time 15, 20 and 25 minutes are not available at all decision epochs but the requests having travel time 30 minutes or more are available at all decision epochs. The value of τ and Δ is 5 minutes and the value of Q is 5, i.e., $\rho = \Delta \cdot (Q + 1) = 30$ minutes.

In these settings, **MSS** and **BD** can perform arbitrarily bad as compared to the myopic approaches. The intuition is that **MSS** and **BD** can keep on taking sub-optimal decision at the current decision epoch with an expectation of getting higher revenue at the next decision epochs but on reaching the next decision epoch it again takes a sub-optimal decision.

The competitive ratio of **MSS** and **BD** in this setting is constantly less than the myopic approaches even after providing multiple samples as shown in Figure 3.17. The reason is as **MSS** and **BD** are only looking at the next 30 minutes, they decide to first serve a request with 15 minute travel time followed by a request having travel time 30 minutes or more. But on reaching the next decision epoch it ignores the request of higher travel time even if there are no requests with 15-minute travel time available. **MSS** and **BD** find it a better option to be idle for a couple of decision epochs in order to wait for a 15 minute request so that it can serve that and follow it with a request having travel time more than 30 minutes. This keeps on happening till the last decision epoch. As a result, **MSS** and **BD** remain idle, waiting for a short trip and the myopic approaches keep on serving the requests, resulting in higher revenue for them.

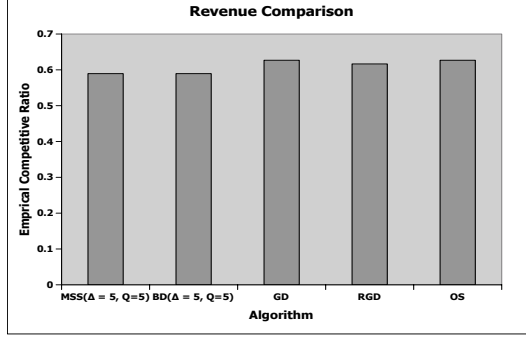


Figure 3.17: High proportion of long distance trips with far apart zones: $|\mathcal{L}| = 5$, $\sum_{i \in \mathcal{L}} \mathcal{N}_i^1 = 10$, $\sum_{t=1}^{30} \mathcal{D}^t = 21$

3.5 Real World Deployment

A recommendation engine built on the optimization formulation proposed in this chapter is used to provide personalized guidance to taxi drivers in Singapore. The field trial of the driver guidance system (DGS) was conducted with 500 recruited taxi drivers from September 2017-end of 2019. Comparing the vacant roaming times before finding passengers, we discover that by following the DGS recommendations, drivers manage to reduce their vacant roaming times by 27% across all time periods. Table 3.10 shows the breakdown of the DGS vs. non-DGS performances in 4 periods: 6-10am, 10am-5pm, 5pm-12am, and 12-6am. Although DGS outperforms non-DGS trips in all time periods, we observe that having guidances brings most advantages for drivers after 5pm. Analyzing further, we discovered that this might be due to the fact that supply-demand imbalances are much more variable after 5pm, especially around the midnight hours. Although drivers in general know where demands are during these hours, they do not know the supply level; this is where guidances can help them.

Time Period	DGS	Non-DGS
6am-10am	8.06 minutes	7.32 minutes
10am-5pm	7.10 minutes	5.72 minutes
5pm-12am	8.80 minutes	5.45 minutes
12am - 6am	15.52 minutes	8.50 minutes

Table 3.10: Real World Deployment - DGS results

The optimization formulation used is based on the formulation in table 3.8. In this work, as only 500 drivers are using the DGS, not all taxis are guided and the

recommendation engine needs to account for the unguided taxis.

In summary, the recommendation engine performs the following three steps:

1. **Generating demand samples:** The occurrences of demands (i.e., origins) are generated using a demand prediction model (part of DGS), but the destinations are generated based on historical distribution.
2. **Updating taxi locations and simulating unguided taxis:** At each execution, the current locations of all taxis are updated in the engine. For each demand sample, the unguided taxis transition from their current zone to another zone based on the transition probabilities calculated from the historical data. Each demand in a zone can be served by either guided or unguided taxis (if they are present). The assignment decisions are made probabilistically, and if a demand is picked up by an unguided taxi, it should be removed from the demand sample before we solve the final optimization model for the guided taxis.
3. **Optimization for the guided taxis:** The model is solved for the guided taxis with the generated demand samples after removing the requests assigned to unguided taxis.

More details about the demand prediction model and the experiments can be found in following papers.

1. Jha, Shashi Shekhar, Shih-Fen Cheng, Meghna Lowalekar, Nicholas Wong, Rishikeshan Rajendram, Trong Khiem Tran, Pradeep Varakantham, Nghia Truong Trong, and Firmansyah Bin Abd Rahman. "Upping the game of taxi driving in the age of Uber." In Thirty-Second AAAI Conference on Artificial Intelligence. 2018.
2. Jha, Shashi Shekhar, Shih-Fen Cheng, Meghna Lowalekar, Nicholas Wong, Rishikeshan Rajendram, Pradeep Varakantham, Nghia Truong Trong, and Firmansyah Bin Abd Rahman. "A Driver Guidance System for Taxis in Singapore." In Proceedings of the 17th International Conference on Autonomous

Agents and MultiAgent Systems, pp. 1820-1822. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

3. Cheng, Shih-Fen, Shashi Shekhar Jha, and Rishikeshan Rajendram. "Taxis strike back: A field trial of the driver guidance system." In Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, pp. 577-584. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

3.6 Summary

In this chapter, we presented multi-period two-stage optimization model to solve U-OLYMPIAD problems. We also presented, Benders Decomposition to handle the large scale nature of the problems. The experimental results on real world datasets show that our approaches significantly outperform the existing approaches. Finally, we also present the results from the real world deployment of the driver guidance system in which the recommendation engine was built based on the the work presented in the chapter.

Chapter 4

Neural Approximate Dynamic Programming Approach to Solve M-OLYMPIAD Problem

In this chapter ¹, we present the Neural Approximate Dynamic Programming approach to solve M-OLYMPIAD problem.

4.1 NeurADP: Neural Approximate Dynamic Programming

Figure 4.1 represents the overall framework used for solving the M-OLYMPIAD problem. As shown in the figure, the framework executes 6 steps at each decision epoch to assign incoming user requests to available servers. Existing myopic approaches only execute steps (A), (B), (D) and (F). The crucial steps (C) and (E) help in maximizing the expected long-term value of serving a request rather than its immediate value. To learn this long-term value, we model the M-OLYMPIAD problem using ADP and use deep neural networks to learn the value functions of

¹Neural Approximate Dynamic programming is a joint work which was primarily done by Sanket Shah. My contribution lies in modelling it in approximate dynamic programming framework instead of reinforcement learning framework.

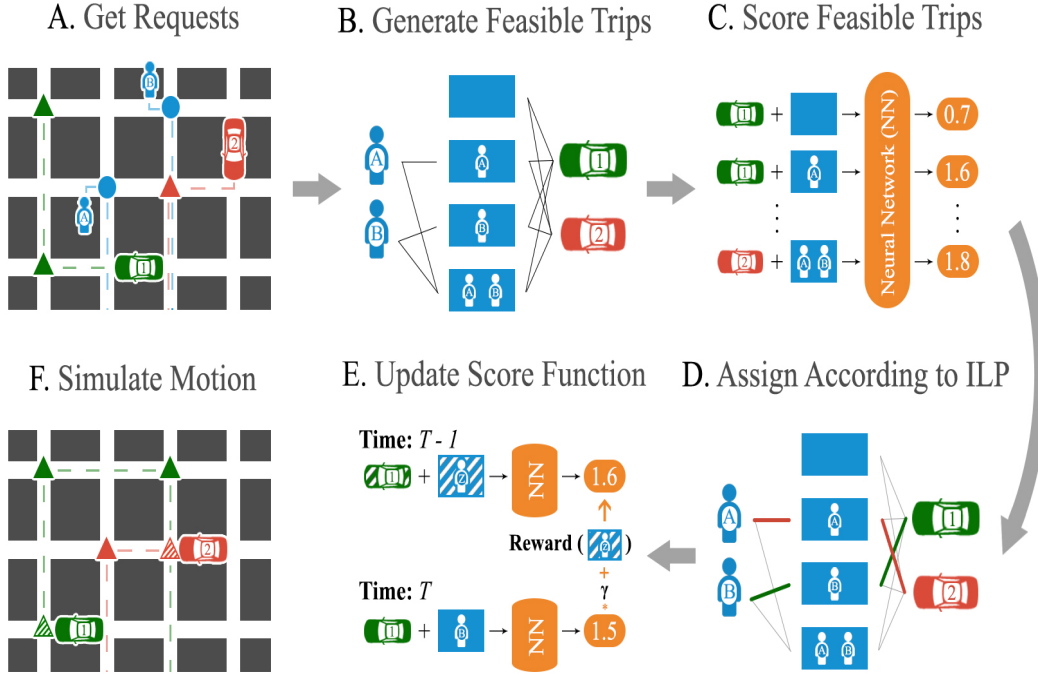


Figure 4.1: **Schematic outlining our overall approach.** We start with a hypothetical \mathcal{G} , $\mathcal{D}_{\nabla}^{\infty}$ and \mathcal{V} in (A). The grid represents a road network. The blue people and circles correspond to user requests and the nearest street intersection that they’re mapped to respectively. The blue dotted lines represent the shortest path between the pick-up and drop-off points of a request. The red and green triangles correspond to existing pick-up/drop-off points for the red and green servers respectively. The dotted lines describe their current trajectory. In (B) we map the requests and their combinations to servers that can serve them under the constraints defined by τ and λ to create feasible actions using the approach presented in (Alonso-Mora, Samaranayake, et al., 2017). In (C), we score each of these feasible actions using our Neural Network Value Function. In (D), we create a mapping of requests to servers that maximizes the sum of scores generated in (C) using the Integer Linear Program (ILP) in Table 4.1. In (E), we use this final mapping to update the score function (Section 4.1.4). In (F), we simulate the motion of servers until the next epoch either based on their current trajectories or a re-balancing strategy. This process then repeats for the next decision epoch.

post-decision states.

In this section, we first indicate key challenges that preclude direct application of existing ADP methods. We next provide the ADP model for the M-OLYMPIAD problem and describe our contributions in using neural function approximations for scalable and effective policies in M-OLYMPIAD problem.

4.1.1 Departure From Past Work

Approximate Dynamic Programming has been used to model many different transportation problems such as fleet management (Simao et al., 2009), ambulance allocation (Maxwell, Restrepo, Henderson, & Topaloglu, 2010) etc. While we also model our M-OLYMPIAD problem using ADP, we cannot use the solutions from past work for the following reasons:

1. ***Non-trivial generation of feasible actions:*** In using ADP to solve the unit-capacity fleet management problem, the action for a single empty server is to match a single request. Computing the feasible set of requests for a server is a straightforward and the best action for all servers together can then be computed by solving a Linear Program (LP). In the case of the M-OLYMPIAD problem, multiple requests can be assigned to a single empty or partially filled server. Generating the set of feasible actions, in this case, is complex and real-time solutions to this problem have been the key challenge in literature on myopic solutions to ride-pooling. In this chapter, we use the approach proposed by (Alonso-Mora, Samaranayake, et al., 2017) to generate feasible actions for a single server (Section 4.1) and then use an Integer Linear Program (ILP) to choose the best action (Table 4.1) over all servers.
2. ***Inability to use LP-duals to update the value function:*** Past work in ADP for fleet management (Simao et al., 2009) uses the dual values of the matching LP to update the parameters of their value function approximation. However, choosing the best action in the M-OLYMPIAD problem requires solving an Integer Linear Program (ILP) that has bad LP-relaxations. As a result, we cannot use

duals to update our value function. Instead, we show the connection between ADP and Reinforcement Learning (RL), and use the more general Bellman update used in RL to update the value function (Section 4.1.4).

3. ***Curse of Dimensionality:*** Past work in ADP for transportation problems addresses the curse of dimensionality by considering the value function to be dependent on a small set of hand-crafted attributes (e.g., aggregated number of servers in each location) rather than on the states of a large number of servers. Hand-crafting of state attributes is domain-specific and is incredibly challenging for a complex problem like M-OLYMPIAD, where aggregation of servers is not a feasible attribute (as each server can have different number of passengers going to multiple different locations). Instead, we use a Neural Network based value function to automatically learn a compact low dimensional representation of the large state space.
4. ***Incorporating Neural Network value functions into the optimization problem:*** Past work in ADP for fleet management uses linear or piece-wise linear value function approximations that allow for the value function to be easily integrated into the matching LP. Non-linear value functions (such as neural networks) cannot be integrated in this way, however, as they would make the overall optimization program non-linear. In Section 4.1.3), we address this issue by using a two-step decomposition of the value function that allows it to be efficiently integrated into the ILP as constants.
5. ***Challenges of learning a Neural Network value function:*** In Deep Reinforcement Learning literature (Mnih et al., 2015), it has been shown that naive approaches to approximating Neural Network value functions are unstable. Additionally, training them requires millions of samples. To address this, we propose a combination of methodological and practical solutions in Section 4.1.5.

The combination of using a Neural Network value function (instead of linear approximations) and updating it with a more general Bellman update (instead of LP-duals) represents a general alternative to past ADP approaches that we term Neural

ADP (NeurADP).

4.1.2 Approximate Dynamic Programming Model for the M-OLYMPIAD problem

We model the M-OLYMPIAD by instantiating the tuple in Section 2.3.

S: The state of the system is represented as $s_t = (r_t, u_t)$ where r_t is the state of all servers and u_t contains all the requests waiting to be served. A server $r \in \mathcal{V}$ at decision epoch t is described by a vector $r_t^i = (\mu^i, t, L^i)$ which represents its current trajectory. Specifically, it captures the current location (μ^i), time(t) and an ordered list of future locations (along with the cut-off time by which each must be visited) that the server has to visit (L^i) to satisfy the currently assigned requests. Each user request j at decision epoch t is represented using vector $w_t^j = (o^j, e^j)$ which captures its origin and destination.

A: For each server, the action is to assign a group of users from the set \mathcal{D} to it. These actions should satisfy:

1. **Constraints at the server level** - satisfying delay constraints (τ, λ) and server capacity constraints
2. **Constraints at the system level** - Each request is assigned to at most one server.

Handling exponential action space: To reduce the complexity, feasible actions are generated in two steps. In the first step, we handle server-level constraints by generating a set of feasible actions (groups of users) for each server. To do this efficiently, we first generate an RTV (Request, Trip, Vehicle/Server) graph using the algorithm by Alonso *et.al.* (Alonso-Mora, Samaranayake, et al., 2017). We use \mathcal{F}_t^i to denote the set of feasible actions generated for server i at decision

epoch t .

$$\mathcal{F}_t^i = \{f^i \mid f^i \in \cup_{c'=1}^{c^i} [\mathcal{D}]^{c'}, \text{PickUpDelay}(f^i, i) \leq \tau, \\ \text{DetourDelay}(f^i, i) \leq \lambda\}$$

To ensure that system-level constraints are satisfied, we solve an ILP that considers all servers and requests together. Let $a_t^{i,f}$ denote that server i takes action f at decision epoch t . Then, the decision variables $a_t^{i,f}$ need to satisfy following constraints:

$$\sum_{f \in \mathcal{F}_t^i} a_t^{i,f} = 1 \quad \forall i \in \mathcal{V} \quad (4.1)$$

$$\sum_{i \in \mathcal{V}} \sum_{f \in \mathcal{F}_t^i; j \in f} a_t^{i,f} \leq 1 \quad \forall j \in \mathcal{D}^t \quad (4.2)$$

$$a_t^{i,f} \in \{0, 1\} \quad \forall i, f \quad (4.3)$$

Constraint (4.1) ensures that each server is assigned a single action and constraint (4.2) ensures that each request is a part of, at most, one action. Together, they ensure that a request can be mapped to at most one server.

We use A_t denote the set of all actions that satisfy both individual and system-level constraints at time t and $a_t \in A_t$ to denote a feasible action in this set.

During training time, \mathcal{D}^t is populated from $\xi_t^{D,k}$, depending on the sample used in the episode in the algorithm 3. During test time, it is populated using \mathcal{D}_r^1 for current decision epoch.

ξ : As in previous work, exogenous information ξ_t represents the user demand that arrives between time $t - 1$ and t .

T : The transition function T^a defines how the server state changes after taking an action. In the case of the M-OLYMPIAD problem, all user requests that are not assigned are lost (Alonso-Mora, Samaranyake, et al., 2017). Therefore, the

AssignmentILP(t):

$$\begin{aligned} \max \quad & \sum_i \sum_{f \in \mathcal{F}_t^i} o_t^{i,f} \cdot a_t^{i,f} + V^i(T^{i,a}(r_t^{i,a}, f)) \cdot a_t^{i,f} \quad (4.4) \\ \text{subject to} \quad & \text{Constraints (4.1) - (4.3)} \end{aligned}$$

Table 4.1: Optimization Formulation for assignment of servers to feasible actions

user demand component of post-decision state will be empty, i.e., $u_t^a = \phi$.

$$T^a(s_t, a_t) = r_t^a \quad (4.5)$$

Here, r_t^a denotes the post decision state of the servers. We use $T^{i,a}(s_t^i, a_t^i) = r_t^{i,a}$ to denote the transition of individual servers. At each decision epoch, based on the actions taken, $(\mu^i, t, L^i) \forall i$ are updated and are captured in $r_t^{i,a}$. Each server has a fixed path corresponding to each action and as a result the transition above is deterministic.

O: When server i takes a feasible action f at decision epoch t , its contribution to the objective is $o_t^{i,f}$. For the objective of maximizing the number of requests served, $o_t^{i,f}$ is the number of requests that are part of a feasible action f . The objective function at time t is as follows:

$$o_t(s_t, a_t) = \sum_{i \in \mathcal{V}} \sum_{f \in \mathcal{F}_t^i} o_t^{i,f} \cdot a_t^{i,f}$$

4.1.3 Value Function Decomposition

Non-linear value functions, unlike their linear counterparts, cannot be directly integrated into the matching ILP. One way to incorporate them is to evaluate the value function for all possible post-decision states and then add these values as constants. However, the number of post-decision states is exponential in the number of resources/servers.

To address this, we propose a two-step decomposition of our overall value function that converts it into a linear combination over individual value functions asso-

Algorithm 3 NeurADP ()

- 1: Initialize: replay memory M , Neural value function V
(with random weights θ)
 - 2: **for** each episode $1 \leq n < |\xi^D|$ **do**
 - 3: Initialize the state s_1^n by randomly positioning servers.
 - 4: Choose a sample path $\xi^{D,n}$
 - 5: **for** each step $1 \leq t \leq Q$ **do**
 - 6: Compute the feasible action set \mathcal{F}_t based on s_t^n .
 - 7: Solve the ILP in Table 4.1 to get best action a_t^n .
(Add the Gaussian noise for exploration.)
 - 8: Store (r_t^n, \mathcal{F}_t) as an experience in M .
 - 9: **if** t % updateFrequency == 0 **then**
 - 10: Sample a random mini-batch of experiences
 from M
 - 11: **for** each experience e **do**
 - 12: Solve the ILP in Table 4.1 with the information
 from experience e to get the objective value y^e
 - 13: **for** each server i **do**
 - 14: Perform a gradient descent step on $(y^{e,i} - V(r_t^{i,n}))^2$ with respect to
 the network parameters θ
 - 15: Update: $s_t^{a,n} = T^a(s_t^n, a_t^n)$, $s_{t+1}^n = T^\xi(s_t^{a,n}, \xi_{t+1}^n)$
-

ciated with each server²:

- ***Decomposing joint value function based on individual servers' value functions:*** We use the fact that we have rewards associated with each server to decompose the joint value function for all servers' rewards into a sum over the value function for each server's rewards. The proof for this is straightforward and follows along the lines of (Russell & Zimdars, 2003).

$$V(r_t^a) = \sum_i V^i(r_t^a)$$

- ***Approximation of individual servers' value functions:*** We make the assumption that the long-term reward of a given server is not significantly affected by the specific actions another server makes in the current decision epoch. This makes sense because the long-term reward of a given server is affected by the interaction between its trajectory and that of the other servers and, at a macro level, these do not change significantly in a single epoch. This as-

²As mentioned in equation (4.5), the post-decision state only depends on the server state. Therefore, $V(s_t^a) = V(r_t^a)$.

sumption allows us to use the pre-decision, rather than post-decision, state of other servers.

$$V^i(r_t^a) = V^i(\langle r_t^{i,a}, r_t^{-i,a} \rangle) \approx V^i(\langle r_t^{i,a}, r_t^{-i} \rangle)$$

Here, $-i$ refers to all servers that are not server i . This step is crucial because the second term in the equation above r_t^{-i} can now be seen as a constant that does not depend on the exponential post-decision state of all servers.

Therefore, the overall value function can be rewritten as:

$$V(r_t^a) = \sum_i V^i(\langle r_t^{i,a}, r_t^{-i} \rangle)$$

We evaluate these individual V^i values for all possible $r_t^{i,a}$ and then integrate the overall value function into the ILP in Table 4.1 as a linear function over these individual values. This reduces the number of evaluations of the non-linear value function from exponential to linear in the number of servers.

4.1.4 Value Function Estimation for NeurADP

To estimate the value function V over the post-decision state, we use the Bellman equation (decomposed in the ADP as equation (2.1) and (2.2)) to iteratively update the parameters of the function approximation. In past work (Simao et al., 2009), the parameters of a linear (or piece-wise linear) value function were updated in the direction of the gradient provided by the dual values at every step. Hence, the LP-duals removed the need to explicitly calculate the gradients in the case of a linear function approximation.

Given that we use a neural network function approximation and require an ILP (rather than an LP), we cannot use this approach. Instead, we use standard symbolic differentiation libraries (Abadi et al., 2015) to explicitly calculate the gradients associated with individual parameters. We then update these parameters by trying to

minimize the L2 distance between a one-step estimate of the return (from the Bellman equation) and the current estimate of the value function (Mnih et al., 2015), as shown in Algorithm 3.

4.1.5 Overcoming challenges in Neural Network Value

Function Estimation

In this section, we describe how we mitigate the stability and scalability challenges associated with learning neural network value functions through a combination of methodological and practical methods.

4.1.5.1 Improving stability of Bellman updates:

It has been shown in Deep Reinforcement Learning (DRL) literature that using standard on-policy methods to update Neural Network (NN) based value function approximations can lead to instability (Mnih et al., 2015). This is because the NN expects the input samples to be independently distributed while consecutive states in RL and ADP are highly correlated. To address these challenges, we propose using off-policy updates. To do this, we save the current state and feasible action set for each server $\forall_i (s_t^i, \mathcal{F}_t^i)$ during sample collection. Then, offline, we score the feasible actions using the value function and use the ILP create the best matching. Finally, we update the value function of the saved post-decision state with that of the generated next post-decision state. This is different from experience replay in standard Q-Learning because the state and transition functions are partly known to us and choosing the best action, in our case, involves solving an ILP. In addition to off-policy updates, we use standard approaches in DRL like using a target network and Double Q-Learning (Van Hasselt, Guez, & Silver, 2016).

4.1.5.2 Addressing the data scarcity:

Neural Networks typically require millions of data points to be effective, even on simple arcade games (Mnih et al., 2015). In our approach, we address this challenge

in 3 ways:

- Practically, we see that in the M-OLYMPIAD problem, the biggest bottleneck in speed is in generating feasible actions. To address this, as noted above, we directly store the set of feasible actions instead of recomputing them for each update.
- Secondly, we use the same Neural Network for the value function associated with each of the individual servers. This means that a single experience leads to multiple updates, one for each server.
- Finally, we use Prioritized Experience Replay (Schaul, Quan, Antonoglou, & Silver, 2015) to reuse existing experiences more effectively.

4.1.5.3 Practical simplifications:

Finally, based on our domain knowledge, we introduce a set of practical simplifications that makes learning tractable:

- Instead of using one-hot representations for discrete locations, we create a low-dimensional embedding for each location by solving the proxy-problem of trying to estimate the travel times between these locations.
- During training, we perform exploration by adding Gaussian noise to the predicted V_i values (Plappert et al., 2017). This allows us to more delicately control the amount of randomness introduced into the training process than the standard ϵ -greedy strategy.
- We don't use the pre-decision state of all the other servers to calculate the value function for a given server (as suggested in Section 4.1.3). Instead, we aggregate this information into the count of the number of nearby servers and provide this to the network, instead.

The specifics of the neural network architecture and training can be found in the appendix.

4.2 Summary

In this chapter, we presented the Neural Approximate Dynamic Programming approach to solve M-OLYMPIAD problems. The approach learns the expected future value of the assignment by learning the value function for each state. To ensure scalability, a two-step decomposition of the overall value function is proposed which converts it into a linear combination over individual value function associated with each server. We compare the performance of NeurADP approaches against TBF approach in chapter 6.

Chapter 5

Optimization Approaches to Solve M-OLYMPIAD Problems

In the previous chapter, we presented a NeurADP approach which can assign servers to combination of request while considering expected future value of current assignment. For generating the set of feasible request combinations, NeurADP approach uses a faster heuristic version of TBF approach by Alonso *et.al.* (Alonso-Mora, Samaranayake, et al., 2017). There are two drawbacks of the NeurADP approach:

1. NeurADP approach uses TBF approach by Alonso *et.al.* (Alonso-Mora, Samaranayake, et al., 2017) to generate feasible request combinations. Therefore, it can only compute the future value of those assignments which are generated by TBF.
2. NeurADP requires training a different network model for each dataset and for each change in input parameter.

To overcome these limitations, in this chapter, we first present our Zone path construction approach (ZAC) to efficiently generate request combinations in real-time. The approach outperforms the TBF approach by generating more combinations in real-time. Then in section 5.2 we provide the future demand driven approach ZACBenders which can assign incoming customer requests to the servers

while considering future information. This approach uses ZAC to compute the current set of assignments and is an optimization based approach which can work with different datasets and parameters as shown in the experimental results in the next chapter.

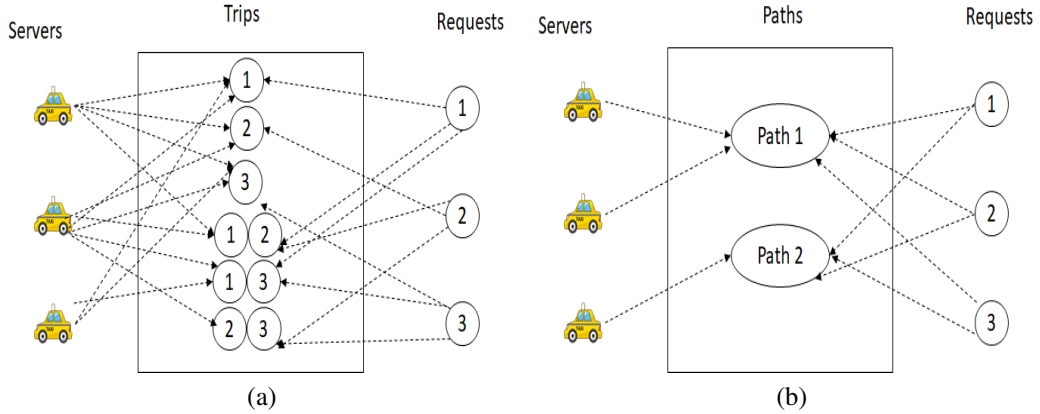


Figure 5.1: (a) Representation of RTV graph generated by the model in (Alonso-Mora, Samaranayake, et al., 2017) for capacity 2. (b) Representation of RPS graph generated by ZAC approach.

5.1 ZAC: A Zone pAth Construction Approach for Solving M-OLYMPIAD Problem

We propose a framework called ZAC (**Z**one **p**Ath **C**onstruction), that employs two crucial ideas to identify significantly more relevant trips in real-time:

- **Focus on zone paths instead of trips:** A zone path is a path that connects zones (a zone is an abstraction for multiple individual locations) and therefore it can group multiple requests that have "nearby" or "on the way" pick-ups and drop-offs. This focus on zone paths helps automatically capture multiple *relevant* combinations with one zone path.
- **Offline-online computation of zone paths:** Since, we focus on zone paths, we can generate partial zone paths offline. This helps capture more number of relevant combinations online in real-time, where the partial paths are completed.

Instead of an RTV (Request Trip Vehicle/Server) graph in Alonso *et al.*'s (Alonso-Mora, Samaranayake, et al., 2017) approach, we construct an RPS (Request Path Server) graph, where we associate requests and servers to zone paths. This is shown in Figure 5.1. Once the RPS is constructed, we then employ a scalable integer linear program to find the optimal assignment (e.g., maximize revenue, maximize the number of requests served or minimize the delay) of servers and requests to paths.

Given the importance of zone path to ZAC, we first define and explain about zone and zone path. We then describe the intuitive advantages of using zone paths and then we explain the ZAC algorithm.

Definition 1. Zone: *refers to an abstracted location obtained by clustering locations in set \mathcal{L} .*

In this work, we investigated Grid Based Clustering (GBC), Hierarchical Agglomerative Clustering with Complete Linkage (HAC_MAX) and Hierarchical Agglomerative Clustering with Mean Linkage (HAC_AVG) to cluster locations into zones. We use these methods as they do not require prior knowledge about the number of clusters and have been used in earlier works on similar problems (S. Ma et al., 2013; Hasan, Van Hentenryck, Budak, Chen, & Chaudhry, 2018).

Definition 2. Zone path: *refers to an ordered sequence of nodes, where each node corresponds to either a location from set \mathcal{L} or a zone.*

There are two key advantages to a zone path:

- Zone path represents multiple trips that have “nearby” or “on the way” pick-ups and drop-offs; and
- Zone path can be generated at different levels of granularity (e.g., individual locations, communities) depending on the time available.

Due to these two advantages, zone paths assist in identifying more relevant trips (combinations of requests) within a given amount of runtime. We further enhance the ability to identify more relevant trips within limited runtime, by generating zone

paths partially offline and completing them in real-time depending on the set of active requests.

We generate the zone path of time span τ offline and complete the rest of the zone path online. This is because requests can be picked up only in initial τ seconds¹. Therefore, partial zone paths generated offline automatically provide a pick-up order for the active requests. As a result, online, we only need to compute drop-off order while ensuring that the delay constraints are not violated. This is in contrast to Alonso *et al.*'s (Alonso-Mora, Samaranayake, et al., 2017) approach, where both pick-up and drop-off order along with the delay feasibility have to be computed online.

Intuitively, the inherent nature of zone paths to capture multiple relevant trips coupled with the extra time made available online due to offline computation of partial zone paths enables ZAC to consider significantly more relevant trips in real-time.

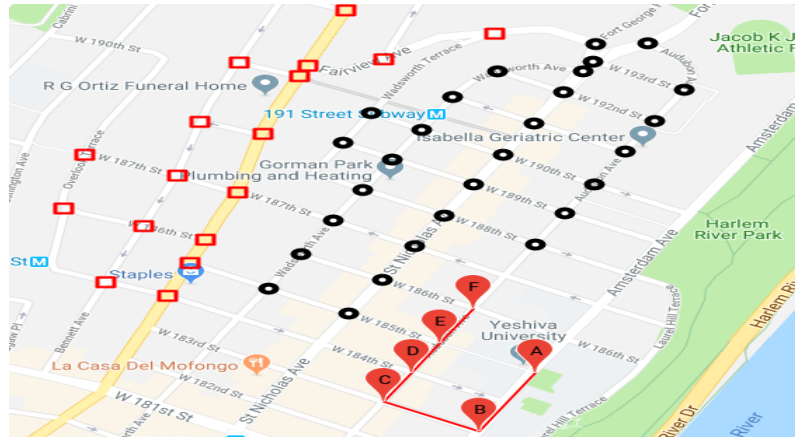
Due to abstraction of locations into zones, the travel time is approximately represented when considering zone paths. This can result in longer wait times or longer estimate of wait times than a path over locations in set \mathcal{L} . Customers prefer to have a shorter wait time pre-process (Dube-Rioux, Schmitt, & Leclerc, 1989; Maister et al., 1984), i.e., before pick-up in this case. Therefore, it is essential to reduce this approximation in travel time computation during pick-up. We reduce this approximation by generating offline partial paths at the level of locations. This is another benefit of having an offline partial path.

ZAC is an offline-online approach for solving the M-OLYMPIAD every few seconds on active requests and available servers by using offline generated partial paths. The key components of the ZAC algorithm are as follows:

- Offline: generation of all partial location paths of time span, τ from every location.

¹ τ is typically 300 and we experiment with values between 120-420 seconds.

- Online: generation of RPS graph by loading and processing offline partial paths, completing the partial paths and identifying edges in RPS graph.
- Online: finding optimal assignment of requests to paths to servers by using an efficient integer (0/1) linear optimization



Path A->B->C->D->E->F->Black Zone ->Red Zone
 The part A->B->C->D->E->F is generated offline.
 The decision to move from F to Black Zone then to Red Zone is taken online based on available requests.
 Black Zone – Consists of Black circled nodes.
 Red Zone – Consists of Red rectangle nodes.

Figure 5.2: Example Zone Based Path.

Example 1. Figure 5.2 provides an example of a zone path generated using ZAC. There is a partial zone path (generated offline) over individual locations (i.e., $A \rightarrow \dots \rightarrow F$) and the completion of that zone path (online) using larger zones (black and red).

5.1.1 Offline: Partial Paths Generation

The main challenge with generating a partial path at the level of individual locations – even for a time span of only maximum wait time, τ – is the time taken to generate all the paths. Therefore, we compute these partial paths (of span τ seconds) offline by generating all simple paths of duration τ in the network \mathcal{G} . The number of all possible paths grows exponentially with the increase in the value of τ and increase in the number of locations. In case all possible paths can not be generated due to memory constraints, we can employ a data driven approach (based

on historical data) to generate paths which have high likelihood of grouping large number of requests. These offline partial paths (\mathcal{P}_{off}) are stored by indexing on the start location and start time ($\mathcal{P}_{off}[l, t]$)². The start time associated with the path indicates the time at which the first node (location) in the path is visited. For a clear explanation, two paths starting at the same location but having different start times are considered different. This is because servers can become available at the same location but at different time. These offline partial paths are further indexed by the location and time of each node present in the path for quick online processing.

Algorithm 4 ZAC-Online()

```

1:  $t = starttime$  (in seconds)
2:  $\mathcal{P}_{off} = \bigcup_{\substack{l \in \mathcal{L}, \\ t' < \tau}} \mathcal{P}_{off}[l, t'] = \text{LoadOfflinePartialPaths}()$ 
3:  $\mathcal{T} = \text{LoadTravelTimes}()$ ,  $\mathcal{S}_p = \text{LoadShortestPaths}()$ 
4: while  $t < endtime$  do
5:    $t_1 = t - starttime$ 
6:   if  $(t_1) \% \Delta == 0$  then
7:      $\mathcal{D}_r^1, \mathcal{V} \leftarrow \text{GetCurrentDemand-ServerStatus}(t)$ 
8:      $\mathcal{P}, Pv, Pr, b, N = \text{GenerateRPSGraph}(t, \mathcal{P}_{off}, \mathcal{D}_r^1, \mathcal{V}, \mathcal{T}, \mathcal{S}_p)$ 
9:      $\text{SolveOptimization}(\mathcal{P}, Pv, Pr, b, N)$ 
10:     $\text{UpdateServerStatus}()$ 
11:     $t = t + 1$ 

```

5.1.2 Online

We now describe the crucial online component of ZAC that generates the RPS graph and finds the optimal match on the generated RPS graph. The pseudocode for the online component ZAC-Online is provided in Algorithm 4. After loading the offline computed partial paths, travel times and shortest paths, at every decision epoch, ZAC-Online considers the currently available batch of requests and current server status to find the optimal assignment in two steps: (1) Generation of the Request, Path and Server (RPS) graph and (2) Finding optimal match in RPS graph using a linear integer optimization model.

We now describe the two steps of ZAC in detail.

²We discretize the time at the level of 10 seconds.

5.1.2.1 Generation of the RPS graph

As shown in Algorithm 5, there are three key steps to RPS graph generation: (1) Online processing of Offline Partial Paths; (2) Online Partial Zone Path Completion; (3) Identifying edges in the RPS graph.

Algorithm 5 GenerateRPSGraph($t, \mathcal{P}_{off}, \mathcal{D}_r^1, \mathcal{V}, \mathcal{T}, \mathcal{S}_p$)

- 1: $\mathcal{P}'_{off}, \mathcal{R}' = \text{ProcessOfflinePartialPaths}(t, \mathcal{P}_{off}, \mathcal{D}_r^1, \mathcal{V}, \mathcal{T}, \mathcal{S}_p)$
 - 2: $\mathcal{P}, \mathcal{R}'' = \text{OnlineCompletion}(t, \mathcal{P}'_{off}, \mathcal{R}', \mathcal{T}, \mathcal{S}_p)$
 - 3: $\mathcal{P}, P_v, P_r, b, N = \text{IdentifyEdgesRPSGraph}(t, \mathcal{P}, \mathcal{D}_r^1, \mathcal{V}, \mathcal{R}'')$
 - 4: **return** $\mathcal{P}, P_v, P_r, b, N$
-

Online Processing of Offline Partial Paths: The offline generated partial paths are processed online based on the current available demand and server status (server location, currently assigned requests to server) as shown in Algorithm 6. Steps 1-2 ensure that we consider only those paths which start at a location and time where atleast one server is present and these paths are processed in parallel using multiple threads. The GetPathsFromIndex function returns the set of offline partial paths which visit the given location within given time interval and uses the pre-computed offline indexes for quick online retrieval. Step 12 stores the set of destination locations of the currently available requests grouped along the path (based on the pick-up). In addition to the destination location, we also store the lower and upper bound on the time by which the location should be visited. Similarly, in step 19, we store the destination locations of the requests previously assigned to servers. In, step 19, we consider only those paths which can potentially satisfy all the previously assigned requests for a server. This is because a server will be assigned to a path if and only if it can serve all the previously assigned requests. In addition, a server should deviate from its current path only if it can be assigned to a new request, therefore, we consider only those paths which can pick at least one of the newly available request.

Steps 14 and 19 ensure that if the drop-off location of request can be visited in the partial path, then it is considered in the processing. In the end, in steps 20-22, as an optimization, we only keep those locations in the partial paths which correspond

to a pick-up or drop-off location and update the travel time and path between the locations using \mathcal{T} and \mathcal{S}_p .

The offline generated partial paths significantly improve the scalability of completing the path online using exhaustive search. This provides more time online for considering more zone paths and hence more relevant trips.

Algorithm 6 ProcessOfflinePartialPaths($t, \mathcal{P}_{off}, \mathcal{D}_r^1, \mathcal{V}, \mathcal{T}, \mathcal{S}_p$)

```

1:  $\mathcal{P}'_{off} = [], Lt_v = \bigcup_{i \in \mathcal{V}} (\mu_i)$ 
2: Create  $H$  threads. Each thread  $h$  processes  $\mathcal{P}_{off}^h = \bigcup_{k' \in Lt_v^h} \mathcal{P}_{off}[k']$ , s.t.,  $Lt_v^a \cap Lt_v^b = \phi, \forall a \neq b$  and  $\cup_h Lt_v^h = Lt_v$ 
3: for each thread  $h$  do
4:    $\mathcal{V}' \subset \mathcal{V}, s.t., \forall i' \in \mathcal{V}', (\mu_{i'}) \in Lt_v^h$ 
5:   for  $j \in \mathcal{D}_r^1$  do
6:      $\mathcal{P}_{off}^{h,j} = \text{GetPathsFromIndex}(\mathcal{P}_{off}^h, o_j, a_j - t, a_j - t + \tau)$ 
7:     for each path  $k \in \mathcal{P}_{off}^{h,j}$  do
8:        $lb_j = a_j - t + \mathcal{T}(o_j, d_j), ub_j = lb_j + \lambda$ 
9:       if  $\mathcal{R}[k]$  contains  $d_j$  then
10:         $\mathcal{R}[k][d_j][1] = \max(\mathcal{R}[k][d_j][1], ub_j)$ 
11:       else
12:         $\mathcal{R}[k].add(d_j, (lb_j, ub_j))$ 
13:         $\mathcal{R}_p[k].add(o_j)$ 
14:        if  $lb_j < \tau$  and  $k$  visits  $d_j$  then
15:           $\mathcal{R}_p[k].add(d_j)$ 
16:        else if  $ub_j < \tau$  and  $k$  does not visit  $d_j$  then
17:           $\mathcal{R}[k].remove(d_j, (lb_j, ub_j))$ 
18:       for  $i \in \mathcal{V}'$  do
19:         $\mathcal{R}[k], \mathcal{R}_p[k] = \text{GetPathsForServer}(i, q_i, \mathcal{R}[k], \mathcal{R}_p[k], \mathcal{P}_{off})$ 
20:       for each path  $k$  do
21:        if  $|\mathcal{R}_p[k]| > 0$  then
22:          Remove nodes not in  $\mathcal{R}_p[k]$ , update  $\mathcal{R}_k$  using  $\mathcal{T}, \mathcal{S}_p$ 
23:           $\mathcal{P}_{off}^h.add(k)$ 
24:       for each thread  $h$  do
25:         $\mathcal{P}'_{off}.addAll(\mathcal{P}_{off}^h)$ 
26: return  $\mathcal{P}'_{off}, \mathcal{R}$ 

```

Online Partial Zone Path Completion: The partial paths generated offline are completed online using exhaustive search starting at the end location of the partial path as shown in the Algorithm 7. By online processing of offline paths (refer Algorithm

6), we can identify the requests that can be associated with each of the offline generated partial paths (based on their pick-up location). We use the destination locations of these requests to complete the remaining path online. As with each destination location, we also store a lower and upper limit on the time at which it should be visited, we only explore those branches in the search tree where these time limits are satisfied. The computational complexity of online partial path completion is dependent on the number of destination locations (size of $\mathcal{R}[k]$ in Algorithm 7) and can be significant, therefore, we use zones (and not individual locations) in this step. As mentioned before, by using zones, travel time is approximately represented which can result in additional delay for requests. The additional delay introduced is dependent on the size of the zones³ chosen. Therefore, to consider a trade-off between computational complexity and the quality of solution, we propose picking the zone sizes dynamically for each offline partial path. In order to fix the amount of dynamism in zone size, we use a parameter M that defines the number of different zone sizes that can be used in completion of offline partial paths. $M = 1$, implies static zone sizes, i.e., using zones of a fixed size for online completion of all offline partial paths. The zones of M different sizes are generated offline and in the step 6, depending on the number of destination locations and M available zone sizes, we decide the appropriate zone size for the partial path k ⁴.

As the partial paths are independent of each other, to further speed up the path generation process, we perform the online path completion process in parallel by creating multiple threads as shown in the pseudocode provided in Algorithm 7. Please note that the exhaustive search in step 8, will return multiple completed zone paths corresponding to a single partial path k .

For the objective of maximizing the number of requests served, the paths which start at the same location at the same time and serve a subset of requests served by

³The size of the zone is defined as the time taken to travel within a zone. Zone size 0 indicates that locations in set \mathcal{L} are used.

⁴In the experiments, we use $M = 4$ with zone sizes 0,60,120,300 and use the zone size which reduces the number of locations to 12 (this provides the best trade-off between runtime and solution quality and is determined based on experiments).

Algorithm 7 OnlineCompletion($t, \mathcal{P}'_{off}, \mathcal{R}', \mathcal{T}, \mathcal{S}_p$)

- 1: $\mathcal{P} = [], \mathcal{R} = []$
 - 2: Create H threads.
Each thread h processes $\mathcal{P}'_{off}{}^h \subset \mathcal{P}'_{off}$, s.t., $\mathcal{P}'_{off}{}^h \cap \mathcal{P}'_{off}{}^{h'} = \phi, \forall h \neq h'$ and $\cup_h \mathcal{P}'_{off}{}^h = \mathcal{P}'_{off}$
 - 3: **for** each thread h **do**
 - 4: $\mathcal{P}_{on}^h = [], \mathcal{R}_{on}^h = []$
 - 5: **for** each path k **do**
 - 6: $z = \text{getAppropriateZoneSize}(\mathcal{R}[k], M)$
 - 7: $\mathcal{R}' = \text{convert}(\mathcal{R}[k], z)$
 - 8: $\mathcal{P}_{on}^h, \mathcal{R}_{on}^h = \text{ExhaustiveSearch}(\text{end_node}(k), \mathcal{R}', \mathcal{P}_{on}^h, \mathcal{R}_{on}^h)$
 - 9: **for** each thread h **do**
 - 10: $\mathcal{P}.\text{addAll}(\mathcal{P}_{on}^h)$
 - 11: $\mathcal{R}.\text{addAll}(\mathcal{R}_{on}^h)$
 - 12: **return** \mathcal{P}, \mathcal{R}
-

another path are redundant. This is because, we check for capacity constraints in the optimization formulation presented in the next section. So a single path serving r requests can be used to represent all request combinations, $\sum_{i=1}^r \binom{r}{i}$. Therefore, the search tree in step 8 of Algorithm 7 can be pruned appropriately to search only for non redundant paths. This reduces the size of set \mathcal{P} which in-turn reduces the complexity of optimization formulation presented in the Section 5.1.2.2.

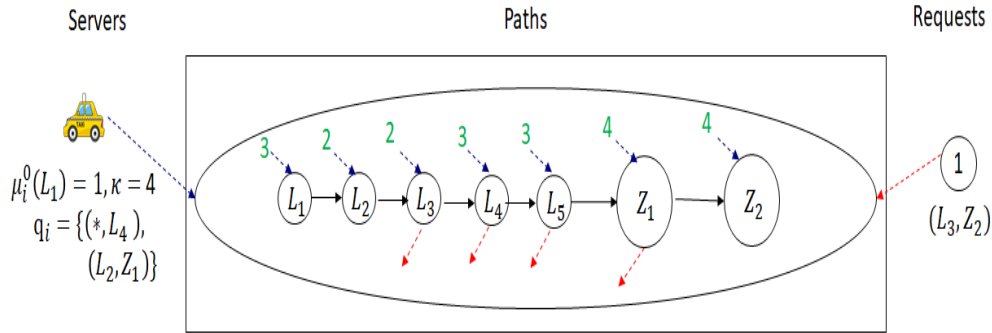


Figure 5.3: Representation of assignment of server and request to a zone path

Identifying Edges in the RPS Graph: Once the zone paths (\mathcal{P}) are created using the offline-online method described above, we construct the RPS (Request Path Server) graph by finding the set of requests and servers which can be assigned to each of the generated zone path. We use the information available from previous 2 steps about the requests and servers which can be assigned to these zone paths and

process the paths in parallel using multiple threads to speed up the computation. This step is essential as in Algorithm 6, when same destination location has different value for the upper limit on time in step 10, we take the maximum value. Therefore, in the path generated using Algorithm 7, the delay constraint may be violated for some requests in such cases.

In this step, we ensure that a request is assigned to a zone path, if and only if, the path visits the pick-up and drop-off location of a request within the delay constraints. The binary constants b (defined in Table 5.1 and used in optimization formulation presented next) are also populated in this step. A server i represented by the tuple (μ_i, q_i, κ) can be assigned to a zone path if the initial location of server is same as the starting location of the path, start time of the path is same as the availability time of server and currently assigned set of requests, q_i , can be served using the path. The server capacity κ along with q_i is used to compute the number of free seats (N) in the server at each zone/location.

Example 2. *Figure 5.3 shows a graphical view of the same for a single server, request and path. The path is represented using a sequence of locations/zones in the order in which they will be visited. In Figure 5.3, we use blue arrows to denote the incoming flow by server i assignment and green numbers indicate the number of free seats at the location/zone⁵ for server i . The red arrows indicate outgoing flow by request assignment.*

At each location, the optimization formulation presented next, will ensure that the outgoing flow (the number of requests assigned) is less than or equal to the incoming flow (total number of free seats in the servers assigned to the path), i.e., at each location/zone capacity constraints are satisfied.

⁵Number of free seats is computed by taking κ and q_i of the server into consideration. In figure, in the representation of q_i , we use * to indicate that customer is already present in the server and provide its drop-off location.

5.1.2.2 Finding Optimal match in RPS graph

We now describe the integer linear programming optimization formulation to optimize the assignment of requests and servers to zone paths. \mathcal{P} denotes the set of zone paths generated in previous step. \mathcal{P}_m^n is used to denote the n^{th} location/zone in zone path m . Let $Pr_j \subset \mathcal{P}$ denotes the set of paths which can serve request j while satisfying delay constraints. Similarly Pv_i denotes the set of paths which can be assigned to server i based on its current location and availability time (μ_i) and already assigned/picked-up requests q_i . Binary constants b_{jm}^n are set to 1 if the pick-up location of request j is visited but drop-off location/zone is not visited along path m by n^{th} location/zone. These are computed as part of generation of RPS graph as shown in previous section. Table 5.1 describes the notation used in the optimization formulation.

Variable	Description
x_{jm}	Binary variable denoting if the request $j \in \mathcal{D}_r^1$ is assigned to path m .
y_{im}	Binary variable denoting if the server i is assigned to the path m .
Pv_i	$Pv_i \subset \mathcal{P}$ denotes the set of paths which can be assigned to server i based on its current status μ_i and q_i .
Pr_j	$Pr_j \subset \mathcal{P}$ denotes the set of paths which can be assigned to request $j \in \mathcal{D}_r^1$.
b_{jm}^n	Binary constant: 1 if $\exists n' : n > n' \ P_m^{n'} = o_j$ && $\nexists n'' : n'' < n, n'' > n' \ P_m^{n''} = d_j$
$N(i, m, n)$	Number of free seats in the server i for path m at n^{th} location/zone.

Table 5.1: Notations

The objective of the optimization formulation described in Table 5.2 is to maximize the number of served requests. Constraints (5.2) and (5.3) ensure that each server and each request is assigned to at most one path. Constraint (5.4) ensure that for every path at every location/zone capacity constraints are satisfied. The capacity constraints can be violated only while picking up a new request, therefore, the constraint (5.4) is redundant for the locations/zones visited after τ duration.

The formulation is run at every decision epoch, i.e., after every Δ seconds. The solution of the optimization formulation provides assignment of servers and requests to paths. Using these assignments, we can perform the assignment of re-

SolveOptimization($\mathcal{P}, Pv, Pr, b, N$):

$$\max \sum_{j \in \mathcal{D}_r^1} \sum_{m \in Pr_j} x_{jm} \quad (5.1)$$

$$s.t. \sum_{m \in Pr_j} x_{jm} \leq 1 \quad :: \forall j \in \mathcal{D}_r^1 \quad (5.2)$$

$$\sum_{m \in Pv_i} y_{im} \leq 1 \quad :: \forall i \in \mathcal{V} \quad (5.3)$$

$$\sum_{j \in \mathcal{D}_r^1} x_{jm} \cdot b_{jm}^n \leq \sum_i y_{im} \cdot N(i, m, n) \quad :: \forall m \forall n \quad (5.4)$$

$$x_{jm} \in \{0, 1\} \quad :: \forall j \in \mathcal{D}_r^1, \forall m \in \mathcal{P} \quad (5.5)$$

$$y_{im} \in \{0, 1\} \quad :: \forall i \in \mathcal{V}, \forall m \in \mathcal{P} \quad (5.6)$$

Table 5.2: Optimization Formulation for ZAC

quests to servers ⁶. Once a server is assigned to a set of requests at any decision epoch, the assignment is not changed but the path of server can change at next decision epoch to accommodate additional requests. The current set of requests assigned to a server, q_i , limits the number of paths to which it can be assigned in subsequent decision epochs. The number of free seats in server i for path m at location/zone n , $N(i, m, n)$ is computed based on κ and q_i (as shown in Figure 5.3) and is 0 if $m \notin Pv_i$.

Similar to Alonso *et al.* (Alonso-Mora, Samaranayake, et al., 2017), we perform a re-balancing of unassigned servers to high demand areas at the end of optimization formulation.

5.2 ZACBenders: A non-myopic approach for solving M-OLYMPIAD

In this section, we first present the challenges in solving M-OLYMPIAD with future information (represented using samples, ξ^D in the M-OLYMPIAD model) and then present our non-myopic approach ZACBenders. The potential samples, ξ^D can be

⁶The paths assigned to server are also updated to keep only those locations which correspond to pick-up or drop-off location of assigned requests and update the travel time and path between the locations using \mathcal{T} and \mathcal{S}_p .

obtained by considering the demand observed in the past data. After considering future information, the goal is to find the assignment of servers to requests that maximizes the sum of objective value at the current decision epoch and the expected objective value for the future decision epochs.

5.2.1 Challenges in solving M-OLYMPIAD with future information

As shown in the Figure 5.4, to solve M-OLYMPIAD with future information (ξ^D) we need to assign servers and request to the zone paths at each decision epoch and for each sample ($\xi^{D,k}$). The state (i.e., location and requests being served) of server at each decision epoch should be updated based on the assignments (obtained by solving the RPS graph) at previous decision epochs. The paths at future decision epochs for each sample need to be generated by taking into account the requests present in the sample and the optimization problem should be updated to consider the assignments at future decision epochs for all the samples.

There are two major bottlenecks in the above process.

1. As described in the Section 5.1, the path generation process for a single decision epoch is challenging, so generating paths in real-time after considering requests in each sample for all future decision epochs for all possible updates to the servers states is computationally intractable.
2. The optimization formulation of assigning servers and requests to zone paths is an integer optimization problem. Including requests for all the samples at future decision epochs in this integer optimization formulation increases the number of variables and constraints which makes it difficult to solve online in real-time.

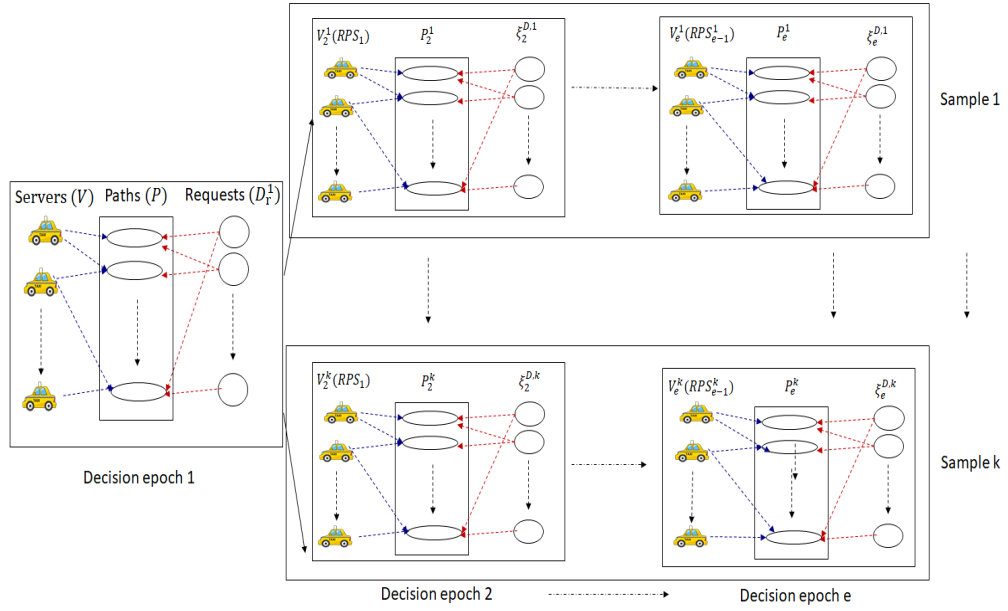


Figure 5.4: Assignment of servers and requests to zone paths over multiple samples of future demand. We use RPS_e^k to denote the RPS graph for decision epoch e in sample k . For the special case of first decision epoch, we denote the graph by RPS_1 . \mathcal{P}_e^k denote the set of paths generated for decision epoch e in sample k . $\xi_e^{D,k}$ denote the set of requests available at decision epoch e in sample k . $V_e^k(RPS_{e-1}^k)$ denote the state of servers at decision epoch e in sample k as a result of assignments obtained by solving the RPS graph at previous decision epoch in the same sample. Therefore, at each decision epoch for each sample, it is a tripartite matching between servers, paths and requests.

5.2.2 ZACBenders Approach

The overall flow of ZACBenders is similar to ZAC with the only difference in the step of finding the optimal assignment of servers and requests to zone paths. Table 5.3 highlights the difference between ZAC and ZACBenders approach. ZACBenders considers future information in the step of finding the optimal assignment of servers and requests to zone paths. As mentioned in Section 5.2.1, incorporating future information makes the problem challenging, therefore, we first provide a two-stage stochastic approximation, ZACFuture to handle these challenges. To efficiently solve the ZACFuture optimization formulation in real-time, we employ Benders Decomposition.

ZAC	ZACBenders
Offline	
1. Generation of all partial location paths of time span, τ from every location.	1. Same
Online	
1. Generation of RPS graph 2. Finding optimal assignment of requests and servers to zone paths by using the (0/1) integer optimization in Table 5.2.	1. Same 2. Process the requests in $ \xi^D $ samples to generate the second stage of the proposed two-stage approximation. 3. Follow the steps in Figure 5.6 to find the optimal assignment of requests and servers to zone paths while considering future information.

Table 5.3: Differences between ZAC and ZACBenders

5.2.2.1 Two-Stage Stochastic Approximation

As mentioned in the Section 5.2.1, it is difficult to solve M-OLYMPIAD with future information by generating paths considering requests in all samples, therefore, we propose a two-stage stochastic approximation ⁷. The first stage assigns servers and requests available at current decision epoch to the zone paths. In the second stage,

⁷We have experimented with many other approximations such as considering simple extensions of the existing zone paths to include request in samples and then assigning requests in samples to these extended paths but we describe in detail the approximation which worked best in practice. We acknowledge that it is possible to improve the performance even more by designing better approximations.

for each sample, instead of solving a tripartite matching problem (between servers, paths and requests) at each future decision epoch, we solve a weighted bipartite matching between servers and requests available for assignment at all future decision epochs. The tripartite matching is NP-hard but the weighted bipartite matching is polynomial time solvable, therefore, this approximation makes the second stage problem simpler. Specifically, we employ a decomposition of the resultant optimization problem (more details in Section 5.2.2.3) to get real-time performance.

We now describe the approximations which allow us to simplify the second stage problem for each sample by modelling it as a weighted bipartite matching problem.

- *Approximation 1:* Instead of using exact locations from set \mathcal{L} , we use abstracted locations, i.e., zones. The origin/destination of each request in sample and the location of each server is mapped to the zones.
- *Approximation 2:* A server will serve requests in samples (ξ^D) only after it finishes serving all the currently assigned requests. That is to say, we ignore that any future request can be inserted in the server's path and as a result a server is considered available again for assignment only after it reaches the end of zone path generated in first step.
- *Approximation 3:* Requests in samples (ξ^D) can be assigned to the same server if and only if they have identical origin zone, identical destination zone and the decision epoch at which they become available for assignment is also the same.

These approximations help in reducing the complexity of the problem but they still allow us to get a good estimate of the future because of the following reasons:

- The second approximation ensures that the servers which are considered for assignment at second stage are empty, i.e., they do not have any request assigned to them. So when the assignment optimization problem is solved (with limited look ahead duration of ρ), instead of assigning server to a longer duration path (which keeps it occupied for more than ρ duration), it will assign the server to those shorter duration zone paths (in the first stage) which redirect it to zones

where future requests are present. At any decision epoch, it is easier to assign multiple requests to an empty server as compared to a server which has a passenger on board. Therefore, in spite of ignoring that future requests can be picked up before dropping all currently assigned requests, this provides a good approximation.

- The third approximation (along with first approximation) ensures that the requests are grouped when they have nearby pick-up and drop-off locations. Though we will miss grouping the requests that have on the way pick-ups/drop-offs, by making this approximation and using an appropriate zone size, we will still be able to implicitly consider a subset of possible paths.

Formally, we map the origin and destination location of requests at future decision epochs to zones of size ⁸ Z^s and the elements in $\xi^{D,k}$ are grouped together based on the origin/destination zone and decision epoch. After grouping, each element j' of $\xi^{D,k}$ is represented using tuple $\langle o_{j'}^{z,k}, d_{j'}^{z,k}, e_{j'}^k, \eta_{j'}^k \rangle$ where $o_{j'}^{z,k}$ denotes the origin zone of the element j' in sample k , $d_{j'}^{z,k}$ denotes the destination zone of the element j' in sample k , $e_{j'}^k$ denotes the decision epoch at which element j' of sample k will be considered for assignment and $\eta_{j'}^k$ denotes the number of requests with origin $o_{j'}^{z,k}$, destination $d_{j'}^{z,k}$ at decision epoch $e_{j'}^k$ in sample k . We use \mathcal{A} to denote the set containing all possible pairs of zones of size Z^s and decision epochs $e+1, e+2, \dots, e+Q$ (if e is the current decision epoch). Each server is mapped to an element in set \mathcal{A} and is assigned requests in samples by using above approximations.

The assignment of servers to paths at first stage determines the zone and the decision epoch at which the servers will become available again for assignment (Approximation 2) and as all the servers have identical maximum capacity ⁹, in the second stage, we can group the servers based on the zone and the decision epoch at which they become available again for assignment. A server can be assigned to a request if and only if it can reach the origin location of request within the maximum

⁸As mentioned before, the size of the zone is defined as the time taken to travel within a zone.

⁹In the experiments, we show that even if all servers do not have identical maximum capacity, the approximation still works well. In that case, in the second stage, we take maximum capacity of each server as average of all server's maximum capacity.

allowed wait time, i.e. τ . Let \mathcal{E}^k denotes the set of all such assignment edges between servers and requests for sample k . To ensure that a server can be assigned at most κ requests and these requests can be grouped together as per Approximation 3, if $\eta_{j'}^k > \kappa$, we divide the element into $\lceil \frac{\eta_{j'}^k}{\kappa} \rceil$ subelements of element j' and allow each subelement to be assigned atmost once but if a server of type i' is assigned to r^{th} subelement of element j' in sample k , the weight received is given by

$$w_{i'j'r}^k = \begin{cases} 0 & \text{if } (i', j', r) \notin \mathcal{E}^k \\ \kappa & \text{if } (i', j', r) \in \mathcal{E}^k \text{ and } r < \lfloor \frac{\eta_{j'}^k}{\kappa} \rfloor \\ \eta_{j'}^k \% \kappa & \text{otherwise} \end{cases}$$

Therefore, this creates a bipartite graph with one side containing servers grouped based on their type (zone and decision epoch at which they become available based on the path assigned at first stage) and other side containing the request groups (all subelements of the element $j' \in \xi^{D,k}, \forall j'$). Figure 5.5 shows the graphical representation of the two-stage stochastic approximation, where first stage performs the tripartite matching between servers, requests and zone paths and at the second stage within each sample there is a bipartite matching.

We now describe the optimization formulation which can be used to solve this two-stage stochastic approximation.

5.2.2.2 ZACFuture: Optimization formulation to Solve the Two-Stage Stochastic Approximation

In this section, we describe the optimization formulation ZACFuture to solve the two-stage stochastic approximation presented in previous section. Table 5.4 presents the optimization formulation ZACFuture, which maximizes the number of requests served for the current decision epoch and the expected number of requests served over future demand samples.

We use $u_{i'j'r}^k$ to denote the assignment of server of type i' (i.e, the servers present in the zone $z_{i'}$ at decision epoch $e_{i'}$, where $(z_{i'}, e_{i'})$ is the tuple representation of element $i' \in \mathcal{A}$) to the r^{th} subelement of j' element of $\xi^{D,k}$. We also use $f(m)$ to

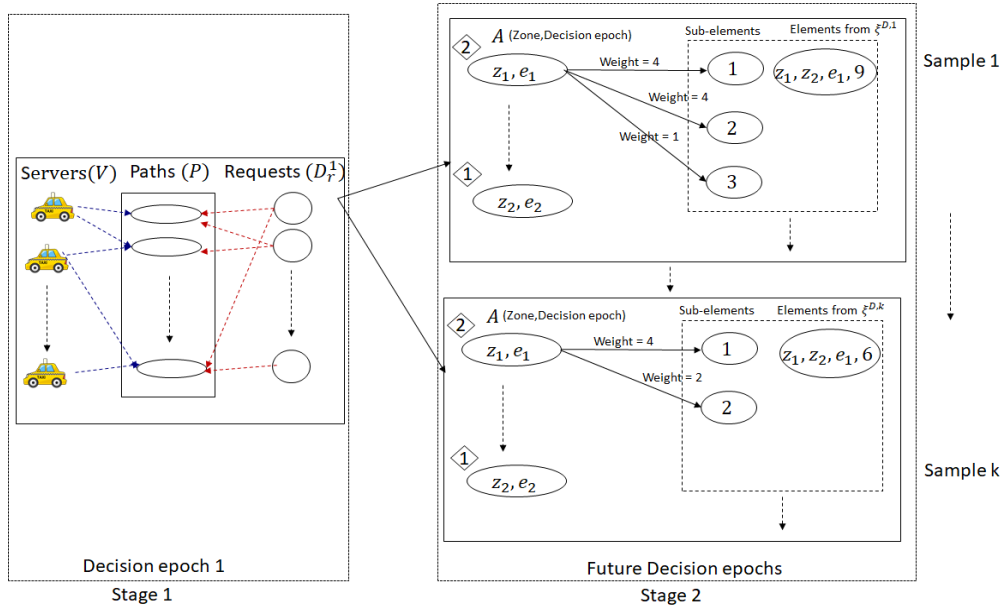


Figure 5.5: Two-Stage stochastic approximation for assignment of servers and requests to zone paths over multiple samples of future demand (For $\kappa = 4$). The zone and decision epoch mentioned in the oval are the zone and decision epoch at which the paths at the first stage in set \mathcal{P} ends. Therefore, servers have dropped all the assigned requests from set \mathcal{D}_r^1 (in first stage), once they reach the zone and decision epoch present in the oval. The number inside diamond represents the number of empty servers present in the zone and decision epoch mentioned in the oval box. At second stage, for each sample, a bipartite matching is performed between empty servers and available requests for all future decision epochs as compared to the tripartite matching between servers paths and requests for each sample in each decision epoch as shown in Figure 5.4.

denote the tuple (z_m, e_m) where z_m and e_m denote the zone and decision epoch at which the server will become available if it is assigned to path m . Constraints (5.9) ensure that the each subelement of j^{th} element of $\xi^{D,k}$ is assigned at most once and Constraints (5.12) ensure that the number of type i' servers assigned is less than the number of servers available of type i' .

ZACFuture $(\mathcal{P}, Pv, Pr, b, N, \xi^D)$:	
$\max \sum_{j \in \mathcal{D}_r^1} \sum_{m \in Pr_j} x_{jm} + \frac{1}{ \xi^D } \sum_{k=0}^{ \xi^D } \sum_{j' \in \xi^{D,k}} \sum_{r=0}^{\lceil \frac{\eta_{j'}^k}{\kappa} \rceil} \sum_{i' \in \mathcal{A}} w_{i'j'r}^k \cdot u_{i'j'r}^k \quad (5.7)$	$s.t. \sum_{m \in Pr_j} x_{jm} \leq 1 \quad \forall j \in \mathcal{D}_r^1 \quad (5.8)$
$\sum_{i' \in \mathcal{A}} u_{i'j'r}^k \leq 1 \quad \forall j' \in \xi^{D,k}, 0 \leq r < \lceil \frac{\eta_{j'}^k}{\kappa} \rceil, \forall 0 \leq k < \xi^D \quad (5.9)$	$\sum_{m \in Pv_i} y_{im} \leq 1 \quad \forall i \in \mathcal{V} \quad (5.10)$
$\sum_{j \in \mathcal{D}_r^1} x_{jm} \cdot b_{jm}^n \leq \sum_i y_{im} \cdot N(i, m, n) \quad \forall m \forall n \quad (5.11)$	$\sum_{j' \in \xi^{D,k}} \sum_{r=0}^{\lceil \frac{\eta_{j'}^k}{\kappa} \rceil} u_{i'j'r}^k \leq \sum_{i \in \mathcal{V}} \sum_{m: f(m)=i'} y_{im} \quad \forall i' \in \mathcal{A} \quad (5.12)$
$y_{im} \in \{0, 1\} \quad \forall i \in \mathcal{V}, m \in \mathcal{P} \quad (5.13)$	$x_{jm} \in \{0, 1\} \quad \forall j \in \mathcal{D}_r^1, m \in \mathcal{P} \quad (5.14)$
$u_{i'j'r}^k \in \{0, 1\} \quad \forall i' \in \mathcal{A}, j' \in \xi^{D,k}, 0 \leq r < \lceil \frac{\eta_{j'}^k}{\kappa} \rceil, 0 \leq k < \xi^D \quad (5.15)$	

Table 5.4: Optimization Formulation for Two-Stage stochastic approximation of M-OLYMPIAD with future samples

5.2.2.3 Benders Decomposition to Efficiently Solve ZACFuture Optimization Formulation

The complexity of the optimization formulation ZACFuture increases with the increase in the number of samples. To reduce this complexity, we exploit the following observation:

Observation 1. *In ZACFuture, once the assignment of servers to paths at the current decision epoch (y_{im}) is given, the optimization models for computing the as-*

signment of servers to requests at future decision epochs, $(u_{i'j'r}^k)$ for each of the samples k , are independent of each other.

The observation 1 allows us to use Benders Decomposition (Benders, 1962) to decompose the large optimization formulation into multiple smaller problems which can be solved in parallel. Benders Decomposition is a master slave decomposition technique where the master problem finds the solutions for the integer variables; and the slave problem(s) is (are) used to find the solutions to all other variables (which can take any value in the interval and need not be integers) while keeping the values of the integer variables fixed to the value obtained by the master problem. The values obtained by slave problems help in generating Benders cuts, which are added to the master problem and the master problem is solved again with these cuts to obtain an improved solution. This process is repeated till no more cuts can be added to the master problem. It is widely used to solve such two-stage stochastic problems (Murphy, 2013; Lowalekar, Varakantham, & Jaillet, 2018).

Based on Observation 1, y_{im} are the difficult variables as they impact the values assigned to all the other variables. x_{jm} are also difficult variables as they can take only integer values. As described in previous section, the second stage problem for each sample is a weighted bipartite matching problem. As the constraint matrix for weighted bipartite matching is totally unimodular, therefore, integrality constraints on the $u_{i'j'r}^k$ variables can be relaxed (Hoffman & Kruskal, 2010) after fixing the values of y_{im} variables. Therefore, the master problem obtains the assignments for the “difficult” integer variables (x_{jm} and y_{im}) and the slave problem(s) obtain the assignments to the $u_{i'j'r}^k$ variables.

For the master (Table 5.5), in the optimization provided in ZACFuture, we replace the part of the objective dealing with future variables, $\{u_{i'j'r}^k\}$ by the recourse function $\mathcal{Q}(\{y_{im}\}_{i \in \mathcal{V}, (m \in \mathcal{P}, k)})$ which becomes the objective function in the slave problems. The recourse function $\mathcal{Q}()$ needs to be computed for each value of y_{im} . In the slaves (Table 5.6), we consider the fixed values of y_{im} and to avoid confusion, we refer to them using the capital letter notation, Y_{im} .

Master($\mathcal{P}, P_v, P_r, b, N$):

$$\max \sum_{j \in \mathcal{D}_r^1} \sum_{m \in Pr_j} x_{jm} + \frac{1}{|\xi^D|} \sum_{k=0}^{|\xi^D|} Q(\{y_{im}\}_{i \in \mathcal{V}}, (j \in \mathcal{P}, k)) \quad (5.16)$$

$$s.t. \sum_{m \in Pr_j} x_{jm} \leq 1 \quad \forall j \in \mathcal{D}_r^1 \quad (5.17)$$

$$\sum_{m \in Pv_i} y_{im} \leq 1 \quad \forall i \in \mathcal{V} \quad (5.18)$$

$$\sum_{j \in \mathcal{D}_r^1} x_{jm} \cdot b_{jm}^n \leq \sum_i y_{im} \cdot N(i, m, n) \quad \forall m, \forall n \quad (5.19)$$

$$x_{jm} \in \{0, 1\} \quad \forall j \in \mathcal{D}_r^1, \forall m \in \mathcal{P} \quad (5.20)$$

$$y_{im} \in \{0, 1\} \quad \forall i \in \mathcal{V}, \forall m \in \mathcal{P} \quad (5.21)$$

Table 5.5: Optimization Formulation for Master problem - ZACBenders

SlavePrimal($\mathcal{P}, P_v, P_r, b, N, Y, k$):

$$\max \frac{1}{|\xi^D|} \sum_{j' \in \xi^{D,k}} \sum_{r=0}^{\lceil \frac{\eta_{j'}^k}{\kappa} \rceil} \sum_{i'} w_{i'j'r}^k \cdot u_{i'j'r}^k \quad (5.22)$$

$$s.t. \sum_{j' \in \xi^{D,k}} \sum_{r=0}^{\lceil \frac{\eta_{j'}^k}{\kappa} \rceil} u_{i'j'r}^k \leq \sum_i \sum_m Y_{im} \quad \forall i' \in \mathcal{A} \quad (5.23)$$

$$\sum_{i'} u_{i'j'r}^k \leq 1 \quad \forall j' \in \xi^{D,k}, 0 \leq r < \lceil \frac{\eta_{j'}^k}{\kappa} \rceil \quad (5.24)$$

$$u_{i'j'r}^k \in [0, 1] \quad \forall i' \in \mathcal{A}, j' \in \xi^{D,k}, 0 \leq r < \lceil \frac{\eta_{j'}^k}{\kappa} \rceil, 0 \leq k < |\xi^D| \quad (5.25)$$

Table 5.6: Optimization Formulation for Slave problem (Primal)- ZACBenders

The dual (Bertsimas & Tsitsiklis, 1997) of the primal slave problems are provided in Table 5.7, where α variables are the dual variables corresponding to the constraints (5.23) and β variables are the dual variables corresponding to the constraints (5.24).

SlaveDual ($\mathcal{P}, Pv, Pr, b, N, Y, k$):	
$\max \sum_{i'} \sum_i \sum_{m \in Pv_i} \alpha_{i'}^k \cdot Y_{im} + \sum_{j' \in \xi^{D,k}} \sum_r \beta_{j'r}^k$	(5.26)
$s.t. \alpha_{i'}^k + \beta_{j'r}^k \geq w_{i'j'r}^k \quad \forall i' \in \mathcal{A}, j' \in \xi^{D,k}, 0 \leq r < \lceil \frac{\eta_{j'}^k}{\kappa} \rceil$	(5.27)
$\alpha_{i'}^k \geq 0 \quad \forall i' \in \mathcal{A}$	(5.28)
$\beta_{j'r}^k \geq 0 \quad \forall i' \in \mathcal{A}, j' \in \xi^{D,k}, 0 \leq r < \lceil \frac{\eta_{j'}^k}{\kappa} \rceil$	(5.29)

Table 5.7: Optimization Formulation for Slave problem (Dual) - ZACBenders

The *weak duality theorem* (Bertsimas & Tsitsiklis, 1997) states that the solution to a maximization primal problem is always less than or equal to the solution of the corresponding dual problem. Therefore, using the concept of weak duality we can say that by taking the dual of the slave problems, we can find an upper bound on the value of the recourse function ($\mathcal{Q}()$)(objective of primal slave problem), in terms of the master problem variables y_{im} . These can then be added as optimality cuts to the master problem (Murphy, 2013) for generating better first stage assignments¹⁰.

Let θ^k be the approximation of $\mathcal{Q}()$ function then the master problem with optimality cuts is provided in the Table 5.8.

It should be noted that we are using y_{im} variables in the “master with optimality cuts” and not the fixed values, Y_{im} . In each iteration we solve the master problem and the computed y_{im} variable values are passed to the dual slave problems. After solving the dual slave problems, optimality cuts are generated. If the current values of $\theta^k(\forall k)$ satisfy the optimality cut conditions then we have obtained an optimal solution, else cuts are added to the master problem and the master problem is solved again. Figure 5.6 shows the flow diagram for the same.

¹⁰As the slave problems are always feasible for any value of the master variables we only need to add optimality cuts to the master problem.

MasterWithOptimalityCuts($\mathcal{P}, Pv, Pr, b, N$):

$$\max \sum_{j \in \mathcal{D}_r^1} \sum_{m \in Pr_j} x_{jm} + \frac{1}{|\xi^D|} \sum_k \theta^k \quad (5.30)$$

$$s.t. \theta^k \leq \sum_{i'} \sum_i \sum_{m \in Pv_i} \alpha_{i'}^k \cdot y_{im} + \sum_{j' \in \xi^{D,k}} \sum_r \beta_{j'r}^k \quad (5.31)$$

$$\sum_{m \in Pr_j} x_{jm} \leq 1 \quad \forall j \in \mathcal{D}_r^1 \quad (5.32)$$

$$\sum_{m \in Pv_i} y_{im} \leq 1 \quad \forall i \in \mathcal{V} \quad (5.33)$$

$$\sum_{j \in \mathcal{D}_r^1} x_{jm} \cdot b_{jm}^n \leq \sum_i y_{im} \cdot N(i, m, n) \quad \forall m \in \mathcal{P}, \forall n \quad (5.34)$$

$$x_{jm} \in \{0, 1\} \quad \forall j \in \mathcal{D}_r^1, \forall m \in \mathcal{P} \quad (5.35)$$

$$y_{im} \in \{0, 1\} \quad \forall i \in \mathcal{V}, \forall m \in \mathcal{P} \quad (5.36)$$

Table 5.8: Optimization Formulation for Master problem (with optimality cuts) - ZACBenders

The slave problems are independent of each other (Table 5.7) and are only connected by the choice of the master variables (“difficult” integer variables). Therefore, once the master variables are fixed, the slave problems can be solved in a parallel fashion.

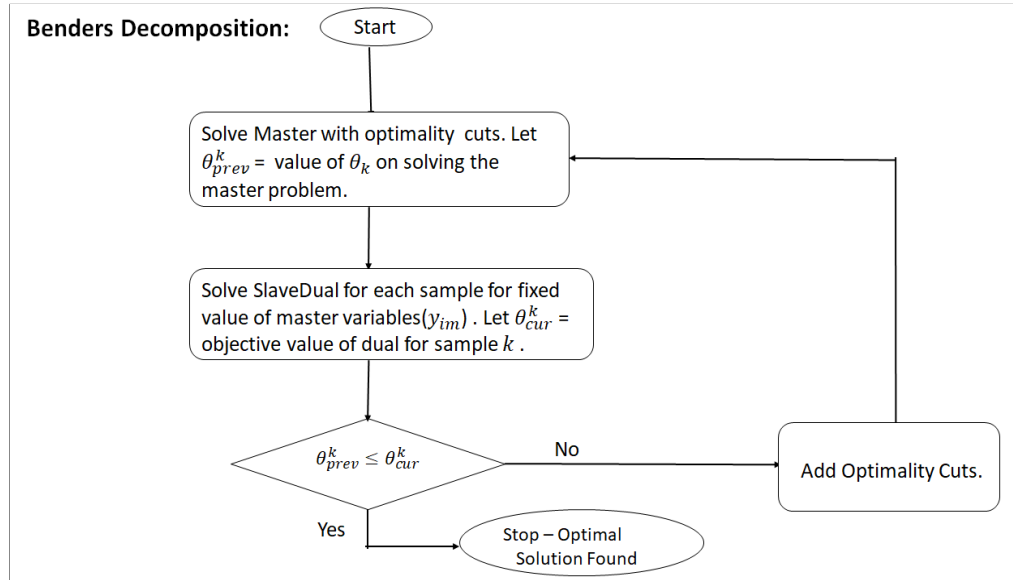


Figure 5.6: ZACBenders Approach: Finding Optimal Assignment of requests to paths to servers

5.3 Summary

In this chapter, we presented Zone path Construction approach to efficiently generate request combinations in real-time. We also propose ZACBenders which can consider future effects of current assignments by using a two-stage stochastic approximation. To efficiently solve the two-stage stochastic approximation in real-time, ZACBenders uses Benders Decomposition. In the next chapter, we compare ZAC, ZACBenders and NeurADP with TBF on real world and synthetic datasets.

Chapter 6

Experimental Results for M-OLYMPIAD Problems

In this chapter, we present the experimental results which compare the performance of our proposed approaches ZAC, ZACBenders and NeurADP against TBF¹. As NeurADP requires training a different model for each change in input parameter, using limited academic resources, it was not possible to run the exhaustive set of experiments with NeurADP. Therefore, we first show the detailed experimental results comparing the performance of TBF, ZAC and ZACBenders and then in section 6.3.1 we show the experimental results with NeurADP on limited set of parameters. For ZACBenders we kept a maximum timelimit of Δ seconds for each assignment but the Benders Decomposition can converge before the maximum timelimit is reached.

We evaluate the algorithms on following metrics: (1) Service Rate, i.e., percentage of total available requests served. (2) Runtime to compute a single step assignment. We experimented by taking demand distribution from two real world and one synthetic dataset.

Table 6.1 provides the outline for this section. We will show two main results that demonstrate the significant utility of our approaches:

¹The complexity of TBF increases with the increase in server capacity. It is not possible to run it up to optimality. Therefore, we run it with the heuristics mentioned in the paper (0.2 second for each server and keeping 30 servers for each request (but keeping all request edges)). We use the objective of maximizing the number of requests served for all algorithms. The objective can be changed to the objective of minimizing the delay or maximizing the revenue for both TBF and ZAC.

- Our myopic approach ZAC outperforms the current best myopic approach TBF. While the improvement varies, ZAC serves up to 4% more requests on real world datasets and up to 20% more requests on synthetic dataset.
- Our non-myopic approach ZACBenders further improves the performance of ZAC. It provides upto 14.7% improvement over TBF and 4.5% improvement over other non-myopic approach NeurADP.

Section	Description	Key Content
6.1	Datasets	Details on the datasets and different data fields used from the datasets.
6.2	Experimental Settings	Details on the different inputs, parameters and evaluation settings used.
6.3	Results on Real World Datasets	Describes our key results on real world datasets and shows the comparison of TBF, ZAC and ZACBenders for different parameters at on the three metrics of service rate and runtime.
6.3.1	Comparison with NeurADP	Describes our key result by comparing TBF, ZAC and ZACBenders with NeurADP on the limited set of parameters for which NeurADP trained models are available.
6.4	Results on Synthetic Dataset	Describes the performance of algorithms on specially created first and last mile scenarios where it is advantageous to explore more request combinations at a decision epoch.

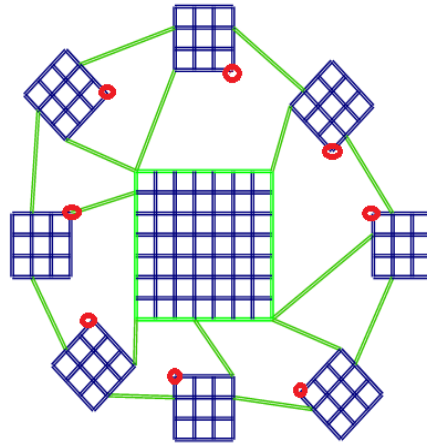
Table 6.1: Experiment Section Outline

6.1 Datasets

The first real world dataset is the publicly available New York Yellow Taxi Dataset (NYYellowTaxi, 2016), henceforth referred to as the NYDataset. The name of the other real world dataset can not be revealed due to confidentiality agreements. It is referred to as Dataset1. We use the street intersections as the set of locations \mathcal{L} . To find out the street intersections in real world dataset, we take the street network of the city from openstreetmap using osmnx with drive network type (Boeing, 2017). From these we remove the network nodes which do not have any outgoing edges, i.e., we take the largest strongly connected component of the network. For NYDataset, as considered in earlier works (Alonso-Mora, Samaranayake, et al., 2017), we only consider the street network of Manhattan as 75% of the requests have pick-up and drop-off locations in Manhattan. Moreover, less than 15% of the

total requests have pick-up and drop-off location in different boroughs of New York indicating that these boroughs can be solved independently.

Both real world datasets contain data of past customer requests for taxis at different time of the day and for different days of the week. From these datasets, we take the following fields: (1) Pick-up and drop-off locations (latitude and longitude coordinates) - These locations are mapped to the nearest street intersection. (2) Pick-up time - This time is converted to appropriate decision epoch based on the value of Δ . The travel time on each road segment of the street network is taken as the daily mean travel time estimate computed using the method proposed in (Santi et al., 2014).



(a)

Figure 6.1: Street network for synthetic dataset. Train stations are marked with red.

Dataset	Locations (\mathcal{L})	Edges (\mathcal{E})	Avg No. of Requests per day (on test days)	Avg No. of Requests per hour (Peak) (on test days)
NYDataset	4373	9540	313683	20910
Dataset1	21212	41424	403770	23664
Synthetic	192	640	173557	8578

Table 6.2: Details for different datasets

To simulate the scenario for on demand shuttle services (Shotl, 2018; Bee-line, 2016; Grab, 2018) having a small set of pick-up/drop-off points in a city, we also perform experiments on a synthetic dataset introduced by Bertsimas *et*

al. (Bertsimas, Jaillet, & Martin, 2019). The network (Figure 6.1) has one downtown area represented by the big square in center and 8 suburbs. We create a train station at one node of each suburb (marked by red circle) to simulate special cases of first and last mile transportation. At each decision epoch, requests are randomly generated by taking pick-up and drop-off location uniformly. In addition, every 180 seconds (frequency of arrival of train at the train stations), we generate first and last mile requests in each suburb (representing arrivals by train).

The number of nodes/locations, edges in the street network of the city and the number of requests present in each dataset are shown in the Table 6.2.

6.2 Experimental Settings

There are three different categories of experimental settings that have an impact on the performance of algorithms

1. **Inputs provided to all algorithms:** These include

- Number of Servers ($|\mathcal{V}|$): The number of servers used is dependent on the fleet size of the company. At the start of the experiment, empty servers are distributed uniformly at random in different locations. Based on the assignment obtained by algorithms at any decision epoch, the status of servers at the next decision epoch is updated. In the results section, we vary the number of servers to show the performance of algorithms for different number of servers.
- Maximum Capacity (κ): The maximum number of passengers which can be present in a server at any time.
- τ and λ : τ represents the maximum time within which the server should reach the origin location of request and λ denotes the maximum allowed travel delay for any request (in seconds).
- Decision epoch duration (Δ): This parameter determines how often, the algorithm should be executed and assignment decisions are made. For

example if $\Delta = 60$ seconds, then requests are batched for the duration of 60 seconds and the decision of serving or rejecting these requests is taken every 60 seconds by the algorithm. We vary this parameter to show the performance of algorithms for different values.

Table 6.3 show the values of different input parameters considered in the experiments.

Input Parameter	Values considered in Experiments
Δ (in seconds)	10,30,60
τ (in seconds)	120,180,300,420
λ (in seconds)	240,600,840,900
$ \mathcal{V} $	1000,2000,3000,5000,8000,10000
κ	1,2,3,4,8,10

Table 6.3: Inputs to all algorithms

2. **Parameters of the algorithm:** The parameters required by our algorithms are:

- **Clustering Method:** To construct zones from the set of locations \mathcal{L} , we compare the performance by using different clustering methods and different static zone sizes. Zone size is taken as the intra zone travel time (in seconds).
- **Number of Different Zone Sizes (for drop-off locations) (M):** In the online completion phase of the offline path, , instead of using a fixed zone size, ZAC dynamically decides the zone size to be used from a predefined fixed set of zone sizes. We vary the number of different zone sizes from which the ZAC algorithm picks the best zone size for a path.
- **Zone Size for Samples (\mathcal{Z}^s):** For samples we use a static zone size of 600 seconds. While it is possible to improve the performance of ZACBenders by using different zone size for different capacities and different value of τ and δ , we observe in the experiments that by using a fixed zone size, it is possible to get improvement across different parameters and different datasets.

- Number of Samples ($|\xi^D|$): While computing an assignment at decision epoch e , our non-myopic approach ZACBenders require samples of customer requests at decision epochs $e + 1, e + 2, \dots, e + Q$, where $Q = \lfloor \frac{\rho}{\Delta} \rfloor$, from past data (at the same decision epoch on the past days). We identify the right value for the number of samples through experiments as described in results section. Each sample correspond to past one day. For example if 10 samples are used, it means that requests from past 10 days are used to compute the expected future value in ZACBenders.
- LookAhead Duration (ρ): This determines how far ahead ZACBenders look into the future. If look ahead duration is 600seconds and current time is 09:00AM ZACBenders considers samples of customer requests up to 09:10AM.

Table 6.4 shows the different values for the parameters used in the experiments for ZAC and ZACBenders. To obtain right set of parameter values for ZAC and ZACBenders, we compare the performance of approaches by running them on 5 different weekdays from 21-03-2016 to 25-03-2016 and taking the average value over these five days. NeurADP is trained using the data for 8 weekdays (23 March - 1 April 2016).

Algorithm Parameter	Values considered in Experiments
M	2,4,6
Clustering Method	GBC, HAC_MAX,HAC_AVG
Number of Samples ($ \xi^D $)	1,3,5,8,10
Look Ahead Duration (in seconds) (ρ)	600, 900 ,1200 ,1500
Zone size for Samples (Z^s) (in seconds)	600

Table 6.4: Algorithm Parameter Settings

3. Evaluation Settings:

- Evaluation duration: We evaluate the performance of algorithm over 1 hour by varying different input and algorithmic parameters. For a subset of parameter combinations, we also compared the performance over 24

hours ².

- Number of days and time of evaluation: We performed experiments with requests at various times of the day, 8:00 AM, 3:00 PM, 6:00 PM, 12:00 AM and on different days. We evaluated the approaches by running them on 15 different weekdays between 04-04-2016 and 22-04-2016 and taking the average values over 15 days. These 15 days are different from the 5 days used to obtain the right set of algorithm parameters for ZAC and ZACBenders and 8 training days for NeurADP.

We conducted experiments with all the combinations of settings and inputs mentioned in this section. To avoid repeating similar results over and over again, we provide the representative results. All experiments are run on 24 core - 2.4GHz Intel Xeon E5-2650 processor and 256GB RAM. The algorithms ZAC and ZACBenders are implemented in Java and optimization models are solved using CPLEX 12.6. NeruADP is implemented in Python.

6.3 Results on Real World Datasets

In this section, we compare the number of requests served by TBF, ZAC and ZACBenders. We also compare the average time taken to compute an assignment by all the approaches.

We choose the best configuration for parameters of algorithms (justification provided in the Section 6.5). For ZAC and ZACBenders, we cluster locations into zones using HAC_MAX and use $M=4$ (with zone sizes 0,60,120,300). ZACBenders uses 5 samples with a look ahead duration of 15 minutes and the value of Z^s (zone size used in second stage) is taken as 600 seconds.

We first compare the service rate and runtime of TBF, ZAC and ZACBenders by varying different parameters on two real world datasets.

²As running all the algorithms for 24 hours over different set of parameters takes a long time and the difference in the performance of algorithms over 1 hour was following a similar trend as over 24 hours, we ran it for 24 hours only for a subset of parameters.

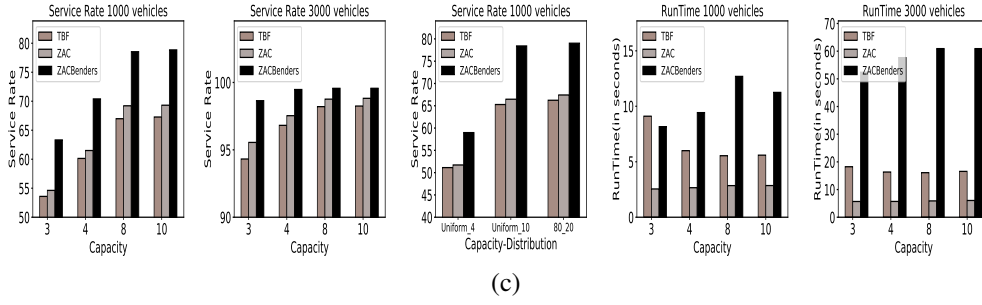


Figure 6.2: Comparison of ZACBenders, ZAC and TBF on NYDataset for $\tau=180$ seconds, $\lambda=600$ seconds and $\Delta = 60$ seconds

Effect of change in server capacity (κ) and number of servers ($|V|$): Figure 6.2 and Figure 6.3 show the service rate and runtime comparison of TBF, ZAC and ZACBenders for NYDataset and Dataset1 respectively at 8am (Peak time) .

For the change in the number of servers, we make the following observations:

- On Dataset1, the difference in the service rate obtained by ZAC and TBF increases as the number of servers increases from 3000 to 5000. One of the reasons is that TBF limits the number of servers considered for each request to 30, so the number of requests missed due to this limit will be more for higher number of servers. But on further increasing the number of servers to 10000, the gap between ZAC and TBF reduces. This is because, when more servers are available, it reduces the need of generating all combinations. On NYDataset, the difference between service rate obtained by ZAC and TBF is maximum for 1000 servers.
- The difference in service rate of ZACBenders and ZAC decreases as the number of servers are increased. This is because when more servers are available, they will be free even after executing current assignments at the current decision epoch, so future demands can be met irrespective of the current assignment. On Dataset1 for capacity 4, ZACBenders obtains 4.2% improvement over ZAC for 1000 servers, 3.27% improvement for 3000 servers and 2.24% improvement for 5000 servers. For 10000 servers, the service rate obtained by ZAC and ZACBenders is almost same. On NYDataset for capacity 4, the maximum improvement

obtained by ZACBenders over ZAC is 8.89% which is for 1000 servers.

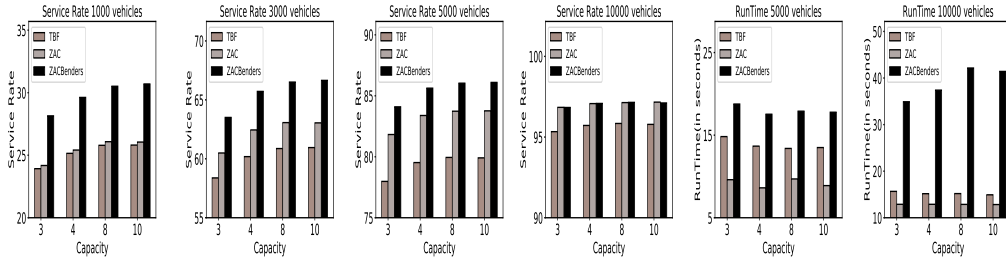


Figure 6.3: Comparison of ZACBenders, ZAC and TBF on Dataset1 for $\tau = 180$ seconds, $\lambda = 600$ seconds and $\Delta = 60$ seconds

Here are the key observations when server capacity is changed for a fixed number of servers:

- Service rate obtained by ZAC is more than TBF for both datasets. For capacity 4 with 1000 servers for NYDataset, the service rate obtained by ZAC is 1.36% more than the service rate obtained by TBF and for capacity 10 we obtain a gain of 2.03%. On the other hand for Dataset1 for capacity 4 with 5000 servers, the service rate obtained by ZAC is up to 4% more than the service rate obtained by TBF. On Dataset1, we do not observe much increase in service rate beyond capacity 4 due to large size of the network and longer travel times which allows fewer requests to be paired.
- ZACBenders improves the performance of ZAC by using future information. For capacity 4 with 1000 servers on NYDataset, it obtains 8.89% improvement over ZAC which increases to 9.5% for capacity 10. On Dataset1 for 1000 servers with capacity 4, ZACBenders obtains 4.2% improvement over ZAC which increases to 4.6% for capacity 10.
- While both ZAC and TBF can compute a solution in less than 20 seconds, the time taken by ZAC is much less than TBF. The time taken by ZACBenders is much more than the myopic algorithms TBF and ZAC but the service rate improvement compensates for the additional runtime.

We also experimented with all the servers having different maximum capacity. In this case, in ZACBenders, to compute the weight of edges in bipartite graph,

κ is taken as average of all server's maximum capacity. Figure 6.2c shows the results where server capacities are generated by taking different distributions. We experimented with following three distributions:

1. Uniform_4: The maximum capacity of each server is sampled uniformly between 1 to 4.
2. Uniform_10: The maximum capacity of each server is sampled uniformly between 1 to 10.
3. 80_20: 80% of the servers have maximum capacity as 4 and 20% of the servers have maximum capacity as 6. This is based on the observation that ridesharing companies like Uber, Lyft etc have majority of servers with maximum capacity 4 and some servers with maximum capacity 6.

In this case also, we observe that ZACBenders obtains improvement over myopic approaches. For Uniform_4, the improvement over ZAC is 7.26%, for Uniform_10 the improvement is 11.94% and for 80_20, the improvement obtained is 11.61%. The two-stage stochastic approximation in this case works better than all servers having identical maximum capacity as when servers have identical maximum capacity, the κ value used in second stage will be higher and it will allow more requests to group at future decision epoch causing the future value to be over estimated in some cases.

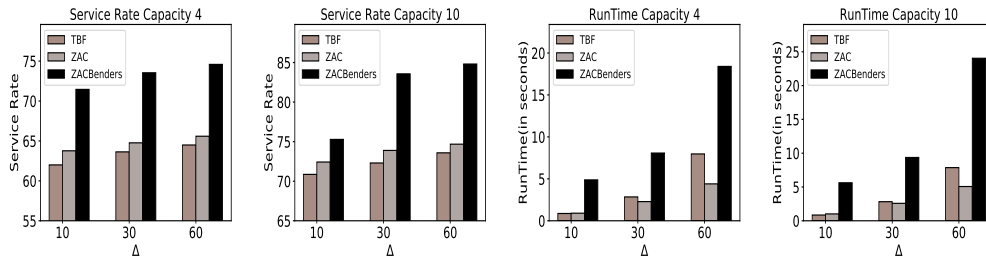


Figure 6.4: Comparison of ZACBenders, ZAC and TBF for NYDataset for 1000 servers and varying values of Δ , $\tau = 300$, $\lambda = 600$ seconds

Effect of change in value of Δ : We compare the service rate and runtime of algorithms for different values of Δ (Figure 6.4). Here are the key observations:

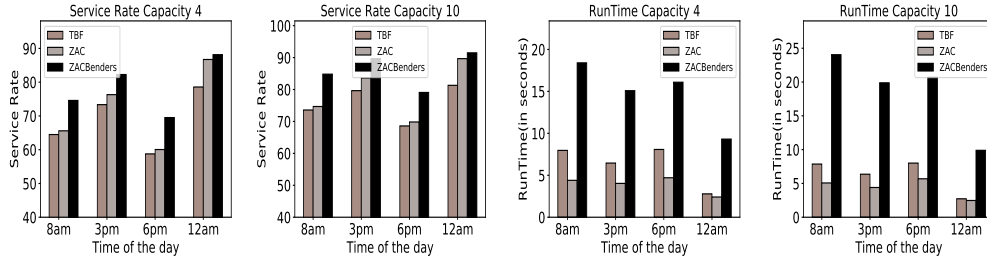


Figure 6.5: Comparison of ZACBenders, ZAC and TBF for NYDataset for 1000 servers and different time of the day. $\tau = 300$, $\lambda = 600$ seconds

- Service rate increases as the value of Δ increases. This is because more requests are available at each decision epoch which allows grouping more requests together.
- The difference between the service rate of ZACBenders and ZAC increases as the Δ increases. One of the reasons is that the value of Δ limits the time available for computation of assignments, when Δ value is low, less number of Benders decomposition iterations can be executed within time limit which affects the performance of ZACBenders.
- The time taken by TBF is much more than ZAC for larger Δ values due to the presence of more number of requests at each decision epoch.

Effect of time of the day: We compare the effect of time of day on the performance of algorithms (Figure 6.5). Here are the key observations:

- The service rate of ZAC is more than TBF in each time interval and ZACBenders further improves this service rate.
- The difference between service rate of ZAC and TBF is more during non-peak hours (3pm and 12am). This is likely as there are less requests available at each decision epoch, so as opposed to peak time where there are more possibility of grouping requests across decision epochs, at non-peak times it is advantageous to explore more combinations at a single decision epoch. The other reason is that ZAC is able to rebalance servers better by assigning them to zone paths.

Effect of change in values of τ and λ : We show the service rate and runtime results for different values of τ and λ in Figure 6.6. Irrespective of the delay constraints,

service rate obtained by ZAC is either more or same as TBF and the runtime of ZAC remains less than TBF in all cases. The improvement in the service rate obtained by ZACBenders over ZAC is also consistent across different values of τ and λ . The time taken by ZACBenders increases as the value of τ and λ increases due to increase in the complexity of the optimization formulation.

On real datasets, ZAC obtains up to 4% gain in service rate over TBF across different parameter values. ZACBenders obtains nearly 10% improvement in service rate over ZAC on NYDataset and 5% improvement on Dataset1³. Typically, even a 0.5% gain is considered significant on real taxi datasets (as shown by a real car aggregation company (Xu et al., 2018)), so the gain obtained by our algorithms is a significant gain.

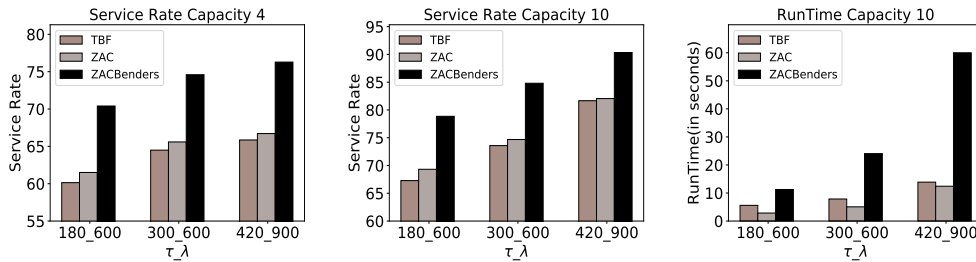


Figure 6.6: NYDataset - 1000 servers, $\Delta=60$ seconds.

6.3.1 Comparison with NeurADP

In this section, we compare TBF, ZAC, ZACBenders and NeurADP on NYDataset. Both ZACBenders and NeurADP can compute an assignment within maximum Δ seconds. As mentioned before, NeurADP requires training a neural network model for each change in input parameter and for each dataset and on academic computers due to limited resources, it is not possible to get the models trained for all the datasets and parameter settings. Therefore, we perform a comparison with NeurADP on the limited set of parameters for which trained model is available. We compare the performance of all algorithms over 24 hours as both ZACBenders and

³This gain further increases when evaluated over longer duration (24 hours) as opposed to 1 hour as seen in the results in the Section 6.3.1.

NeurADP consider future information and ignore requests at initial decision epochs to serve more requests in future and as a result achieve higher service rates when evaluated over longer durations.

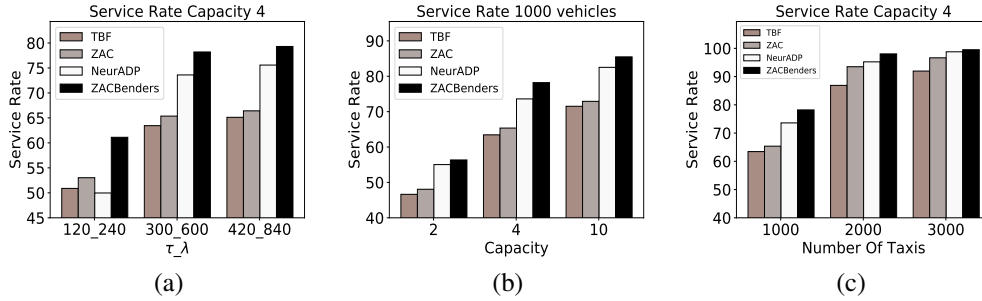


Figure 6.7: Comparison of service rate on NYDataset. (a) (b) Number of servers =1000 (b)(c) $\tau = 300, \lambda = 600$

Following are the key observations:

- ZACBenders consistently outperforms NeurADP across all parameters as shown in Figure 6.7.
- For faster training on academic computers, NeurADP uses a faster-heuristic version of TBF as baseline myopic algorithm and as a result does not generate all trips generated by TBF. While this heuristic of generating limited trips combined with learning the future value works well for higher value of τ and λ , it does not work well when τ is 120 seconds. As a result, the service rate of our myopic algorithm ZAC is 3% more than NeurADP and ZACBenders further obtains a 8% improvement in service rate over ZAC. This highlights the importance of considering future information along with using a myopic algorithm which can generate explore more combinations from currently available requests.
- For 1000 servers of capacity 4 with $\tau = 300$ and $\lambda = 600$ seconds, NeurADP serves 8.2% more requests than ZAC over 24 hours but ZACBenders servers 4.5% more requests than NeurADP. In this case, ZACBenders obtains 14.7% improvement over TBF. As shown in Figure 6.7, we obtain similar results for other parameter values.

6.4 Results on Synthetic Dataset

The real world taxi datasets can not capture the scenarios for on demand shuttle services (Shotl, 2018; Beeline, 2016; Grab, 2018) having a small set of pick-up/drop-off points in a city. These involve scenarios where many requests can be combined at each decision epoch. We represent these scenarios by simulating the case of first and last mile transportation in the synthetic network (details provided in experimental setup), where there are multiple requests at each decision epoch with either identical pick-up location and nearby drop-off locations or identical drop-off locations and nearby pick-up locations resulting in higher possibility of having large number of request combinations at a decision epoch.

The gain obtained by ZAC over TBF is even more significant in these scenarios as TBF will not be exploring all relevant combinations while ZAC can explore more combinations by using zone paths. ZACBenders provide a slight improvement over ZAC by using future information but in these scenarios, as the travel times are small and the pick-up and drop-off locations of requests are near each other, the major improvement is obtained by exploring more combinations at a single decision epoch.

We compare the service rate obtained by TBF, ZAC and ZACBenders with different number of servers and different capacities and make following observations:

- We observe that with 500 servers and capacity 10, ZAC can obtain 20.8% improvement in service rate over TBF. The gain reduces to 16% on increasing servers to 1000 as when more servers are available, it reduces the need of generating all combinations.
- The service rate obtained by ZACBenders and ZAC is almost same on this dataset as it is more important to explore more combinations in these scenarios. For 100 servers with capacity 10, ZACBenders obtains 2.2% improvement over ZAC and for 500 servers with capacity 4 ZACBenders obtains 2% improvement over ZAC.

These results demonstrate that ZAC is able to consider significantly more trips than TBF.

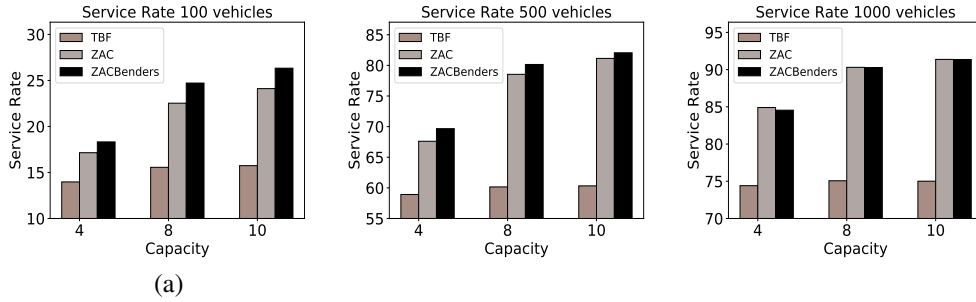


Figure 6.8: Synthetic Dataset with $\tau = 120, \lambda = 240$ and $\Delta = 60$ seconds

6.5 Justification for values of algorithmic parameter settings

In this section, we show the reason for using the fixed algorithmic parameter values (used in previous sections) for ZAC and ZACBenders.

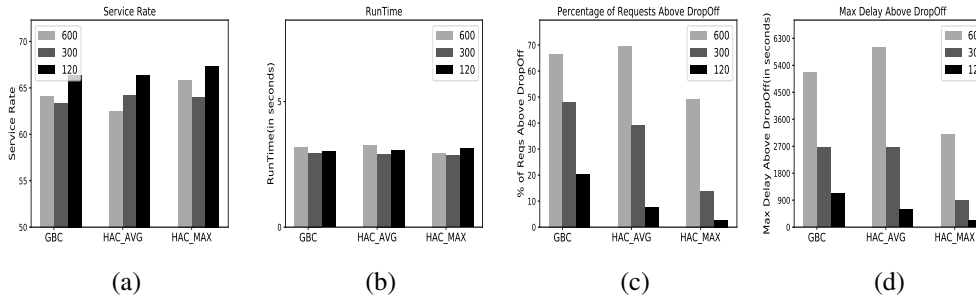


Figure 6.9: Comparison of service rate, runtime and abstraction error with different clustering methods and zone sizes for $M = 1$, number of servers = 1000, capacity = 10, $\tau = 300, \lambda = 600$ seconds

6.5.1 Identification of Right Clustering Method

We first conduct experiments by using different clustering methods, with $M = 1$, by varying the zone sizes. Zone size is taken as the intra zone travel time (in seconds). Figure 6.9 shows the comparison of GBC, HAC_MAX and HAC_AVG on

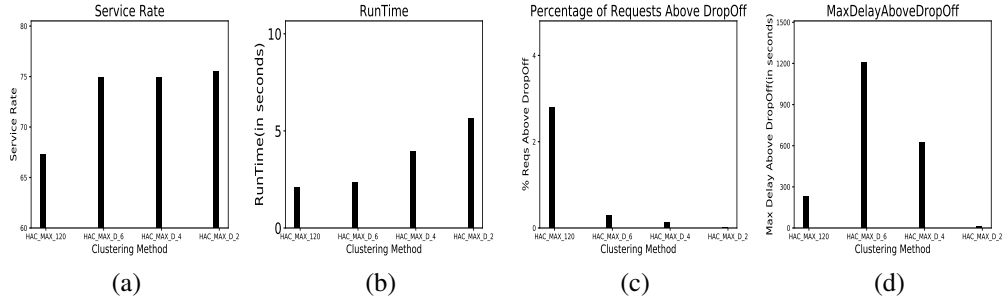


Figure 6.10: Comparison of service rate, runtime and abstraction error with different values of M and zone sizes for NYDataset, number of servers = 1000, capacity 10, $\tau = 300$, $\lambda = 600$ seconds

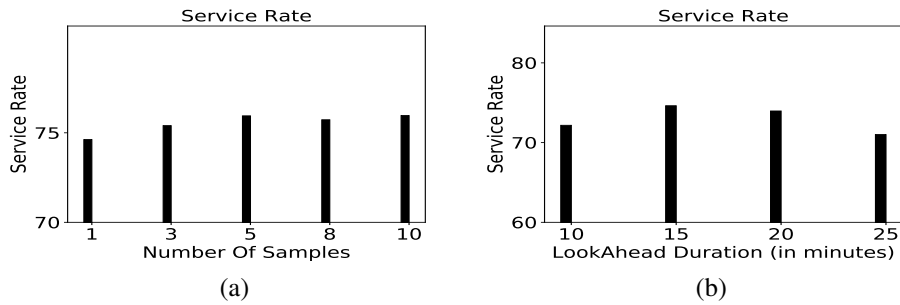


Figure 6.11: Comparison of service rate for different number of samples and lookahead duration number of servers = 1000, capacity = 4, $\tau = 300$, $\lambda = 600$ seconds

NYDataset. We compare the service rate, runtime and abstraction error with different clustering methods and different zone sizes for ZAC. We measure abstraction error by computing the percentage of requests having delay above λ and maximum delay obtained by any request which is above λ . We can observe that with HAC_MAX not only we can serve more requests but the error due to abstraction is also minimum. We also observe that as the zone size decreases, the number of requests served increases, error due to abstraction decreases with a slight increase in runtime. Based on these results, we use HAC_MAX as the clustering method for our next set of experiments.

6.5.2 Identification of Right Value of M

Our next set of experiments compare the service rate, runtime and abstraction error obtained using different values of M . Based on the observations made earlier, for

$M = 1$, we use HAC_MAX with zone size 120. For $M > 1$, the clustering method used is HAC_MAX and we run the experiments with different values of M . We use the zone sizes as 0, 60, 120, 300, 480, 600. The zone size of 0 means that the actual locations in the street network are used. Zone size of 60 means that the intra zone travel time is 60 seconds and so on. For $M = 2$, zone sizes used are 0 and 60, for $M = 4$ zone sizes used are 0, 60, 120 and 300 and for $M = 6$, zone sizes used are 0, 60, 120, 300, 480, 600.

We show the comparison of service rate and runtime with $M = 1$ (with zone size 120) and different values of M in Figure 6.10. HAC_MAX_120 is used to denote that $M = 1$ with zone size 120 is used. HAC_MAX_D_< m > denotes that value of M used is m . From the Figure 6.10 we can observe that we can serve more requests when $M > 1$, as compared to using fix large size zones. The abstraction error also reduces significantly by using $M > 1$. As the value of M is reduced, quality of solution improves with the increase in runtime. With $M = 2$ (for zone sizes 0 and 60), the abstraction error is almost 0 but runtime also increases. With $M = 4$, the abstraction error is less than 1%.

From these experiments on NYDataset, we obtain that by clustering locations into zones using HAC_MAX and using $M=4$ (with zone sizes 0,60,120,300), we get the right trade-off between computational complexity and solution quality. Therefore, we use this configuration for ZAC and ZACBenders. We now identify the right number of samples and lookahead duration for the ZACBenders algorithm.

6.5.3 Number of Samples

We compare the service rate and runtime by varying number of samples from 1 to 10 as shown in Figure 6.11a. We can observe from the figure, the service rate obtained by using a single sample is 74.6% and increases to 75.9% on using 5 samples. The service rate obtained by using 10 samples is 75.96% which is only 0.06% more than the service rate obtained by 5 samples. As the improvement beyond 5 samples is not much and comes at the cost of extra average runtime, we use 5 samples for the

ZACBenders algorithm.

6.5.4 LookAhead Duration

We compare the service rate by varying the look ahead duration from 10 minutes to 25 minutes for a single sample as shown in Figure 6.11b. We can observe from the figure that the service rate obtained by using look ahead duration of 10 minutes is 72.16% and increases to 74.6% on using look ahead of 15 minutes. The service rate obtained by using higher lookahead is less as with higher look ahead more requests are present at future decision epochs which increases the complexity of the problem and so the number of Benders decomposition iterations which can be completed within the maximum time limit reduces affecting the performance of ZACBenders. Moreover, due to the approximations used in the ZACBenders algorithm, it is not necessary that the performance will improve on using higher look ahead duration. Based on these experimental results, we choose a look ahead of 15 minutes for the ZACBenders algorithm.

Chapter 7

Background for Competitive Ratio

Analysis for M-OLYMPIAD

Problems

In this chapter, we provide the formal definition of expected competitive ratio and the research relevant (Dickerson et al., 2017) to the competitive ratio analysis for M-OLYMPIAD problems. The competitive ratio analysis works for a special case of M-OLYMPIAD problems where each server is assigned a fixed location and servers come back to their original location after serving requests, therefore, we define the model for these special cases and do not use the model proposed in chapter 2. We first describe the related research and then provide the background relevant to the competitive ratio analysis for the M-OLYMPIAD problems.

7.1 Related Work

There are two major threads of relevant research. The *first thread* is on online unit-capacity ridesharing where the underlying problem is an online bipartite matching problem. The standard online bipartite matching problem involves matching known (i.e., available offline) disposable servers ¹ on one side to the online arriv-

¹Once a server is assigned, it can not be used by any other incoming vertex/request.

ing vertices/requests on the other side, over multiple timesteps. Many approaches provide performance guarantees under different arrival assumptions for incoming vertices (Karp et al., 1990b; Devanur, Jain, & Kleinberg, 2013; Jaillet & Lu, 2013). Mehta (Mehta et al., 2013) provides a detailed survey of the same. One popular arrival assumption is the known identical independent distribution (KIID) (Jaillet & Lu, 2013; Manshadi et al., 2012), where online vertices arrive over T rounds and their arrival distributions are assumed to be identically distributed and independent over T rounds. This distribution is also known to the online algorithm in advance. The existing literature provide bounds of at least $1 - \frac{1}{e}$ on the expected competitive ratio (ratio of the expected value obtained by the algorithm to the expected value obtained by an offline optimal algorithm) for online bipartite matching problems under KIID.

In case of unit-capacity ridesharing, the offline available servers (i.e., vehicles) are reusable. Dickerson *et.al.* (Dickerson et al., 2017) were able to provide a $\frac{1}{2}$ bound for the unit-capacity ridesharing in which servers are reusable and they join the system after serving the requests at the same location. Instead of KIID, they consider that arrival distributions of online vertices can change from time to time (i.e., it is not iid) but this distribution is also known to the algorithm. They refer to this distribution as the Known Adversarial Distribution (KAD).

Unfortunately, this thread of work is only applicable for unit-capacity servers and cannot be directly adapted to consider multi-capacity servers because the underlying problem is no longer an online bipartite matching problem (see below). Another limitation is that the existing work for unit-capacity ridesharing has primarily focused on requests arriving sequentially (i.e., one by one) and not in batches which is a desirable property when considering multi-capacity ridesharing problems (for instance, last mile services at train stations need to consider that the large number of passengers will arrive and request for last mile transportation to their home at the same time.).

The *second thread* of relevant research is on approaches to solve online multi-

capacity (capacity > 1) ridesharing problems. There have been multiple heuristic approaches (Alonso-Mora, Samaranayake, et al., 2017; Lowalekar, Varakantham, & Jaillet, 2019) provided for solving the ridesharing problem for multi-capacity servers in batch arrival model. However, none of these approaches provide any bounds on the performance and are typically myopic (i.e., they do not consider any future information) due to the challenging nature of the problem.

The multi-capacity servers (capacity > 1) make the problem challenging because servers have to be matched to groups of requests and not just to individual requests. This results in a significant change in the structure of the underlying matching graph. Unlike unit-capacity ridesharing, where the underlying graph is bipartite, the multi-capacity ridesharing has a tripartite graph (Beineke, 1980) with reusable servers (vehicles), request groups (i.e., combinations of passenger requests) and on-line vertices (corresponding to passenger requests). The desired matching between the servers and request groups (combination of requests) is constrained by the edges between requests and request groups (i.e., a request can be part of at most one request group in final assignment) in this tripartite graph. It should be noted that this matching problem in tripartite graph is not equivalent to any variant of bipartite matching problem (Aggarwal et al., 2011; Feldman, Korula, Mirrokni, Muthukrishnan, & Pál, 2009; Lee & Singla, 2017; Z. Huang et al., 2018) studied in the literature. This is because the weight of a match and the time after which server becomes available again is dependent on the requests which are paired together in the group assigned to the server.

To the best of our knowledge, there has been no research on providing performance guaranteed algorithms for such tripartite graphs. There has been some work on solving a part of this matching problem which focused on finding the requests which can be grouped together over time by considering the sequential arrival of requests (Ashlagi et al., 2018, 2019) in the adversarial and random order arrival. However, these works ignore the main component of matching the servers to the request groups.

7.2 Background

In this section, we provide the formal definition of expected competitive ratio and the OM-RR-KAD model relevant (Dickerson et al., 2017) to the competitive ratio analysis presented in next chapter.

7.2.1 Competitive Ratio in expectation

The performance of any online algorithm is measured using a metric called competitive ratio. In case of known distribution models, we measure the competitive ratio in expectation. The competitive ratio in expectation of any algorithm ALG is defined (Mehta et al., 2013) as $\min_D \frac{E[ALG(I)]}{E[OPT(I)]}$, where I denotes the input and D denotes the arrival distribution and $E[OPT(I)]$ denotes the expected value of the offline optimal algorithm. In general, an upper bound on the value of $E[OPT(I)]$ is provided by using a benchmark linear program. This results in providing a valid lower bound on the resulting competitive ratio.

Since we only employ competitive ratio in expectation for the next chapters, we henceforth just refer to it as competitive ratio.

7.2.2 OM-RR-KAD

We now describe the Online Matching with (Offline) Reusable Servers/Resources under Known Adversarial Distributions (OM-RR-KAD) model (Dickerson et al., 2017) for OLYMPIAD problems in which the server capacity is restricted to 1. OM-RR-KAD is a bipartite matching problem between offline reusable servers (e.g., vehicles), \mathcal{U} , and vertices that arrive online, \mathcal{V} (e.g., user requests), over T rounds². Online vertices arrive according to a *Known Adversarial Distribution (KAD)* represented by a set of arrival probabilities, $\{p_v^t\}$ ($\sum_v p_v^t = 1, \forall t$). Once an online vertex of type v arrives (i.e., sampled from p_v^t), an irrevocable decision needs to be taken immediately to match it to one of the offline servers, for which a weight, $w_{u,v}^t$ is re-

²We use round and timestep interchangeably in the dissertation

LPSequential:

$$\max \sum_{t \in T} \sum_{u \in \mathcal{U}} \sum_{v \in \mathcal{V}} w_{u,v}^t \cdot x_{u,v}^t$$

$$s.t. \quad \sum_{u \in \mathcal{U}} x_{u,v}^t \leq p_v^t \quad \forall v, t \quad (7.1)$$

$$\sum_{t' < t} \sum_{v' \in \mathcal{V}} x_{u,v'}^{t'} \cdot Pr[c_{u,v'}^{t'} > t - t'] + \sum_{v \in \mathcal{V}} x_{u,v}^t \leq 1, \forall u, t \quad (7.2)$$

$$0 \leq x_{u,v}^t \leq 1 \quad \forall u, v, t \quad (7.3)$$

Table 7.1: Optimization Formulation - Unit-Capacity Sequential Arrival

ceived, or to reject it. The offline server becomes unavailable for a few rounds after it is matched and the number of rounds of unavailability, $c_{u,v}^t$, is characterized by an integral distribution, $c_{u,v}^t \in \{1, 2, \dots, T\}$. The offline server rejoins the system after $c_{u,v}^t$ rounds. The goal is to design an online assignment policy that will maximize the weight.

There are two key steps in obtaining a performance guaranteed online assignment policy:

First, an upper bound on the offline optimal, \mathbf{x}^* is computed using the LP of Table 7.1. $x_{u,v}^{*,t}$ denotes the probability of assigning server u to online vertex of type v in round t . Constraint (7.1) ensures that the expected number of times a vertex of type v is matched is less than or equal to the expected number of times the vertex is available. Constraint (7.2) ensures that the server u is assigned in round t if and only if it is available in round t . It should be noted that this LP provides a solution over all realizations of online vertices and hence that solution may not be applicable to a specific instantiation of online vertex (as the corresponding u may not be available). **Second**, an assignment rule is provided to compute the online probability of assigning a server u for a specific instantiation of online vertex (of type v in round t) and is given by:

$$\frac{x_{u,v}^{*,t} \cdot \gamma}{p_v^t \cdot \beta_u^t} \quad (7.4)$$

where $x_{u,v}^{*,t}$ is a solution to the LP in Table 7.1 and γ is the desired competitive ratio

of the online assignment; and β_u^t is the probability that server u is safe for assignment in round t . By simulating the current strategy up to t , β_u^t can be estimated with a small error.

The following theorem characterizes the $\frac{1}{2}$ bound on the expected competitive ratio.

Theorem 1. *Dickerson et.al. [(Dickerson et al., 2017)] The optimal value of LPSequential in Table 7.1 provides a valid upper bound on the offline optimal value for OM-RR-KAD. The online assignment rule of Equation 7.4 based on the LP achieves an online competitive ratio of $\frac{1}{2} - \epsilon$ for any given $\epsilon > 0$.*

The ϵ factor comes in the competitive ratio due to the error in the estimation of β_u^t . For a clean presentation, throughout the dissertation, we assume that these values can be estimated correctly and ignore the estimation error.

Chapter 8

Competitive Ratio Analysis for M-OLYMPIAD Problems

In this chapter we first design a performance guaranteed online algorithm that provides a competitive ratio of $\frac{1}{2}$ for the special case of U-OLYMPIAD problems (where servers/vehicles come back to their original location after serving requests) in which online vertices ¹ arrive in a batch under the known arrival distribution. Due to the change in the value obtained by optimal algorithm (more details in Section 8.1), it is not obvious whether the competitive ratio will increase or decrease or remain the same as compared to the sequential arrival case (Dickerson et al., 2017). Therefore, this is an important result where in we are able to show that the same competitive ratio can be achieved even when the vertices arrive in batches.

Next, we provide a performance guaranteed online algorithm that provides a non-zero competitive ratio for the M-OLYMPIAD problems considering batch arrival of online vertices under the known arrival distribution. The competitive ratio is:

- 0.31767 for capacity 2
- γ for any arbitrary capacity κ , where γ is solution to the the expression $(1 - \gamma)^{\kappa+1} = \gamma$.

¹The online arriving vertices correspond to the requests. Throughout the chapter we use vertices and requests interchangeably.

Even though we require groups of vertices in this online algorithm, these groups can be generated offline and hence does not add to the run-time complexity. These general bounds for arbitrary capacity M-OLYMPIAD problems are applicable under the assumption that the type of the servers/vehicles (i.e., their location) rejoining the system (after serving a group of vertices) does not change (Dickerson et al., 2017). Finally, we provide simple heuristics (based on the offline optimal LP) which work well in practice (as demonstrated in our experimental results).

8.1 Batch Arrival of vertices

In OLYMPIAD problems, user requests typically arrive in batches instead of arriving sequentially (e.g., users coming out of a train, theatre or mall looking for shared rides). So, we extend the OM-RR-KAD model to consider batch arrival of online vertices and also provide an online algorithm that achieves the same competitive ratio of $\frac{1}{2}$ as in the sequential arrival case. Batch arrival is different from sequential arrival because multiple online vertices (more information at each step) have to be matched to multiple offline servers at each round.

Since there are more vertices available in each round, online algorithms can potentially make better assignments in the batch case as compared to the sequential case. Due to this, it seems that the competitive ratio in the batch arrival case will be higher than the sequential arrival case. However,

- As the assignment for any vertex should be made in the same round of its arrival, in batch case where each round has multiple vertices, optimal (denominator of competitive ratio) also considers a greater number of vertices in each round and hence optimal can also improve (as compared to the optimal for sequential case).
- Compared to the sequential case, more time is spent deliberating (since we must wait until end of batch to make assignments) and during that time no assignment will happen and hence number of vertices assigned by optimal can be lower.

Therefore, the relationship between competitive ratio for the sequential and batch

cases is non trivial. We now provide an algorithm which ensures that the competitive ratio in batch arrival case is equal to the sequential arrival case.

We first mention the changes required in OM-RR-KAD model for the batch arrival case and then provide the performance guaranteed online algorithm for the unit-capacity case.

Changes to OM-RR-KAD for Batch Case: In the OM-RR-KAD model, at each round t , a single vertex is sampled using the probability $\{p_v^t\}$. However, in the batch extension, b^t vertices arrive at each round and each vertex of b^t is sampled using the same probabilities $\{p_v^t\}$. The expected number of vertices of type v arriving in round t is q_v^t and is given by:

$$q_v^t = b^t \cdot p_v^t$$

LP for Upper Bound on Offline Batch Optimal, LPBatch: The optimization formulation for the batch case is same as the LP in Table 7.1, except for the constraint in Equation (7.1). Given that there are q_v^t (and not p_v^t) expected arrivals of vertex of type v at each round, the modified constraint is:

$$\sum_{u \in \mathcal{U}} x_{u,v}^t \leq q_v^t \quad \forall v, t \quad (8.1)$$

We will refer to the modified LP as LPBatch.

Proposition 6. *The optimal value of LPBatch provides a valid upper bound on the offline optimal value ².*

ADAPBatch The online algorithm presented in Algorithm 8 is used to make an on-line assignment of the servers to the incoming vertices that are arriving in batches.

²Please refer to appendix for the proof.

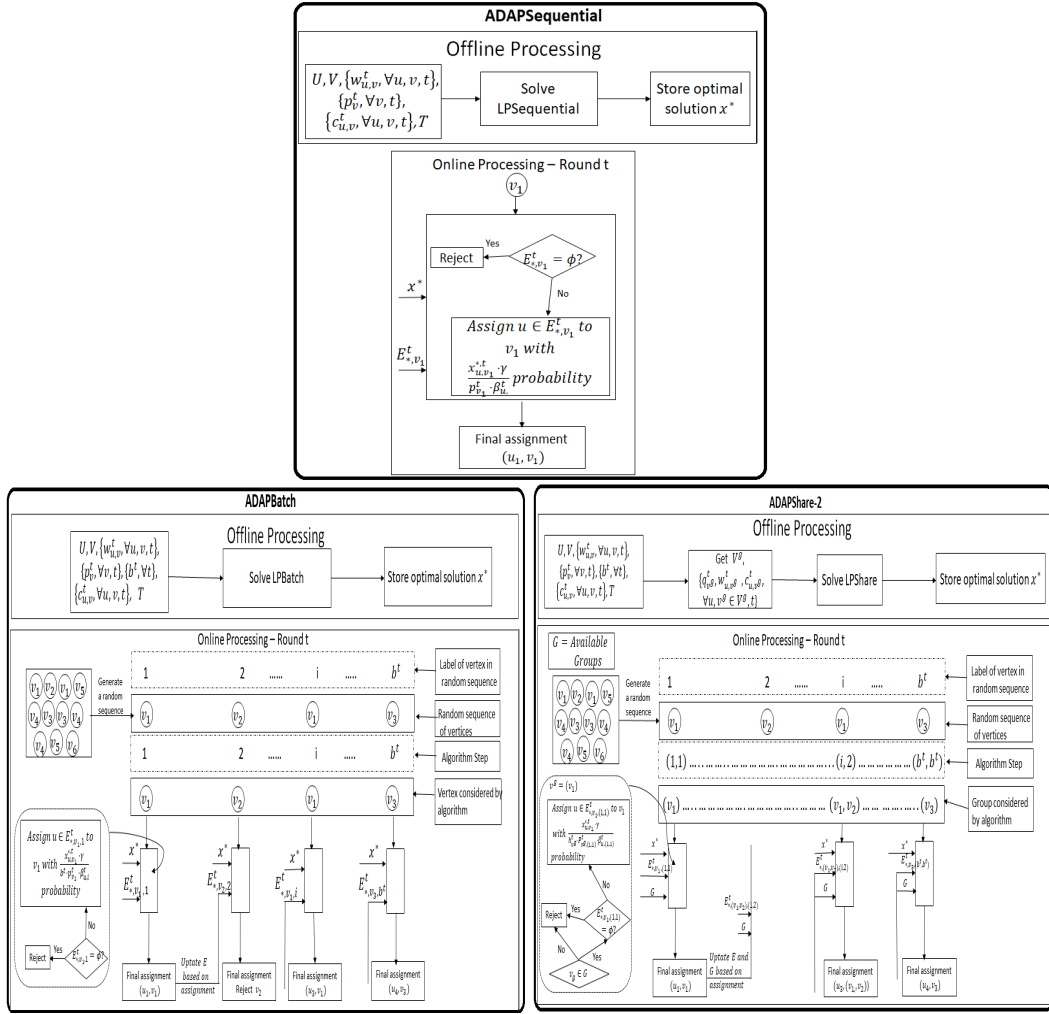


Figure 8.1: The Figure depicts the difference in the processing of algorithms in unit-capacity sequential, unit-capacity batch and multi-capacity share case. The online component in each of the algorithms corresponds to the processing in round t for a single instance of arrival of vertices. We only show the detailed flow diagram in the first block for each of the algorithms, rest of the blocks will have similar flow.

Algorithm 8 ADAPBatch(γ)

- 1: **for** $t < T$ **do**
 - 2: Generate a random shuffling of the incoming b^t vertices. Label the vertices from 1 to b^t .
 - 3: **for** $i = 1$ to b^t **do**
 - 4: $v =$ type of vertex with label i
 - 5: **If** $E_{*,v,i}^t = \phi$, then reject the vertex with label i ;
 - 6: **Else** choose $u \in E_{*,v,i}^t$ with probability $\frac{x_{u,v}^* \cdot \gamma}{q_{v,i}^t \cdot \beta_{u,i}^t}$
 - 7: Update the sets $E_{*,v,j}^t$ for all $j > i$ based on the assignment.
-

We use an adaptive algorithm ³ that employs the probability of a server being safe (available for assignment) while making assignments. The assignment rule to compute the online probability of assigning a u for the vertex of type v with label i in round t is:

$$\frac{x_{u,v}^{*,t} \cdot \gamma}{b^t \cdot p_v^t \cdot \beta_{u,i}^t}$$

where $\beta_{u,i}^t$ denotes the probability that server u is safe in round t when the vertex with label i is being considered; and $E_{*,v,i}^t \subset \mathcal{U}$ is used to denote the set of safe neighbors for a vertex of type v in round t when the vertex with label i is being considered.

In the algorithm, we process the vertices that have arrived in a batch one by one by considering a uniform random shuffling of incoming vertices. The intuition behind the assignment rule is to divide the optimal assignment for round t uniformly into b^t steps ($\frac{x_{u,v}^{*,t}}{b^t}$) and then to make sure that the vertex of type v is matched to server u at any step with probability $\frac{x_{u,v}^{*,t} \cdot \gamma}{b^t}$ unconditionally. Another key change in the algorithm from the sequential case is the last step where the availability of offline servers is updated based on assignments made in the same round. Figure 8.1 highlights the difference in the way the algorithms process online information in the sequential and batch case.

Proposition 7. *The online algorithm ADAPBatch is $\frac{1}{2}$ competitive.*

Proof Sketch: The maximum value of γ for which the algorithm ADAPBatch is valid ⁴ is $\gamma = \frac{1}{2}$. The proof involves showing that the minimum possible value of $\beta_{u,i}^t$ is $\frac{1}{2}$, for which we use mathematical induction. Finally, we show that ADAPBatch is γ competitive and since the maximum value of γ for which the assignment rule is valid is $\frac{1}{2}$, the algorithm is $\frac{1}{2}$ competitive. ■

³For an LP-based algorithm, we say that the algorithm is adaptive if for a given LP solution, the computation of strategy in each round t depends on the strategies in the previous rounds (Dickerson et al., 2019).

⁴Algorithm is valid when the assignment rule probability lies between 0 and 1.

8.2 Multi-capacity Reusable Servers

In this section, we provide a model, an online algorithm and competitive ratio analysis for the online multi-capacity OLYMPIAD problem with reusable servers.

8.2.1 Model: OPERA

To address the challenges associated with multi-capacity servers, we propose a new model called OPERA (**O**nline matching with offline multi-ca**P**acity r**E**usable **R**esources in b**A**tch Arrival Model). In OPERA, online vertices arrive in batches according to a *Known Adversarial Distribution (KAD)*. Once the online vertices arrive, there has to be an irrevocable decision made immediately on matching each offline server, u to a group of online vertices, v^g . The groups chosen for all servers should be such that each online vertex appears in at most one group. For each assignment of an offline server, u to a group of online vertices, of type v^g in round t , a weight, w_{u,v^g}^t is received. After the assignment, the offline server u is unavailable for c_{u,v^g}^t rounds before joining the system again⁵. The goal is to design an online assignment policy for assigning offline reusable servers to the groups of online vertices that will maximize the weight received over all time steps. Figure 8.2 shows the tripartite graph formed in the case of OPERA.

Unlike in OM-RR-KAD, the underlying problem in OPERA is no longer a bipartite matching problem but a matching in a tripartite graph containing offline servers, \mathcal{U} groups of online vertices, \mathcal{V}^g and online vertices, \mathcal{V} .

Here are other key differences between OPERA and OM-RR-KAD:

\mathcal{U} : Each offline server, $u \in \mathcal{U}$ in OPERA has a fixed capacity κ .

\mathcal{V}^g : As $\kappa > 1$, unlike in OM-RR-KAD model, servers can be assigned to more than one vertex at a round, i.e., servers can be assigned to groups of vertices where

⁵In the context of last mile ridesharing – after serving the group of passengers, server comes back to its initial location

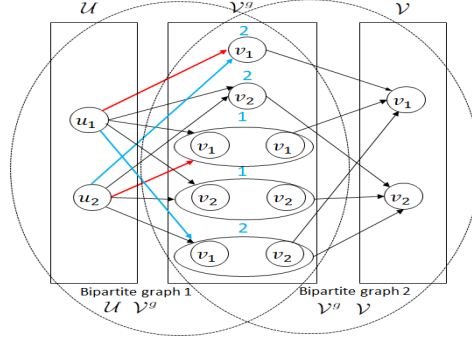


Figure 8.2: The Figure depicts the tripartite graph used in the OPERA model. It is a combination of 2 bipartite graphs. The goal is to find the matching in the first bipartite graph subject to the constraints enforced due to the edges present in the second bipartite graph. The blue numbers in \mathcal{V} indicate the number of vertices of each type available and blue numbers in \mathcal{V}^g denote the number of groups of each type which can be formed using available vertices in \mathcal{V} . The blue lines indicate a valid assignment of servers in \mathcal{U} to groups in \mathcal{V}^g . Red lines indicate an invalid assignment as the vertex of type v_1 is used 3 times in this assignment but there are only 2 vertices of type v_1 available.

group sizes vary from 1 to κ . For ease of analysis, we consider that all the vertices which can be paired together, and the constraints on the feasibility of pairing of vertices are handled through the weights received. Types of groups of vertices are obtained by generating all possible combinations (with repetitions) of size 1 to κ of the set \mathcal{V}^6 . The resulting set is denoted by \mathcal{V}^g . Therefore,

$$|\mathcal{V}^g| = \sum_{k=1}^{\kappa} \left(\binom{|\mathcal{V}|}{k} \right) = \sum_{k=1}^{\kappa} \binom{|\mathcal{V}| + k - 1}{k}$$

For each group of type v^g , n_{v,v^g} denotes the number of times vertex of type $v \in \mathcal{V}$ is present in group of type v^g (From the example Figure 8.2, for $v^g = (v_1, v_1)$, n_{v_1,v^g} will be 2 and for $v^g = (v_1, v_2)$, n_{v_1,v^g} will be 1.)

q_v^t : We consider batch arrival of vertices. Therefore, similar to the extension in Section 8.1, b^t vertices arrive at each round and each vertex of b^t is sampled using the same probabilities $\{p_v^t\}$. The expected number of vertices of type v

⁶ $\binom{n}{k}$ denotes the number of multisets of cardinality k , with elements taken from a finite set of cardinality n .

arriving in round t is q_v^t and is given by:

$$q_v^t = b^t \cdot p_v^t$$

w_{u,v^g}^t : Weight received is now based on the type of group assigned to the server.

c_{u,v^g}^t : Rounds of unavailability after an assignment is now based on the type of the group assigned to the server.

Apart from the model differences, there are also differences with respect to the online assignments that can be made. The irrevocable assignment of servers in \mathcal{U} to \mathcal{V}^g should satisfy the following constraints:

C1: Each server $u \in \mathcal{U}$ is assigned at most once in each round.

C2: The total number of vertices of each type $v \in \mathcal{V}$ used in the assigned groups is less than or equal to the number of vertices available.

C3: The number of groups of type $v^g \in \mathcal{V}^g$ assigned in round t is less than or equal to the number of available groups of type v^g .

In order to enforce constraint [C3] above in expectation (i.e., over all possible instantiations of arrivals), we need to compute $q_{v^g}^t$ — the expected number of times group of type v^g can be formed in round t . It is given by ⁷:

$$q_{v^g}^t = h_{v^g}^t \prod_{v \in v^g} (p_v^t)^{n_{v,v^g}} \text{ where } h_{v^g}^t = \frac{\prod_{i=0}^{i=|v^g|} (b^t - i)}{\prod_{v \in \mathcal{V}} (n_{v,v^g})!} \quad (8.2)$$

We make the following assumptions in the model: (1) Once a server u is assigned to a group of type v^g at t it becomes unavailable for further matches for c_{u,v^g}^t rounds irrespective of the size of v^g , i.e., insertion is not allowed. (2) The vertices can be grouped together iff they are arriving in same round. (3) For ease of explanation, we assume that $b^t > \kappa, \forall t$. However, this can be relaxed easily.

⁷It corresponds to drawing n_{v,v^g} vertices of each type $v \in v^g$ out of total b^t trials for a multinomial distribution. Please refer to appendix for details on deriving the expression.

LPSHare:

$$\max \sum_{t \in T} \sum_{u \in \mathcal{U}} \sum_{v^g \in \mathcal{V}^g} w_{u,v^g}^t \cdot x_{u,v^g}^t \quad (8.3)$$

$$\begin{aligned} \text{s.t. } & \sum_{t' < t} \sum_{v^{g'} \in \mathcal{V}^{g'}} x_{u,v^{g'}}^{t'} \cdot Pr[c_{u,v^{g'}}^{t'} > t - t'] + \\ & + \sum_{v^g \in \mathcal{V}^g} x_{u,v^g}^t \leq 1 \quad \forall u, t \end{aligned} \quad (8.4)$$

$$\sum_{v^g; v \in v^g} \sum_{u \in \mathcal{U}} n_{v,v^g} \cdot x_{u,v^g}^t \leq q_v^t \quad \forall v, t \quad (8.5)$$

$$\sum_{u \in \mathcal{U}} x_{u,v^g}^t \leq q_{v^g}^t \quad \forall v^g, t \quad (8.6)$$

$$0 \leq x_{u,v^g}^t \leq 1 \quad \forall u, v^g, t \quad (8.7)$$

Table 8.1: Optimization Formulation - Multi-capacity

8.2.2 Online Algorithm

We first provide an LP for computing the upper bound on the offline optimal and then provide an adaptive assignment method based on the offline optimal solution.

LP for Upper Bound on Offline Batch Optimal with Multi-Capacity Servers:

The optimization formulation ⁸ is provided in Table 8.1. We refer to this LP as LPSHare. Since LP is for the offline case over all possible instantiations on arrival vertices, the constraints hold in expectation. Constraints (8.4), (8.5) and (8.6) refer respectively to **C1**, **C2** and **C3** constraints (described in Section 8.2.1) in expectation (i.e., over all possible instantiations of arrivals). Constraint (8.4) ensures that the server u is assigned in round t iff u is available in round t .

Proposition 8. *The optimal value of LPSHare provides a valid upper bound on the offline optimal value ⁹.*

ADAPShare- κ : For ease of explanation, we first present the online algorithm and competitive analysis for $\kappa = 2$.

Let $x_{u,v^g}^{*,t}$ denotes the optimal probability of assigning u to a group of type v^g in round t (computed from offline optimal LP). We use Algorithm 9 to make online

⁸LP is based on satisfying the flow constraints in the graph shown in Figure 8.2.

⁹Please refer to appendix for the proof.

Algorithm 9 ADAPShare-2(γ)

- 1: **for** $t < T$ **do**
 - 2: Generate a random shuffling of the incoming b^t vertices. Label the vertices from 1 to b^t .
 - 3: **for** $i = 1$ to b^t **do**
 - 4: **for** $j = 1$ to b^t **do**
 - 5: $v^g =$ type of group formed at step (i, j) based on the labels assigned to the vertices.
 - 6: **if** v^g is available for assignment at step (i, j) **then**
 - 7: **if** $E_{*,v^g,(i,j)}^t == \phi$, reject v^g
 - 8: **else choose** $(u, v^g) \in E_{*,v^g,(i,j)}^t$ with probability p where $p =$
 - 9:
$$\frac{x_{u,v^g}^{*,t} \cdot \gamma}{h_{v^g}^t \cdot P_{v^g,(i,j)}^t \cdot \beta_{u,(i,j)}^t}$$
 - 9: Update $E_{*,*,(i,j)}^t$, available groups based on the assignment.
-

assignment of servers to the groups of vertices based on $\{x_{u,v^g}^{*,t}\}$ values from the offline optimal LP. As shown in the algorithm, we perform a random shuffling of the b^t vertices (that arrive in a batch in round t) and label the vertices from 1 to b^t . The assignment of servers to groups is performed across $b^t \cdot b^t$ steps (as we consider groups of size 2). Step (i, j) corresponds to a step where we compute the probability for assignment of a group formed by vertices with labels i and j . It should be noted that when $i = j$, (i, j) corresponds to a group of size 1 with only vertex with label i .

The assignment rule to compute the online assignment probability of assigning server u to a group of type v^g at step (i, j) of the algorithm is defined by

$$\frac{x_{u,v^g}^{*,t} \cdot \gamma}{h_{v^g}^t \cdot P_{v^g,(i,j)}^t \cdot \beta_{u,(i,j)}^t} \quad (8.8)$$

where $\beta_{u,(i,j)}^t$ denotes the probability that server u is available for assignment in round t at step (i, j) over all arrival sequences. Similarly $P_{v^g,(i,j)}^t$ denotes the probability that group of type v^g can be considered for assignment in round t at step (i, j) over all arrival sequences. $h_{v^g}^t$ was defined in Equation (8.2). We use $E_{*,v^g,(i,j)}^t \subset \mathcal{U}$ to denote the set of safe servers for group of type v^g at step (i, j) .

Similarities and Differences to ADAPBatch: The intuition behind the assignment

rule for a step is similar to the one in ADAPBatch. Assignment for a group of type v^g in a step is obtained by dividing the optimal assignment of round t for group of type v^g by the total number of steps where group of type v^g can be considered ¹⁰.

The key differences in assignment rule of ADAPShare- κ and ADAPBatch:

- For $\kappa = 2$, since we can consider 2 vertices together (in a group) for assignment, we process the groups in $b^t \cdot b^t$ steps for ADAPShare-2. This is in comparison to b^t steps in ADAPBatch.
- In ADAPBatch, during online processing, vertex with label i in the batch will be considered for assignment only at one of b^t steps. In ADAPShare- κ , a vertex is part of multiple groups, so it will be considered at multiple steps. Therefore, at each step, the probability of vertex being available (and as a result a group being available) needs to be recomputed based on the groups assigned at previous steps in the same round.

Figure 8.1 highlights the difference in the way the algorithms ADAPBatch and ADAPShare- κ process the online information.

Competitive Ratio for ADAPShare-2

In this section, we provide the analysis to compute the competitive ratio for ADAPShare-2. We first find the value of γ for which the assignment rule in Equation (8.8) is valid, i.e., it corresponds to a valid probability value between 0 and 1.

Proposition 9. *The maximum value of γ for which assignment rule in Equation*

¹⁰Each group of type v^g will be considered at $h_{v^g}^t$ steps out of the total $b^t \cdot b^t$ steps. For $\kappa = 2$, from Equation (8.2)

$$h_{v^g}^t = \begin{cases} b^t, & \text{if } |v^g| = 1, \\ b^t \cdot (b^t - 1) & \text{if } |v^g| = 2 \text{ and } v^g = (v, v'), \\ \frac{b^t \cdot (b^t - 1)}{2} & \text{if } |v^g| = 2 \text{ and } v^g = (v, v). \end{cases}$$

This is because when both vertices are of same type in the group, for example if $v^g = (v, v)$, then v^g considered at step (i, j) means that the vertex with label i and the vertex with label j both are v and therefore steps (i, j) and (j, i) would be identical. On the other hand when both vertices are of different type, for example if $v^g = (v, v')$, then v^g considered at step (i, j) means that the vertex with label i is v and the vertex with label j is v' but v^g considered at step (j, i) means the opposite. Hence in this case the group of type v^g will be considered at $b^t \cdot (b^t - 1)$ steps across different online arrivals. Please refer to the example in the appendix for more clarity.

(8.8) is valid is 0.31767.

Proof: Since the assignment rule always generates a positive value, the condition to be satisfied for the assignment rule to be valid is

$$\frac{x_{u,v^g}^{*,t} \cdot \gamma}{h_{v^g}^t \cdot P_{v^g,(i,j)}^t \cdot \beta_{u,(i,j)}^t} \leq 1 \quad (8.9)$$

Using Equation (8.2) in Constraint (8.6) of optimization formulation in Table 8.1, we have

$$\sum_u x_{u,v^g}^{*,t} \leq h_{v^g}^t \cdot \prod_{v \in v^g} (p_v^t)^{n_{v,v^g}} \implies x_{u,v^g}^{*,t} \leq h_{v^g}^t \cdot \prod_{v \in v^g} (p_v^t)^{n_{v,v^g}}$$

Substituting this in Equation (8.9) and rearranging terms, we get

$$\beta_{u,(i,j)}^t \geq \frac{\gamma \cdot \prod_{v \in v^g} (p_v^t)^{n_{v,v^g}}}{P_{v^g,(i,j)}^t} \quad \forall t, i, j, v^g \quad (8.10)$$

By considering the probabilities with which each of the vertex of type $v \in v^g$ is available at step (i, j) , we can show that ¹¹,

$$\frac{\prod_{v \in v^g} (p_v^t)^{n_{v,v^g}}}{P_{v^g,(i,j)}^t} \leq \frac{1}{(1-\gamma)^2}, \quad \forall t, i, j, v^g \quad (8.11)$$

Using Equations (8.10) and (8.11), for the assignment rule to be valid it is sufficient to show that $\beta_{u,(i,j)}^t \geq \frac{\gamma}{(1-\gamma)^2}$.

We can compute a lower bound on the value of $\beta_{u,(i,j)}^t$ based on assignments performed in previous steps and rounds. Specifically, using mathematical induction, we can show that $\beta_{u,(i,j)}^t \geq 1 - \gamma$.

So, to find the maximum value of γ for which the assignment rule is valid, we take γ such that $1 - \gamma = \frac{\gamma}{(1-\gamma)^2}$. Therefore, the possible value of γ is the solution to the equation $\gamma = (1 - \gamma)^3$, which is $\gamma = 0.31767$.

¹¹Please refer to appendix for the detailed proof.

Proposition 10. *The online algorithm ADAPShare-2 is 0.31767 competitive.*

Proof: The proof involves first showing that the ADAPShare-2 is γ competitive. Now, as from Proposition 9, the maximum value of γ for which assignment rule is valid is 0.31767, therefore the algorithm is 0.31767 competitive.

To show that the ADAPShare-2 is γ competitive, we compute with respect to the optimal, the fraction of times any server u is assigned to any group of type v^g . The probability that the server u is assigned to a group of type v^g in round t in step (i, j) is given by

$$\frac{x_{u,v^g}^{*,t} \cdot \gamma}{h_{v^g}^t \cdot P_{v^g,(i,j)}^t \cdot \beta_{u,(i,j)}^t} \cdot \beta_{u,(i,j)}^t \cdot P_{v^g,(i,j)}^t = \frac{x_{u,v^g}^{*,t} \cdot \gamma}{h_{v^g}^t}$$

where first term in the product is the assignment rule, second term is the probability that u is available and the last term is the probability that v^g is available in round t at step (i, j) .

As mentioned before, each group of type v^g will be considered for assignment at a total of $h_{v^g}^t$ steps. Therefore, the expected number of times a server u is assigned to a group of type v^g in round t is given by $h_{v^g}^t \cdot \frac{x_{u,v^g}^{*,t} \cdot \gamma}{h_{v^g}^t} = x_{u,v^g}^{*,t} \cdot \gamma$, i.e., in online case each server u is matched to group of type v^g with probability equal to $x_{u,v^g}^{*,t} \cdot \gamma$.

Therefore, ADAPShare-2 is γ competitive. ■

Corollary 1. *The online algorithm ADAPShare- κ (generalization of ADAPShare-2 for any value of κ) is γ competitive where the value of γ is the solution to the equation $\gamma = (1 - \gamma)^{\kappa+1}$.*

Proof Sketch: The proof is along the same lines as the proof for Proposition 10. In the Equation (8.11), instead of $(1 - \gamma)^2$, we will have $(1 - \gamma)^\kappa$. Therefore, the value of γ for which assignment rule is valid is the solution to the Equation $\gamma = (1 - \gamma)^{\kappa+1}$.

■

Hardness Results: Dickerson *et.al.* (Dickerson et al., 2017) prove that no non-adaptive algorithm based on LPSequential can achieve a competitive ratio of more

than $\frac{1}{2} + o(1)$ in OM-RR-KAD model. The analysis can be easily extended for the batch arrival case when $\kappa = 1$. As unit-capacity batch arrival is a special case of multi-capacity OPERA model with all $w_{u,v^g}^t = 0$, if $|v^g| \geq 2$, therefore, no non-adaptive algorithm based on LPShare can achieve a competitive ratio of more than $\frac{1}{2} + o(1)$ for OPERA model. For a general case of adaptive algorithms the upper bound of 0.823 provided by Manshadi et.al. (Manshadi et al., 2012) for unit capacity disposable resources will be applicable. This is because OM-RR-KAD model generalizes the setting proposed in Manshadi et.al. (Manshadi et al., 2012) and multi-capacity OPERA model generalizes the setting in OM-RR-KAD model.

Discussion: We now provide the justifications for the choices made in the modelling and analysis in section 8.1 and 8.2.1. (1) We assume that there are b^t arrivals in round t and b^t is known in advance. However, this is not at all a strong assumption because by considering a null type vertex in \mathcal{V} and p_ϕ^t as the probability of null vertex, b^t can be used to denote the maximum number of arrivals in round t . (2) For theoretical analysis of the solution quality, we ignore the computational complexity of generating exponential number of groups in OPERA model. For practical purposes, the algorithms provided in (Alonso-Mora, Samaranayake, et al., 2017) can be used to heuristically prune the exponential set and generate the feasible groups efficiently. The pruned set of groups is used by both offline and online algorithms. This is because, if the offline optimal algorithm can generate the groups, as the type of vertices are known in advance (through the known distribution), the online algorithm can also use those groups.

8.3 Experiments

In this section, we compare the following five approaches on the empirical competitive ratio metric:

- **Greedy** - Runs an integer optimization at each timestep (based on the current information) to assign the incoming requests/vertices to the available offline servers ¹².
- **Random** - Shuffles incoming requests/vertices in the current batch randomly and then assigns each request/vertex randomly to an available offline server.
- **Alg-OPERA-1** - Algorithm based on the offline optimal LP where match for any available server u to a vertex or group is performed by looking at the value of $\frac{x_{u,vg}^{*,t}}{q_{vg}^t}$ ¹³.
- **Alg-OPERA-2** - Another algorithm based on the offline optimal LP where match for any available server u to a vertex or a group is performed by looking at the value of $\frac{x_{u,vg}^{*,t}}{\sum_u x_{u,vg}^{*,t}}$.
- **ϵ -Greedy** - With probability ϵ , greedy algorithm is executed and with probability $1 - \epsilon$, Alg-OPERA-1 algorithm is executed.

The goal of the experiments is to show that the algorithms which use guidance from the offline optimal LP, outperform the myopic approaches ¹⁴, which do not consider future information. All the values in the results are computed by taking an average over 10 instances and each instance is run 100 times.

Synthetic Dataset: We first present the results on a synthetic dataset. We use 200 timesteps/rounds and generate the unavailability (or time occupied serving requests) time ($c_{u,v}^t$ or $c_{u,vg}^t$) for each server and vertex/group pair randomly between 1 and 60. Weights received (revenue) are generated based on revenue model used by taxi companies – base revenue + $0.5 \cdot c_{u,v}^t$ or $c_{u,vg}^t$. The probability of arrival of each vertex type at each round (p_v^t) is also generated randomly. The test instances are generated by sampling the online vertices from the generated p_v^t values. We vary

¹²Equivalent to the myopic approaches used in practice (Alonso-Mora, Samaranayake, et al., 2017; Lowalekar et al., 2019)

¹³We provide heuristics, which are close to ADAPShare- κ , as computing β exactly is not always simple and may require large number of simulations. We observed that even though these heuristics are non-adaptive, they can achieve empirical competitive ratio higher than the theoretical competitive ratio of ADAPShare- κ .

¹⁴Currently used in practice for multi-capacity servers (Alonso-Mora, Samaranayake, et al., 2017; Lowalekar et al., 2019)

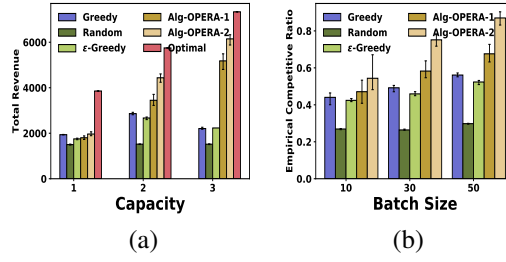


Figure 8.3: $|\mathcal{U}| = 10$, $|\mathcal{V}| = 10$, $T = 200$ (a) Varying κ (b) $\kappa = 2$

the batch size and capacity and present the representative results.

Figure 8.3a shows the total revenue obtained by different algorithms for different values of capacity κ . The key observations are:

(1) Our online approaches (Alg-OPERA-1 and Alg-OPERA-2) outperform other algorithms, with Alg-OPERA-2 performing better than Alg-OPERA-1 on all the instances.

(2) The performance of greedy algorithm decreases with the increase in capacity. Higher capacity provides more opportunity to serve requests at each timestep. Due to its myopic nature, greedy algorithm serves more requests initially, keeping the servers occupied for a longer time. On the other hand Alg-OPERA-1 and Alg-OPERA-2, based on the guidance provided by the offline optimal LP, ignore some requests/groups which have higher $c_{u,v}^t$ value, to serve more requests at future timesteps.

(3) Figure 8.3b shows the empirical value of competitive ratio for different batch sizes. For these experiments, we take the identical value of batch size for all the timesteps. Higher batch size for multi-capacity servers provides an opportunity to group more requests. Therefore, as the batch size increases Alg-OPERA-1 and Alg-OPERA-2 show an improvement in performance.

Real World Dataset: We used the New York Yellow Taxi dataset which contains the records of trips in Manhattan city. We divided the map of the city into a grid of squares, each 4 by 4 km, which resulted in a total of 11 squares. Therefore, there can be 121 different types of requests, i.e., $|\mathcal{V}| = 121$ (origin-destination

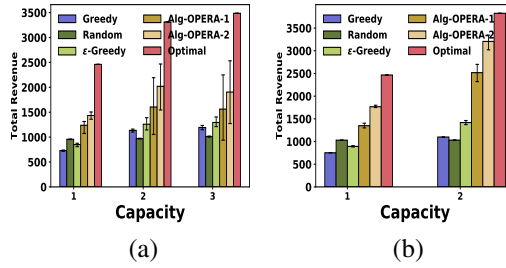


Figure 8.4: $|\mathcal{U}| = 30$, $|\mathcal{V}| = 121$, $T = 240$, Real Dataset (a) 12am (b) 8am

pairs). We experimented by taking real trips from the taxi dataset. We take the data across 10 days to compute the p_v^t values and the average number of requests at each round/timestep, i.e., average value of b^t . We run the offline optimal LP with these values and get a solution. The online algorithms are tested on actual instances (10 days) which are different from the ones we used for computing the parameter values. Therefore, the actual batch size b^t can be different from the value used by an offline optimal solution. The taxis are initialized at random locations and since we are testing the last mile scenario after serving the trips, they come back to their starting location. We observe a high variance in the performance of our algorithms on this dataset during night time (Figure 8.4a). This is because the distribution of requests during night have high variance across days. During the day, the variance in distribution of requests is low, and as a result our algorithms also show low variance. On an average, Alg-OPERA-1 and Alg-OPERA-2 outperform other algorithms on this dataset as well. These results indicate that the algorithms which use the guidance from offline optimal solution can consider the future effects of current matches and as a result provide better performance.

We would like to highlight that, to ensure that the theoretical bound on the competitive ratio holds empirically, correct estimates of probability values (p_v^t, β) are required, which requires running multiple simulations. It is possible to create scenarios, where a high number of simulations are required to get the correct estimates (e.g., when all the p_v^t values are very small and \mathcal{V} is large.). In such cases, empirical competitive ratio measured over low number of simulations, will be a wrong

indicator. We would also like to mention that, it is possible to synthetically create unrealistic scenarios where Greedy algorithm can achieve close to optimal value (essentially having a revenue model such that the difference between one long trip and multiple short trips is almost negligible, so myopic decisions do not hurt) and can perform better than the LP based approaches.

Chapter 9

Background for Decision Making in OLYMPIOD Problems

This chapter explains the model and related research for decision making in OLYMPIOD problems. It also explains the details of the simulation model used for obtaining experimental results presented in chapter 10.

9.1 Model

In this section, I provide a generic model for representing the OLYMPIOD problem while considering future demand samples. We divide the time into discrete timesteps of duration Δ minutes and the future demand samples are considered for the next Q timesteps. In OLYMPIOD, resources are transported between pick-up and drop-off locations using a set of carrier vehicles/servers. Each request element has a single customer location which can serve as a supply location allowing pick-up of resources or can serve as a demand location where resources need to be delivered to meet the required demand. In addition to the customer locations from where the resources can be picked up or delivered, there are stations or warehouses which keep the resources and carrier vehicles/servers are used to redistribute resources across these stations/warehouses. Along with the explicit customer requests for pick-up or drop-off of resources at the customer location (referred to as remote customer

requests here onwards), these stations/warehouses will also have walk-in customers who can purchase/rent these resources. The resources can be consumable resources like food items which can not be reused or rental resources like bikes, car, equipments, clothes which will be returned after usage. The walk-in customers can rent resources from one station/warehouse and can return the resources at any other station/warehouse. Remote customers will return the rented resources by requesting pick-up at their location. While the resources which need to be transferred using carrier vehicles can be of a single type or can have multiple types, in this dissertation, I consider only single type of resources.

OLYMPIOD is formally defined using the following tuple:

$$\langle \mathcal{B}, \mathcal{V}, \mathcal{D}, J, C, d, \sigma, \mathcal{X}, \mathcal{R}, \tau, \pi \rangle$$

- \mathcal{B} denotes the set of all locations. \mathcal{B} can be represented as $\mathcal{B} = \mathcal{S} \cup \mathcal{L}$ where \mathcal{S} is the set of stations/warehouses/shops and \mathcal{L} denotes the set of customer locations.
- \mathcal{V} denotes the set of servers/carrier vehicles used for pick-up and drop-off of resources.
- \mathcal{D} denotes the set of remote customer requests at the current timestep. \mathcal{D}^+ denotes the set of remote customers requesting pick-up of resources with $D_l^{+,n}$ denoting the number of resources to be picked up from location l . $\mathcal{D}^{+,c}$ denotes the cost of ignoring the pick-up request at location l . Similarly, \mathcal{D}^- denotes the set of remote customers requesting drop-off of resources with $D_l^{-,n}$ denoting the number of resources required at customer location l . $\mathcal{D}^{-,c}$ denotes the cost of ignoring the drop-off request at location l ¹.
- \mathcal{J} denotes the set of customer requests for the future timesteps. \mathcal{J} can be represented as $\mathcal{J} = \mathcal{F} \cup \mathcal{G}$, where F denotes samples of walk-in customer requests

¹Customer Requests are kept in this set until they are served by a server. The cost of ignoring requests which were assigned to a server in previous decision epochs is higher than the cost of ignoring the new requests.

and G denotes the set of remote customer requests for the future timesteps. $F^{t,k}$ denotes the set of customer requests in demand sample k at decision epoch t . We use $F_{s,s'}^{t,t',k}$ to denote the number of resources which will be rented at station s timestep t and returned at station s' at timestep t' in sample k ². $F_s^{t,k}$ denotes the total number of resources rented at station s , decision epoch t in sample k . $G_l^{-,t,k,n}$ is used to denote the total number of resources required at location l at timestep t in sample k . Similarly $G_l^{+,t,k,n}$ is the total number of resources which needs to be picked up from location l at timestep t in sample k . $G_l^{-,t,k,c}$ is used to denote the cost of ignoring the dropoff request at location l at timestep t in sample k . Similarly $G_l^{+,t,k,c}$ is used to denote the cost of ignoring the pick up request at location l at timestep t in sample k .

- C denotes the capacity of stations and servers with C_s denoting the capacity of station s . Similarly C_v denotes the capacity of server v .
- d denotes the initial distribution of resources at the stations and in the servers. d_s denotes the initial number of resources at the station s and d_v denotes the initial number of resources in the server v .
- σ denotes the distribution of carrier vehicles/servers at stations and locations. $\sigma_v^t(b)$ is set to 1, if the server v starts from the station or customer location b ($b \in \mathcal{B}$) at decision epoch t and is 0 otherwise.
- $\mathcal{X}_s^{t,k}$ denotes the additional number of resources which will become available at the station s at decision epoch t in sample k ³.
- \mathcal{R} denotes the objective function to be minimized, it can be the lost demand, travel cost for the servers etc. In this dissertation, I use the objective of minimizing lost demand with R_b^t denotes the lost demand at station/location b at timestep t .
- τ denotes the maximum allowed time for fulfilling remote customer requests

²To consider the case of consumable resources, we use $F_{s,\phi}^{t,\phi,k}$ to denote the number of resources which will be rented at stations s at timestep t and will not be returned.

³This could be due to the resources rented in previous decision epochs which will be returned at station s at decision epoch t

with τ_l^+ denoting the timestep by which remote customer request for pick-up of resources at location l should be fulfilled. Similarly, τ_l^- denotes the timestep by which remote customer request for drop-off of resources at location l should be fulfilled.

- π denotes the maximum allowed time for fulfilling samples of remote customer requests with $\pi_l^{+,t,k}$ denoting the timestep by which remote customer request for pick-up of resources at location l at timestep t in sample k should be fulfilled. Similarly, $\pi_l^{-,t,k}$ denotes the timestep by which remote customer request for drop-off of resources at location l at timestep t in sample k should be fulfilled.

The goal in OLYMPIOD is to minimize the expected value of objective \mathcal{R} over multiple samples of demand scenarios over the entire time horizon $Q \cdot \Delta$. It should be noted that all static elements of the OLYMPIOD tuple except \mathcal{D} can be populated directly from the datasets. \mathcal{D} are the online elements representing incoming demand. We assume that if a remote customer requests for the rental resources, the rental period will be long and resources will not be returned within Q timesteps.

For applications such as bikesharing which only involve walk-in customers and do not have any remote customer requests, the elements \mathcal{D} , \mathcal{G} , τ and π are empty.

9.2 Related Work

The OLYMPIOD problems (Dror, Fortin, & Roucairol, 1998) did not receive much attention before the rise of bikesharing systems. Most of the earlier work on these problems is in the offline setting with a single server (Hernández-Pérez & Salazar-González, 2004, 2007). Dror *et al.* (Dror et al., 1998) present mixed integer programming approach, a constrained program and local search heuristic method which are used to solve instances involving to nine locations. Gunes *et.al.* (Gunes, van Hove, & Tayur, 2010) solve the problem for the food rescue program domain where excess food is collected from restaurants and redistributed through agencies to people in need. They also provide a mixed integer program and a constraint program-

ming formulation in offline setting. Similar to food rescue program domain, bike-sharing systems also involve redistribution of bikes between different locations and can be formulated as an OLYMPIOD problem. There has been extensive research on static repositioning approaches (Schuijbroek et al., 2017; Chemla et al., 2013; Raviv et al., 2013) for bike sharing systems. These approaches perform repositioning only at the beginning of the day, when the movements of bikes by the customers are negligible, to ensure desired distribution of bikes across different stations. But they do not consider stations getting imbalanced during the day. Hence there is a need to focus on dynamic repositioning of bikes (Contardo, Morency, & Rousseau, 2012; Ghosh et al., 2015, 2017) during the day.

Shu, Chou, Liu, Teo, and Wang (2013) provide an optimization model for dynamic repositioning of bikes, but they do not consider routing of carrier vehicles /servers. Ghosh et al. (2015, 2017) consider the problem of dynamic repositioning of bikes along with the problem of finding the routing policy for carrier vehicles/servers. They provide an offline policy generation approach based on mean demand computed from the historical data. To overcome the inherent complexity of the problem, they propose a decomposition and abstraction based approach. While the offline policy provides significant improvement over static repositioning approaches, it is unable to consider the changing demand scenarios in real-time. Recently Ghosh and Varakantham (2017) propose a novel dynamic repositioning approach by combining optimization and mechanism design, to reduce carbon footprint using bike trailers. Ghosh, Trick, and Varakantham (2016) provide an online approach to compute dynamic repositioning and routing policy. The focus of their work is on providing a robust approach which optimizes for the worst case scenario. While robust policies provide guarantees for the adversarial inputs, they do not work well when demand follows an expected distribution. In this dissertation, we provide an online approach to compute dynamic repositioning and routing policy using demand samples from historical data. Due to its online nature, this approach can adapt and react to the changing demand patterns in real-time.

9.3 Simulation Model

To evaluate the approaches for solving OLYMPIOD problems, we employ the simulation model similar to the one proposed in (Ghosh et al., 2016). The walk-in customer requests are served every minute in the simulation from the resources available in the stations/warehouses. Remote customer requests are aggregated for Δ duration and the OLYMPIOD solution is computed every Δ minutes, i.e., the resources are picked up and dropped off from the stations and customer locations using carrier vehicles/servers every Δ minutes. Let t_{sim} denotes the simulation time in minutes and $d_s^{\#,t_{sim}}$ denotes the number of resources at station s at t_{sim} minutes. $f_{s,s'}^{t,t'}$ denotes the number of arrival customers at station s at timestep t who want to reach station s' at timestep t' .

We assume the following sequence of events at any station s :

1. Return of resources by walk-in customers.
2. Pick-up/drop-off of resources by carrier vehicles/servers.
3. Renting of resources by walk-in customers.

If the number of resources available at station are less than the number of resources required by walk-in customers, the actual flow of walk-in customers is computed as follows:

$$x_{s,s'}^{t_{sim},t'} = \begin{cases} f_{s,s'}^{t_{sim},t'} & \text{if } \sum_{t',s'} f_{s,s'}^{t_{sim},t'} \leq d_s^{\#,t_{sim}} \\ \frac{f_{s,s'}^{t_{sim},t'}}{\sum_{t',s'} f_{s,s'}^{t_{sim},t'}} \cdot d_s^{\#,t_{sim}} & \text{Otherwise} \end{cases}$$

On the other hand at any customer location l , if the customer request can be fulfilled based on the server status (resources in the server), the server drops or picks up resources at the location depending on the customer request.

1. if $t_{sim} \% \Delta == 0$, availability of resources at the stations is updated as follows:

$$d_s^{\#,t_{sim}+1} = d_s^{\#,t_{sim}} + \left[\sum_{t'} \sum_{s'} x_{s',s}^{t_{sim}-t',t_{sim}} \right]$$

$$d_s^{\#,t_{sim}+1} = d_s^{\#,t_{sim}+1} + \left[Y_s^{-,t_{sim}} - Y_s^{+,t_{sim}} \right]$$

$$d_s^{\#,t_{sim}+1} = d_s^{\#,t_{sim}+1} - \sum_{t'} \sum_{s'} x_{s,s}^{t_{sim},t_{sim}+t'}$$

2. if $t_{sim} \% \Delta \neq 0$, availability of resources is updated as follows:

$$d_s^{\#,t_{sim}+1} = d_s^{\#,t_{sim}} + \left[\sum_{t'} \sum_{s'} x_{s',s}^{t_{sim}-t',t_{sim}} \right]$$

$$d_s^{\#,t_{sim}+1} = d_s^{\#,t_{sim}+1} - \sum_{t'} \sum_{s'} x_{s,s}^{t_{sim},t_{sim}+t'}$$

where $Y_s^{-,t_{sim}}$ and $Y_s^{+,t_{sim}}$ denotes the total number of resources picked up/dropped off by servers/carrier vehicles at station s at time t_{sim} . These values are obtained by solving the OLYMPIOD problem at t_{sim} .

In our experimental results, we compare against the Static Repositioning approach which does not perform any repositioning during the planning period. Therefore, for static repositioning approach $Y_s^{-,t_{sim}}$ and $Y_s^{+,t_{sim}}$ will be 0.

Chapter 10

Optimization Approaches to Solve OLYMPIOD Problems

To solve the OLYMPIOD model described in the previous chapter, we need to find a routing strategy which decides the path to be taken by carrier vehicles/servers and a repositioning strategy which decides the number of resources to be picked up or dropped off at each location at each timestep. In this chapter, we provide a multi-period two-stage stochastic optimization formulation and a Lagrangian dual decomposition approach to compute these repositioning and routing decisions. We also provide a greedy online anticipatory heuristic approach to solve large scale problems effectively and efficiently for special cases when there are no remote customer requests.

10.1 Multi-Period Two-Stage Stochastic Optimization

We now formulate OLYMPIOD as a mixed integer linear optimization formulation. Unlike most current online repositioning approaches that are typically myopic (Pfrommer, Warrington, Schildbach, & Morari, 2014), this multi-period two-stage stochastic (MSS) formulation considers future demand scenarios while opti-

Variable	Description
$y_{b,v}^{+,0}$	Number of resources picked up from the station/location b by server v at current decision epoch.
$y_{b,v}^{-,0}$	Number of resources dropped at the station/location b by server v at current decision epoch.
$y_{b,v}^{+,t,k}$	Number of resources picked up from the station/location b by server v at decision epoch t in sample k
$y_{b,v}^{-,t,k}$	Number of resources dropped at the station/location b by server v at decision epoch t in sample k
$h_s^{t,k}$	Resources rented in sample k at station s at decision epoch t
$g_s^{t,k}$	Resources returned in sample k at station s at decision epoch t
R_b	Lost demand at the location b at current decision epoch
$R_b^{t,k}$	Lost demand in the sample k at location b at epoch t
$d_s^{t,k}$	Number of resources present at the station s at epoch t in sample k
$x_l^{+,t'}$	Indicates that the demand element $D_l^{+,n}$ is fulfilled at decision epoch t' .
$x_l^{-,t'}$	Indicates that the demand element $D_l^{-,n}$ is fulfilled at decision epoch t' .
$x_l^{+,t,k,t'}$	Indicates that the demand element $G_l^{+,t,k,n}$ is fulfilled in sample k at decision epoch t' .
$x_l^{-,t,k,t'}$	Indicates that the demand element $G_l^{-,t,k,n}$ is fulfilled in sample k at decision epoch t' .
u_v^0	Number of resources present in the server v at current decision epoch.
$u_v^{t,k}$	Number of resources present in the server v at the decision epoch t in sample k
$z_{b,v}^{t,k}$	Indicates whether server v is present at station/location b at decision epoch t in sample k
$m_{b,v}^{0,t}$	Indicates whether server v moves towards b at current decision epoch and reaches b at t
$m_{b,v}^{t,t',k}$	Indicates whether server v moves towards b at t and reaches b at decision epoch t' in sample k

Table 10.1: Notation

MSS ():

$$\min \sum_l R_l + \frac{1}{|F|} \sum_{t=0}^{Q-1} \sum_k \sum_b R_b^{t,k} \quad (10.1)$$

$$s.t. \quad R_s^{t,k} = F_s^{t,k} - h_s^{t,k} \quad \forall s, t, k \quad (10.2)$$

$$R_l^{t,k} = \left[\left(1 - \sum_{t'=t}^{\pi_l^{+,t,k}} x_l^{+,t,k,t'} \right) \cdot G_l^{+,t,k,n} \right] \cdot G_l^{+,t,k,c} \\ + \left[\left(1 - \sum_{t'=t}^{\pi_l^{-,t,k}} x_l^{-,t,k,t'} \right) \cdot G_l^{-,t,k,n} \right] \cdot G_l^{-,t,k,c} \quad \forall l, t, k \quad (10.3)$$

$$R_l = \left[\left(1 - \sum_{t'=0}^{\tau_l^+} x_l^{+,t'} \right) \cdot D_l^{+,n} \right] \cdot D_l^{+,c} \\ + \left[\left(1 - \sum_{t'=0}^{\tau_l^-} x_l^{-,t'} \right) \cdot D_l^{-,n} \right] \cdot D_l^{-,c} \quad \forall l \quad (10.4)$$

$$y_{b,v}^{+,t,k} + y_{b,v}^{-,t,k} \leq C_v \cdot z_{b,v}^{t,k} \quad \forall b, v, t > 0, k \quad (10.5)$$

$$z_{b,v}^{t,k}, m_{b,v}^{0,t}, m_{b,v}^{t',k} \in \{0, 1\} \quad (10.6)$$

$$x_l^{+,t}, x_l^{-,t}, x_l^{+,t,k,t'}, x_l^{-,t,k,t'} \in \{0, 1\} \quad (10.7)$$

Constraints (3.3) – (3.24)

Table 10.2: MSS Formulation

Repositioning:

$$D_l^{-,n} \cdot x_l^{-,0} = \sum_v y_{l,v}^{-,0} \dots \forall l \quad (10.8)$$

$$D_l^{-,n} \cdot x_l^{-,t} + \sum_{t'=1}^t G_l^{-,t',k,n} \cdot x_l^{-,t',k,t} = \sum_v y_{l,v}^{-,t,k} \dots \forall l, k, t \quad (10.9)$$

$$D_l^{+,n} \cdot x_l^{+,0} = \sum_v y_{l,v}^{+,0} \dots \forall l \quad (10.10)$$

$$D_l^{+,n} \cdot x_l^{+,t} + \sum_{t'=1}^t G_l^{+,t',k,n} \cdot x_l^{+,t',k,t} = \sum_v y_{l,v}^{+,t,k} \dots \forall l, k, t \quad (10.11)$$

$$\sum_{t'=0}^{\tau_l^+} x_l^{+,t'} \leq 1; \sum_{t'=0}^{\tau_l^-} x_l^{-,t'} \leq 1 \dots \forall l \quad (10.12)$$

$$\sum_{t'=t}^{\pi_l^{t,k}} x_l^{-,t,k,t'} \leq 1; \sum_{t'=t}^{\pi_l^{t,k}} x_l^{-,t,k,t'} \leq 1 \dots \forall t, l, k \quad (10.13)$$

$$h_s^{t,k} \leq F_s^{t,k} \dots \forall s, t, k \quad (10.14)$$

$$h_s^{0,k} \leq d_s - \sum_v y_{s,v}^{+,0} + \sum_v y_{s,v}^{-,0} \dots \forall s, k \quad (10.15)$$

$$h_s^{t,k} \leq d_s^{t,k} - \sum_v y_{s,v}^{+,t,k} + \sum_v y_{s,v}^{-,t,k} \dots \forall s, t > 0, k \quad (10.16)$$

$$g_s^{t,k} \leq \sum_{t'=0}^{t-1} \sum_{s'} h_{s'}^{t',k} \cdot \frac{F_{s',s}^{t',t,k}}{F_{s'}^{t',k}} + \mathcal{X}_s^t \dots \forall s, t > 0, k \quad (10.17)$$

$$d_s^{t,k} = d_s^{t-1,k} - \sum_v y_{s,v}^{+,t-1,k} + \sum_v y_{s,v}^{-,t-1,k} - h_s^{t-1,k} + g_s^{t,k} \dots \forall s, t > 1, k \quad (10.18)$$

$$d_s^{1,k} = d_s - \sum_v y_{s,v}^{+,0} + \sum_v y_{s,v}^{-,0} - h_s^{0,k} + g_s^{1,k} \dots \forall s, k \quad (10.19)$$

$$d_s^{t,k} \leq C_s \dots \forall s, t, k \quad (10.20)$$

$$\sum_v y_{s,v}^{+,0} \leq d_s^0; \sum_v y_{s,v}^{-,0} \leq C_s^\# - d_s^0 \dots \forall s \quad (10.21)$$

$$\sum_v y_{s,v}^{+,t,k} \leq d_s^{t,k}; \sum_v y_{s,v}^{-,t,k} \leq C_s^\# - d_s^{t,k} \dots \forall s, k, t > 0 \quad (10.22)$$

$$u_v^{t,k} = u_v^{t-1,k} + \sum_s y_{s,v}^{+,t,k} - \sum_s y_{s,v}^{-,t,k} \dots \forall v, k, t > 0 \quad (10.23)$$

$$u_v^{0,k} = d_v + \sum_s y_{s,v}^{+,0} - \sum_s y_{s,v}^{-,0} \dots \forall v, k \quad (10.24)$$

$$u_v^{t,k} \leq C_v \dots \forall v, k, t \quad (10.25)$$

$$y_{b,v}^{+,0} + y_{b,v}^{-,0} \leq C_v \cdot \sigma_v^0(b) \dots \forall b, v \quad (10.26)$$

Table 10.3: Repositioning Constraints

Routing:

$$z_{b,v}^{t,k} = \sum_{t'=1}^{t-1} m_{b,v}^{t',t,k} + m_{b,v}^{0,t} + \sigma_v^t(b) \quad \forall b, v, t > 0, k \quad (10.27)$$

$$m_{b,v}^{t,t',k} \leq \sum_{b'} z_{b',v}^{t,k} \cdot \delta_{b',b,v}^{t,t'} \quad \forall b, v, t, k \quad (10.28)$$

$$m_{b,v}^{0,t} \leq \sum_{b'} \sigma_v^0(s') \cdot \delta_{b',b,v}^{0,t} \quad \forall b, v, t \quad (10.29)$$

$$\sum_{t'=t+1}^{Q-1} \sum_b m_{b,v}^{t,t',k} = \sum_{b'} z_{b',v}^{t,k} \quad \forall v, t, k \quad (10.30)$$

$$\sum_{t'=1}^{Q-1} \sum_b m_{b,v}^{0,t'} = \sum_{b'} \sigma_v^0(s') \quad \forall v \quad (10.31)$$

Table 10.4: Routing Constraints

mizing repositioning of resources. The variables used in MSS are described in Table 10.1. The total number of samples is equal to $|F|$ or $|G|$. Here onwards, we use $|F|$ as the number of samples. Table 10.2 presents the MSS formulation. Here are the key constraints that are not specific to only one of repositioning and routing:

Lost demand: Constraints (10.2)-(10.4) ensure that lost demand at any station, at any decision epoch is the difference between demand available and rented resources at the location. For the remote customers, we do not allow split pick-up/drop-off. Therefore, either all the demand is fulfilled or all the demand is lost. For remote customer pick-up requests as well, if the resources are not picked up, we consider it as a lost demand.

Validity of Pick-up/Drop-off: Constraints (10.5) ensure that a server v picks or drops resources from any station/location b at any decision epoch t in any sample k if and only if the station/location b is visited by server at decision epoch t in sample k .

Rest of the constraints are specific to either **Repositioning** or **Routing**. Table 10.3 presents the constraints related to the repositioning problem:

Remote customer demand : Constraints (10.8)-(10.9) ensure that if the demand

element $\mathcal{D}_l^{+,n}$ or $\mathcal{D}_l^{-,n}$ is satisfied at the timestep t , then the total number of resources picked up from the customer location at timestep t is exactly equal to the available demand. Similarly Constraint (10.10)-(10.11) ensure that if the demand element $\mathcal{D}_l^{-,n}$ is satisfied at the timestep t , then the total number of resources dropped at the location l at timestep t is exactly equal to the available demand. Constraint (10.12)-(10.13) ensure that each customer demand element is satisfied at exactly one timestep.

Rented resources : Constraints (10.14)-(10.16) ensure that at any station at any decision epoch, the number of rented resources is the minimum of available resources at station and walk-in demand at the station.

Returned resources: Constraint (10.17) compute the number of returned resources at any station and decision epoch as the sum of resources returned due to hiring at previous decision epochs (within formulation) and resources which will be returned due to previous rentals.

Resources availability: Constraints (10.18)-(10.20) ensure that the number of available resources at any station at any decision epoch is less than the capacity of that station. At any station, at any decision epoch, the number of resources available is calculated by considering rented resources at previous decision epoch, resources picked up/dropped by servers in previous decision epoch and resources returned by customers in current decision epoch.

Station capacity: Constraints (10.21)-(10.22) ensure that the number of resources picked up from any station is less than the number of available resources and the number of resources dropped at any station is less than the number of available free slots.

Server capacity: The number of resources present in any server v is calculated by considering resources picked up and dropped by server in current decision epoch and the number of resources already present in the server. Constraints (10.23)-(10.25) ensure that at any decision epoch, in any sample, the number of resources in any server v is less than capacity of server.

Table 10.4 contains the constraints related to routing problem. These constraints ensure that there exists a valid path between server positions at different decision epochs. $m_{b,v}^{0,t}$ variables are used to ensure that at current decision epoch, server moves towards same station/location across all samples. All the constraints in this case ensure that *movement of servers* are valid.

Constraint (10.27) ensure that a server v is present at station/location b , at decision epoch t , in sample k , if and only if, either it is reaching station/location b at decision epoch t due to previous assignments (given by σ_v^t values) or if it is going to reach station/location b at decision epoch t due to assignments which are part of current formulation. Constraints (10.28)-(10.29) ensure that a server v starts moving towards station/location b' at decision epoch t and reaches b' at decision epoch t' , if at decision epoch t it was present at some station/location b , such that the distance between b and b' can be covered in time $t' - t$. We use binary constants $\delta_{b,b',v}^{t,t'}$ to indicate if server v starting at station/location b at decision epoch t reaches station/location b' exactly at decision epoch t' or not. We assume a fixed travel time between stations/locations based on average speed of server so these binary constants can be calculated beforehand. Constraints (10.30)-(10.31) ensure that at any decision epoch, if a server is present at some station/location, then it will start moving towards exactly one of the stations/locations. A movement between the same station/location indicates server is staying at the same station/location.

We solve the MSS optimization formulation online at each decision epoch to compute the repositioning and routing strategy. At each decision epoch, distribution of resources at stations and server positions are updated based on actual realized customer requests and repositioning strategy executed by servers.

10.2 Lagrangian Dual Decomposition

Since we have to make decisions online, the solution has to be generated quickly. On increasing the number of stations/locations and the number of samples, the num-

ber of variables and constraints increases in MSS which makes it difficult to solve quickly. Decomposition across samples does not help in reducing the computation time, as for problems with large number of stations/locations, even for a single sample MSS takes a long time (thousands of seconds). Therefore, we extend the Lagrangian Dual Decomposition (LDD) (Fisher, 1985) method proposed by Ghosh et al. (Ghosh et al., 2015) for solving the offline repositioning and routing problem in bike sharing. Our key contributions within this LDD approach when applied to OLYMPIOD are two fold:(i) We update LDD to account for multiple demand samples; (ii) We significantly improve the computational complexity of solving the routing problem by using dynamic programming.

In our MSS formulation, as we can see in Table 10.2, only constraints (3.29) link the routing and repositioning variables across samples. Therefore, we dualize constraint (3.29) using price variables $\alpha_{b,v}^{t,k}$ and obtain Lagrangian as follows:

$$\begin{aligned} \mathcal{L}(\alpha) = \min & \left[\sum_l R_l + \frac{1}{|F|} \sum_{t=0}^{Q-1} \sum_k \sum_b R_b^{t,k} \right. \\ & \left. + \sum_{t=1}^{Q-1} \sum_v \sum_k \sum_s \alpha_{b,v}^{t,k} \cdot (y_{b,v}^{+,t,k} + y_{b,v}^{-,t,k} - C_v \cdot z_{b,v}^{t,k}) \right] \end{aligned} \quad (10.32)$$

$$\begin{aligned} = \min & \left[\frac{1}{|F|} \sum_{t=0}^{Q-1} \sum_k \sum_s R_s^{t,k} + \sum_v \sum_{t=1}^{Q-1} \sum_k \sum_b \alpha_{b,v}^{t,k} \cdot (y_{b,v}^{+,t,k} + \right. \\ & \left. y_{b,v}^{-,t,k}) \right] - \min \left[\sum_{t=1}^{Q-1} \sum_v \sum_k \sum_b (C_v \cdot \alpha_{b,v}^{t,k} \cdot z_{b,v}^{t,k}) \right] \end{aligned} \quad (10.33)$$

In equation (10.33) the first two terms correspond to the repositioning problem and last term corresponds to the routing problem. Therefore, we have a decomposition of dual problem into repositioning and routing slaves. The repositioning slave minimizes the first two terms of equation (10.33) subject to constraints (10.8)-(10.26) and (10.2)-(10.4). The routing problem minimizes the last term of equation (10.33) subject to constraints in Table 10.4.

As the task of routing constraints is to ensure the presence of a valid path between server positions at different timesteps and the objective of routing slave is to

minimize the weights of visited station timestep pairs, instead of solving it as an integer optimization problem, we can also solve it using dynamic programming to significantly improve efficiency. We can observe in the routing constraints (Table 10.4) that servers are independent of each other. Therefore, if we have a single sample, for each server the routing problem can be solved separately. The routing problem can be viewed as node weighted shorted path problem with each station at each decision epoch as graph node and node weights as $-1 \cdot C_v \cdot \alpha_{b,v}^{t,k}$.

In case of multiple samples, once the $m_{b,v}^{0,t}$ variables are fixed, samples are independent of each other. Therefore, for each server and each sample we can still solve using dynamic programming and at $t=0$, instead of taking minimum for individual sample, we take the minimum of sum of weights for all samples. Algorithm 10 provides the detailed steps. Steps 3-9 identify the starting station/location and timestep for the server. We then use $w_{b,v}$ variables to store the weight at the vertex and $a_{b,v}$ variables to store the path. Steps 15-18 are the key dynamic programming steps that update the weight at the vertices using backward induction. Steps 29-37 update the variables of the optimization formulation using the stored path in $a_{b,v}$ variables.

To obtain the solution to MSS, we optimize $\max_{\alpha} \mathcal{L}_{\alpha}$. Given an α , the dual value corresponding to the MSS is obtained by adding the solution from both slaves. The master optimization problem is solved iteratively using sub-gradient descent on price variables α as described in Algorithm 11. Convergence in the process is detected when difference between primal solution (defined as p in Algorithm 11) and dual solution (defined as sum of objective values of repositioning and routing slaves) is lesser than a small predetermined value (ϵ).

We need to extract a feasible primal solution from the solution obtained from slaves. The solution obtained from repositioning slave may not be consistent with the routes computed by routing slave but the solution obtained by routing slave is always feasible solution to the original MSS optimization. Therefore, to extract a feasible primal solution, we solve the MSS optimization by fixing the routing variables to the values obtained from routing slave.

Algorithm 10 SolveRouting(α)

```
1:  $obj = 0$ 
2: for  $v \in \mathcal{V}$  do
3:    $startts = -1$ 
4:   for  $t=0$  to  $Q-1$  do
5:     if  $\sigma_v^t(b) == 1$  then
6:        $startts \leftarrow t$ 
7:       if  $t > 0$  then
8:         for  $k = 1$  to  $|F|$  do
9:            $z_{b,v}^{t,k} \leftarrow 1$ 
10:     $t_1 = startts > 0 ? startts : startts + 1$ 
11:    for  $k = 1$  to  $|F|$  do
12:      for  $t = Q - 1$  to  $t_1$  do
13:        for  $b \in \mathcal{B}$  do
14:          if  $t==Q-1$  then
15:             $w_{b,v}^{t,k} \leftarrow -1 \cdot C_v \cdot \alpha_{b,v}^{t,k}$ 
16:          else
17:             $w_{b,v}^{t,k} \leftarrow \min_{b',t'}((w_{b',v}^{t',k} - C_v \cdot \alpha_{b,v}^{t,k}) \cdot \delta_{b,b'}^{t,t'})$ 
18:             $a_{b,v}^{t,k} \leftarrow \arg \min_{b',t'}((w_{b',v}^{t',k} - C_v \cdot \alpha_{b,v}^{t,k}) \cdot \delta_{b,b'}^{t,t'})$ 
19:          if  $startts == 0$  then
20:             $w_{b,v}^0 \leftarrow \min_{b',t'}(\sum_k (w_{b',v}^{t',k}) \cdot \delta_{b,b'}^{0,t'})$ 
21:             $a_{b,v}^0 \leftarrow \arg \min_{b',t'}(\sum_k (w_{b',v}^{t',k}) \cdot \delta_{b,b'}^{0,t'})$ 
22:             $obj \leftarrow w_{b,v}^0$ 
23:          else
24:            for  $k = 1$  to  $|F|$  do
25:               $obj += w_{b,v}^{startts,k}$ 
26:             $b \leftarrow startstation$ 
27:            if  $startts == 0$  then
28:               $b', t' \leftarrow a_{b,v}^0$ 
29:               $m_{b',v}^{0,t'} \leftarrow 1$ 
30:              for  $k = 1$  to  $|F|$  do
31:                 $z_{b,v}^{t',k} \leftarrow 1$ 
32:               $b \leftarrow b', t_1 \leftarrow t'$ 
33:            for  $k = 1$  to  $|F|$  do
34:               $t = t_1$ 
35:              while  $t < Q - 1$  do
36:                 $b', t' \leftarrow a_{b,v}^{t,k}$ 
37:                 $m_{b',v}^{t,t',k} \leftarrow 1, z_{b',v}^{t',k} \leftarrow 1$ 
38:                 $b \leftarrow b', t \leftarrow t'$ 
39: return  $obj, z, m$ 
```

Algorithm 11 SolveLDD()

 $\alpha \leftarrow 0, iter \leftarrow 0$ **repeat** $o_1, y^+, y^- \leftarrow \text{SolveRepositioning}(\alpha^{iter})$ $o_2, z, m \leftarrow \text{SolveRouting}(\alpha^{iter})$ $p \leftarrow \text{ExtractPrimal}(z, m)$ $\alpha_{b,v}^{t,k,iter+1} = \left[\alpha_{b,v}^{t,k,iter} + \gamma \cdot (y_{b,v}^{+,t,k} + y_{b,v}^{-,t,k} - C_v \cdot z_{b,v}^{t,k}) \right]_+$ $iter \leftarrow iter + 1$ **until** $p - (o_1 + o_2) \leq \epsilon$

10.3 Greedy Online Anticipatory Heuristic

There are many applications such as bikesharing, electric vehicle sharing which do not involve any remote customer requests. In these applications, resources need to be redistributed across stations to reduce the lost demand for walk-in customers. In this section, for these special cases, we provide Greedy Online Anticipatory Heuristic (GOAH) approach based on online anticipatory algorithms (Mercier & Van Hentenryck, 2007) that can quickly provide solution for large scale problems. Typically, online anticipatory algorithms are used to solve large scale online stochastic integer programs. These algorithms optimize for each sample scenario and then select the best solution over all samples. We use a similar idea to develop our approach but instead of optimally solving each sample, we approximate the value obtained for each sample due to scalability issues. In our case, solution for a sample would correspond to a set of repositioning and routing decisions for each server. Each server v has maximum $|\mathcal{S}|$ routing choices where $|\mathcal{S}|$ is the number of stations and it has C_v repositioning choices where C_v denotes the capacity of server v . As server v already has d_v resources in server, it can pick at most $C_v - d_v$ resources or it can drop at most d_v resources. So all possible solutions for each server are $C_v \cdot |\mathcal{S}|$.

Unlike in typical anticipatory algorithms where there is only one entity, here, we have multiple carrier vehicles and therefore, the space of possible joint solutions grows exponentially. To address this, we consider one server at a time and use the greedy algorithm (Algorithm 12) to pick the best server policy in each iteration and execute that policy. To pick the best server policy in each iteration, we can

use MSS/LDD for a single server to compute individual server’s policy. But using MSS/LDD for a single server will not provide desirable gain in runtime (as this will not reduce the complexity in the MSS/LDD formulations due to presence of multiple samples). Therefore, we use a heuristic approach, which computes policy for individual server in two steps: (i) Approximate computation of repositioning decisions (extra resources available for pick-up/drop-off) at each station, timestep in each sample. (ii) Compute policy (repositioning and routing decisions) for a server across all samples by using approximate repositioning decisions calculated in previous step.

Algorithm 12 GOAH()

```

 $e \leftarrow \text{ComputeApproxRepositioningValues}()$ 
 $V' = \phi$ 
while  $|V'| < |\mathcal{V}|$  do
  for  $v \in \mathcal{V} \setminus V'$  do
     $val_v = 0$ 
     $val_v, a_v \leftarrow \text{GetServerPolicy}(v, e)$ 
     $v' \leftarrow \arg \min_v val_v$ 
     $e \leftarrow \text{ExecutePolicy}(v', a_{v'}, e)$ 
     $V' \leftarrow V' \cup v'$ 

```

Approximate computation of repositioning decisions: We use $e_s^{t,k}$ to denote the extra resources available for pick-up or drop-off at station s timestep t in sample k . A positive $e_s^{t,k}$ value indicates availability of extra resources and a negative value indicates the number of resources which should be dropped to meet the lost demand. These values can be used by servers to decide the number of resources to pick-up/drop-off at a station timestep pair.

For each sample, we execute ”no repositioning” strategy, i.e., simulate the hiring and return of resources (according to demand observed in the sample) on the current distribution of resources at the stations, to calculate the available resources, lost demand and rented resources at each station timestep pair.

Since the resources which are not required at timestep t can be used to meet demand at timestep $t + 1$, if we use the current computed $e_s^{t,k}$ value, it can be a wrong indicator for server to pick resources. Therefore, we ensure that $e_s^{t,k}$ val-

ues are positive if and only if resources are not required at any future timesteps. Similarly, unallocated resources at time t can remain unallocated at time $t + 1$, so same resources will contribute to the $e_s^{t,k}$ values for two different timesteps. To forbid server from considering same resource as available for pick-up at two different timesteps, we assume that a station will be visited at most once. This is a valid assumption if the lookahead period is small (i.e., less than one hour).

As the station is visited only once, when server visits any station at any timestep, the decision of the number of resources picked up/dropped by server should consider future timesteps as well. Therefore, we update the $e_s^{t,k}$ values for each timestep to account for lost demand at future timesteps or requirement of extra resources at future timesteps. The updated $e_s^{t,k}$ values are used in next step, to guide the policy computation for server.

Computing policy for server given $e_s^{t,k}$ values: In this step, for each sample, we construct a graph. The nodes in the graph correspond to s, c, t , where s is the station id, c is the number of resources in the server and t is the timestep. An edge is created between node $\{s, c, t\}$ and node $\{s', c', t'\}$ if and only if $\delta_{s,s'}^{t,t'} = 1$ (i.e., s' is reachable from s). The edge between nodes $\{s, c, t\}$ and $\{s', c', t'\}$ indicates that at timestep t server is moving from station s towards station s' and reaches station s' at timestep t' . Positive value of $c - c'$ indicates server dropped $c - c'$ resources at station s and negative value indicates server picked $|c - c'|$ resources from station s . Therefore, an edge in the graph represents the routing and repositioning decision of server. We create an additional sink node T . The policy for a single server is computed by finding the maximum weighted path between server start position and node T .

The weight of edges is defined as follows:

$$R^k(\{s, c, t\}, \{s', c', t'\}) = \begin{cases} \min(0, e_s^{t,k} + c - c') & \text{if } c \leq c' \\ \min(c - c', -e_s^{t,k}) & \text{if } c > c' \text{ and } e_s^{t,k} < 0 \\ 0 & \text{otherwise} \end{cases}$$

$$R^k(\{s, c, t\}, T) = \begin{cases} \min(c, -e_s^{t,k}) & \text{if } e_s^{t,k} < 0 \\ 0 & \text{otherwise} \end{cases}$$

i.e., edges have positive weight on dropping resources at station with lost demand, negative weight on picking resources from a station with lost demand and 0 weight otherwise. In other words, weight of edge indicates reduction in lost demand. To incorporate the assumption of visiting a station only once (for picking up or dropping off resources) in the graph, we need to consider constrained graphs where a set of edges can not be part of a path. This problem is NP-hard (Ziegelmann, 2001). However, for $Q=3$, we can easily incorporate this assumption without using constrained graphs. This is because at $t=0$, server position is known, so we can avoid creating edges between same stations at different timesteps.

Once we compute policies for individual server, in each iteration, we execute the policy for server which maximizes the marginal reduction. As a result of executing server policy, the $e_s^{t,k}$ values are updated to take into account the pick-up/drop-off of resources at stations by server.

10.4 Experiments and Results

While the MSS and LDD approaches proposed in this dissertation are applicable to different OLYMPIOD problems, we perform our experiments on a highly relevant real-life domain of bikesharing systems. As mentioned before, in bikesharing systems there are no remote customers, therefore, \mathcal{D} and \mathcal{G} elements are empty. The

GOAH approach is also applicable for bikesharing systems.

In this section, we compare our approaches Multi Stage Stochastic Optimization (MSS), Lagrangian Dual Decomposition (LDD) and Greedy Online Anticipatory Heuristic (GOAH) with the following approaches:

1. Static Repositioning (STREP) - In this approach, stations are rebalanced at the end of the day. That is to say, no repositioning is performed during the planning period.
2. Expected Sample Offline Policy Generation (ESOF): In this case, mean demand between stations from past 30 days of data is used as a demand sample to generate offline repositioning policy. We execute MSS/LDD for this expected sample with Q as evaluation decision epochs for different Δ values to generate an offline policy¹.
3. Expected Sample Offline Policy Generation with revenue as objective (ESOF-Rev) (Ghosh et al., 2015): As the objective of the formulation is to maximize the revenue, the approach tries to minimize the cost of server movement in addition to minimizing lost demand. We compare the lost demand values and fuel cost with this approach.
4. Expected Sample Online Policy Generation (ESON): In this case we use the mean demand sample online in the MSS/LDD approaches.

We use $MSS(\Delta = x, Q = y)$, $LDD(\Delta = x, Q = y)$, $GOAH(\Delta = x, Q = y)$ and $ESON(\Delta = x, Q = y)$ to refer to our approaches when the time interval Δ is set to x and lookahead Q is set to y . We compare the value of lost demand for all the approaches. For MSS, LDD and ESON the time limit to compute solution is set as 1 minute.

Setup: We conducted our experiments by taking the demand distribution over 3 months from 2 real world bike sharing datasets. The first dataset is from Hubway

¹For $\Delta = 10$ (in minutes) and evaluation period of 6 hours, we use Q as 36.

BSS² which has 95 stations and the second dataset is from Capital BikeShare BSS³ which has 305 stations. For the Hubway dataset, we use 3 servers for repositioning of bikes and for the Capital BikeShare 6 servers are used.

	STREP	ESOF-Rev	ESOF($\Delta = 10$)
Hubway(3PM)	278.32	225.95	225.93
Capital BikeShare(3PM)	1320.93	960.30	795.28
Hubway(6AM)	184.82	130.30	122.69
Capital BikeShare(6AM)	508.27	456.96	410.60
	ESON ($\Delta = 10, Q = 6$)	LDD ($\Delta = 10, Q = 6$)	GOAH ($\Delta = 10, Q = 3$)
Hubway(3PM)	172.11	141.16	181.67
Capital BikeShare(3PM)	745.61	666.78	824.03
Hubway(6AM)	98.63	56.85	92.75
Capital BikeShare(6AM)	323.17	288.87	341.71

Table 10.5: Lost demand reduction

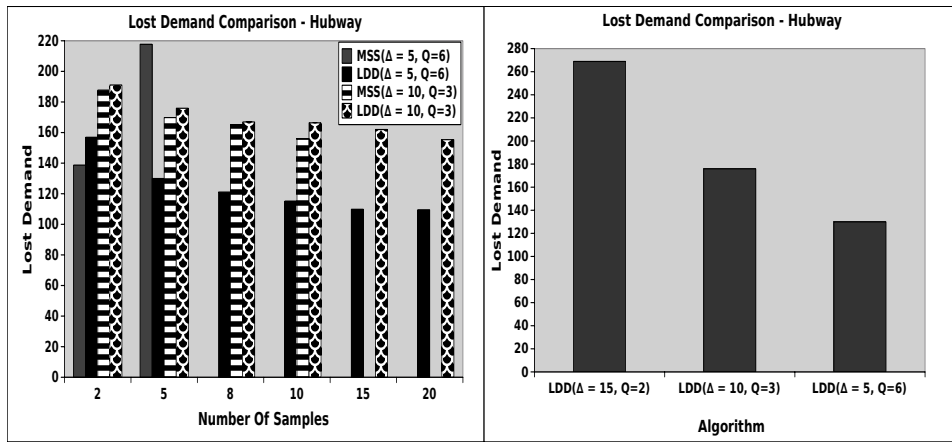
	ESOF-Rev	ESOF($\Delta = 10$)	ESON($\Delta = 10, Q = 6$)
Fuelcost	7.84	32.64	31.26
Revenue	431.10	422.87	461.11
Gain	423.25	390.23	429.85
	LDD($\Delta = 10, Q = 6$)	GOAH($\Delta = 10, Q = 3$)	
Fuelcost	30.60	26.60	
Revenue	488.58	441.72	
Gain	457.98	415.11	

Table 10.6: Fuel cost comparison-Hubway 3pm

As the historical trip data only contains successful bookings and does not capture the unobserved lost demand, we employ a micro-simulation model with 1 minute of timestep to identify the duration when a station got empty and introduce artificial demand at the empty station based on the observed demand at that station in previous timestep. In the 3 months trip data of both datasets, we have data for 60 weekdays. We use the first 30 weekdays to compute the mean demand sample which is used by ESOF and ESOF-Rev approaches. All the approaches are evaluated on remaining 30 weekdays and the average lost demand is computed over these 30 days.

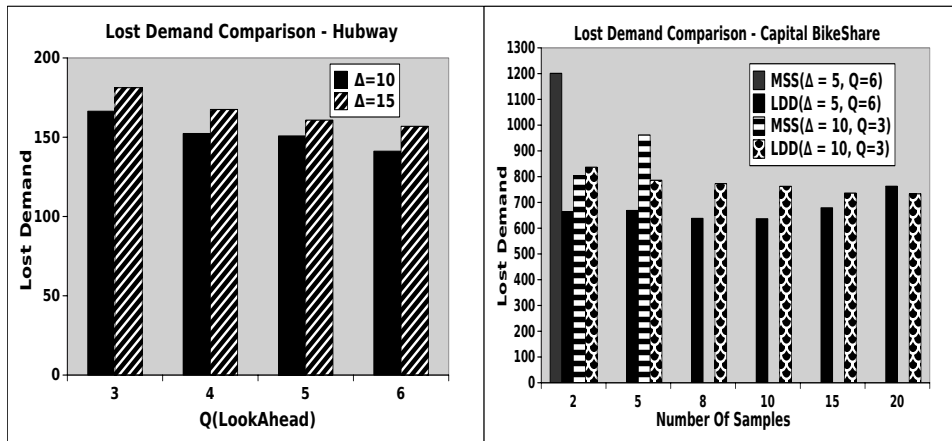
²<http://hubwaydatachallenge.org/trip-history-data/>

³<http://www.capitalbikeshare.com/system-data>



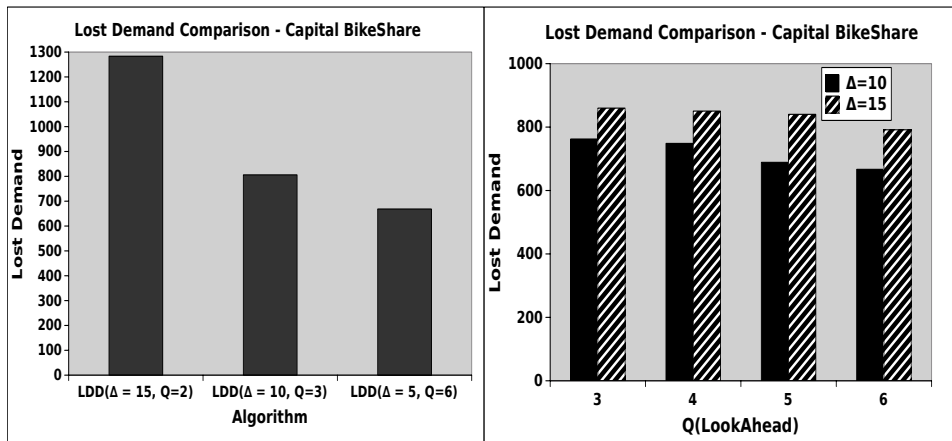
(a)

(b)



(c)

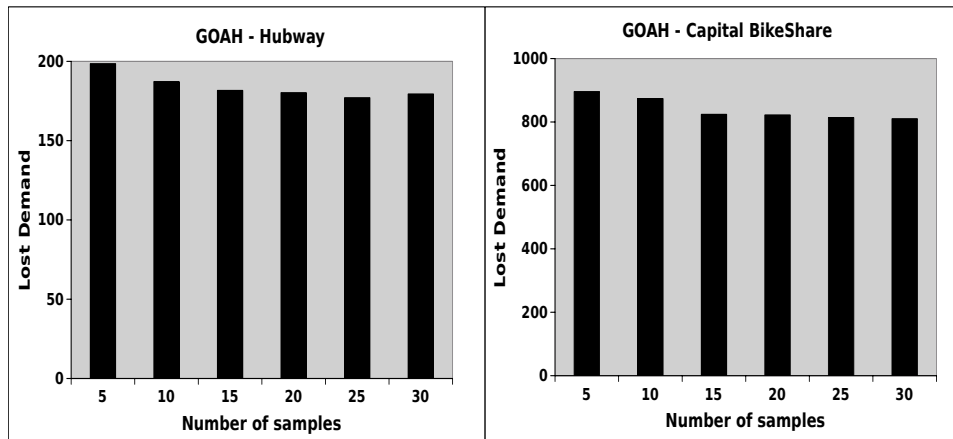
(d)



(e)

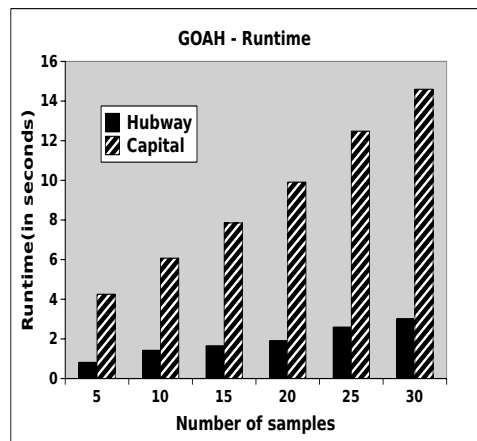
(f)

Figure 10.1: (a) and (d) Performance comparison of MSS and LDD. (b) and (e) Lost demand comparison of LDD for different Δ and Q values. (c) and (f) Lost demand comparison of LDD for different Q values.



(a)

(b)



(c)

Figure 10.2: Performance comparison of GOAH

Trip Duration	ESOF-Rev	ESOF($\Delta = 10$)	ESON($\Delta = 10, Q = 6$)
0-30	1166.74	1169.26	1212.17
30-60	93.43	91.24	102.84
60-90	11.27	11.11	11.62
>90	7.88	7.69	8.11
Trip Duration	LDD($\Delta = 10, Q = 6$)	GOAH($\Delta = 10, Q = 3$)	
0-30	1232.34	1205.55	
30-60	112.51	97.73	
60-90	12.50	11.39	
>90	8.22	7.95	

Table 10.7: Number of bikes rented for different trip duration (in minutes)- Hubway 3pm

While computing the repositioning and routing policy at decision epoch t , for our approaches MSS, LDD and GOAH, we consider k samples of customer requests at decision epoch $t, t + 1, \dots, t+Q-1$ from past k days (from the evaluation day at the same time). Once the repositioning and routing policy is computed, we evaluate the policy on realized customer demand.

We evaluate all the approaches using the simulation model described in previous chapter. The simulation model is run every one minute to serve the customer demand. In contrast, repositioning/routing strategy is computed at an interval of Δ minutes. After the repositioning/routing policy is obtained by algorithms at time $\Delta, 2 \cdot \Delta, \dots$ minutes, the availability of bikes at station and in servers is updated. In case there is no free slot at the station available while returning of bikes, the bikes are distributed in the nearby stations. The simulator is run for 6 hours for each day starting at different time of the day. We experimented with starting time as 6:00AM and 03:00PM. At the start of the experiment starting position of servers is randomly chosen. The objective of all the algorithms (except ESOF-Rev) is to minimize the lost demand⁴.

Results: We first show results for MSS and LDD by varying the values of Δ and Q . For GOAH, as described in Section 10.3, Q is fixed to 3. Therefore, we do not show results for GOAH for different Q values.

Number of Samples: In the first set of experiments on MSS and LDD, we experiment with different numbers of samples. Figure (10.1a) shows results for the Hubway dataset and Figure (10.1d) shows results for the Capital BikeShare dataset. The X-axis represents the number of samples and Y-axis represents the lost demand values. On the Hubway dataset, MSS could not compute a reasonable quality solution (the optimality gap remained at 80%) within 1 minute for more than 5 samples for $\Delta = 5$ and for more than 10 samples for $\Delta = 10$. On the Capital Bikeshare dataset, for $\Delta = 5$ and a time limit of 1 minute, MSS did not find a solution of reasonable quality even when only a single sample was used. On both the datasets, we

⁴All approaches have been implemented in Java using the IBM CPLEX 12.6.0.

observe that for LDD, lost demand reduces on increasing the number of samples, but the reduction in lost demand is less than 5% on increasing samples beyond 10. For $\Delta = 5$, on the Capital BikeShare, LDD could not complete even 10 iterations within 1 minute and hence, the lost demand increases on increasing samples from 10 to 15. As the lost demand reduction is not significant after 10 samples and the problem also becomes complex, we use 10 samples for LDD and MSS in the next experiments.

Decision Epoch Duration(Δ): In the next set of experiments, we fix the lookahead period for approaches to 30 minutes and experiment with different decision epoch duration. As $\Delta \cdot Q = 30$ minutes, the value of Q is taken as $\frac{30}{\Delta}$. Figure (10.1b) shows results on the Hubway dataset and Figure (10.1e) shows the results on the Capital BikeShare dataset. We fix the number of samples as 10. We only show results for LDD as MSS was not able to compute a reasonable quality solution within 1 minute for 10 samples for majority of scenarios. On both datasets, lost demand decreases on decreasing the value of Δ . This is because server is allowed to make more movements and also because we are looking at demand values at smaller intervals which allows making better online decisions. On decreasing Δ from 10 to 5, lost demand reduces by nearly 15% on both datasets.

Lookahead Period: For a fixed value of Δ , we experiment with different lookahead period durations. We show the results for $\Delta = 10$ with lookahead period between 30 minutes and 1 hour. For $\Delta = 15$, we show the results for lookahead period between 30 minutes and 1.5 hours. Figures (10.1c) and (10.1f) show the results for different Q values. On increasing the look ahead period lost demand reduces for both Δ values but the reduction is more with $\Delta = 10$. With $\Delta = 10$ and $Q = 6$ (i.e., lookahead period of 1 hour), the lost demand values are comparable to $\Delta = 5$ and $Q = 6$ (lookahead period of 30 minutes). With $\Delta = 10$, lost demand reduces by 12% on increasing Q value from 3 to 6. But the rate of reduction is low with higher Δ value. As the lost demand reduction provided by $\Delta = 10, Q = 6$ is comparable to lost demand reduction with $\Delta = 5$, we use $\Delta = 10, Q = 6$ for further comparison

as this involves lesser server movement.

GOAH Performance: We experiment with different number of samples for GOAH for $\Delta = 10, Q = 3$. In case of GOAH, along with lost demand we also compare the runtime on both datasets with different number of samples. Figure (10.2a) and (10.2b) show the lost demand comparison on increasing the number of samples. On both datasets, on increasing samples from 5 to 15, lost demand reduces by 8% but on increasing beyond 15 samples reduction is 2%. With 15 samples, on both datasets, GOAH obtains a runtime of less than 8 seconds (Figure (10.2c)). Therefore, it is possible to execute GOAH on larger bike sharing systems where it is even difficult for offline approaches to compute a solution. As described later, GOAH provides nearly 35% reduction in lost demand as compared to no repositioning strategy.

Comparison Of Different Algorithms: Next, we compare the reduction in lost demand values obtained by different algorithms. We compare LDD($\Delta = 10, Q = 6$), GOAH($\Delta = 10, Q = 3$) with ESOF($\Delta = 10$), ESOF-Rev, ESON($\Delta = 10, Q = 6$) and STREP on both datasets. For LDD we use 10 samples and for GOAH 15 samples. Table 10.5 shows the lost demand values on both datasets for 6AM and 3PM. As we can see on both datasets, LDD reduces the lost demand by nearly 50% as compared to STREP and provides 20% gain over ESOF and ESON. The lost demand reduction by GOAH is comparable to ESOF and ESON but it provides improvement in runtime which is the main advantage of using it against other approaches.

Comparison Of Fuel cost: Finally we compare the fuel cost incurred by different algorithms. We use the cost of diesel as 1.5 USD per litre and assume that the server can travel 12 kilometer with 1 litre of fuel. Here, we show the results on the Hubway dataset at 3pm. We obtained similar results on the other dataset. We then compare the fuel cost incurred by various algorithms in Table 10.6. As we do not consider the fuel cost in our objective, fuel cost of our algorithm is nearly 4 times the cost of fuel consumed by ESOF-Rev. We also compute the revenue obtained by bike sharing company by using the standard price model where only rides greater than

30 minutes are charged⁵. The revenue increase compensates for the additional fuel cost in case of LDD. With GOAH, both the fuel cost and revenue gain are less than LDD. Overall gain provided by GOAH is less than ESOF-Rev.

As the rides having travel time less than 30 minutes are included in the subscription cost, we also compare the number of bikes rented for different trip duration by various algorithms (Table 10.7). Once again, LDD provides the best results. As the major percentage of bikes are rented for duration 0-30 minutes, the lost demand reduction of these rides does not directly contribute to daily revenue. But this reduction will help in increasing the number of new subscribers which will provide additional profit to bike sharing companies.

10.5 Summary

In this chapter, we provide multi-period two-stage stochastic optimization and greedy online anticipatory heuristic approaches for efficient repositioning and routing in OLYMPIOD problems. We also provide a Lagrangian Dual Decomposition approach overcome the scalability issues. The empirical results on two real world datasets demonstrate that our approaches outperform the existing best known approaches.

⁵<https://www.thehubway.com/pricing/day>

Chapter 11

Conclusion and Future Work

In this dissertation, I presented future demand driven approaches to solve the online spatio-temporal demand supply matching problems and by extensive experiments on the real world datasets, we show that by considering expected future demand the quality of matching can be significantly improved. For a special case, when servers have a fixed location and they come back to their original location after serving the requests, we also provide theoretical bounds on the performance of algorithms.

11.1 Future Directions

For the future research, we highlight the following directions:

Improving competitive ratio bounds for M-OLYMPIAD problems: In this dissertation, we show that we can obtain a γ competitive (where γ is a solution to equation $\gamma = (1 - \gamma)^{\kappa+1}$ and κ is the capacity) online adaptive algorithm for special case of M-OLYMPIAD problems. As the proposed online adaptive algorithm performs sequential processing and does not use the full power of the batch, it is possible to improve the algorithm and obtain a better competitive ratio. Therefore, the first future direction is to improve the proposed algorithm. The other direction is to improve the existing upper bound on the competitive ratio for M-OLYMPIAD problems. Existing hardness results say that no algorithm can achieve a competi-

tive ratio of more than 0.823 (Manshadi et al., 2012) for unit capacity disposable resources. As the setting used by Manshadi et.al. (Manshadi et al., 2012) is a special case of M-OLYMPIAD problem, the upper bound of 0.823 is applicable for M-OLYMPIAD problems as well. The future direction is to create a worst case instance for which the best possible competitive ratio obtained by any online adaptive algorithm for M-OLYMPIAD problems is less than 0.823.

Handling of Multiple Types of resources for OLYMPIOD Problems: In this dissertation, we focus on only single type of resources for OLYMPIOD problems. In future, the work can be extended to provide scalable approaches for OLYMPIOD problems having multiple types of resources. In context of bikesharing, handling multiple types will allow us to differentiate between normal bikes which need to be transported between stations and faulty bikes which need to be taken to service centers. Also with the station-less bikesharing systems becoming popular, considering multiple types will also help us in assigning different priorities to bikes parked at valid and invalid locations (by considering them as different types).

Dynamic Pricing Scheme for OLYMPIAD Problems: In this dissertation, we considered a fixed revenue model based on distance for U-OLYMPIAD problems. For M-OLYMPIAD problems, we considered the objective of number of requests served. The revenue computation is much more challenging for multi-capacity problems, as it needs to consider about the inconvenience caused due to delay, number of passengers in the server during the course of journey (for which we need to take into account the future customer requests). It is important to come up with a pricing scheme which can consider the expected future demand and supply and is also profitable to the company. Banerjee *et.al.* (Banerjee, Johari, & Riquelme, 2015, 2016) have shown that while the performance of dynamic pricing does not exceed static pricing (which knows all system parameters) but dynamic pricing scheme is more robust to fluctuation in system parameters as opposed to static pricing scheme. They provide a detailed analysis for single region case but the extension to network

of regions is not well explored. Ma *et.al.* (H. Ma, Fang, & Parkes, 2018) design a spatio-temporal pricing mechanism and prove that under full observation, following the mechanism's dispatches forms a subgame perfect equilibrium among the drivers. Biswas *et.al.* (Biswas et al., 2018) assume that discount for sharing ride is a linear function of the fractional distance-wise detour and propose a iterative discount function learning algorithm to learn the discount parameter. They do not consider the expected demand and supply to compute the price. Therefore, there is a need to develop a dynamic pricing scheme which can computes the price of each assignment by taking into account the expected demand and supply and is also fair to the customers. The dynamic pricing scheme will be useful for unit-capacity OLYMPIAD problems as well because this will help in modelling the surge price scenarios.

Appendix A

Supplementary material for the Chapter 3

A.1 Proof of Proposition 1

Proposition 1: If $\forall j, i, i', t, t' \delta_{ij}^{t,t'} = \delta_{i'j}^{t,t'}$, then the U-OFLYMPIAD is reducible to a min cost flow problem.

Proof: Min cost flow optimization problems are polynomial time solvable, as they consider deterministic unconditional flows. For the case when each request completes at a fixed decision epoch irrespective of the server assigned to it, we show that the U-OFLYMPIAD problem also has deterministic unconditional flows and hence the optimization model for U-OFLYMPIAD is equivalent to the min cost flow optimization model. We first describe the network corresponding to the U-OFLYMPIAD problem.

Nodes: Nodes of the network represent the server locations and requests. Formally, we create a node for each element in the set \mathcal{L} at each decision epoch. A node is also created for each element in $\mathcal{D}^t, \forall t$. We also create a source node S and a sink node T . Therefore, the maximum number of nodes in the network will be $(|\mathcal{L}| + (|\mathcal{L}| \cdot |\mathcal{L}|)) \cdot M + 2$.

Edges: Intuitively, we have edges to indicate the assignment (server location to a

request) and movement (origin location to the destination location). Specifically, we have the following edges in the network.

- An edge is created between a node corresponding to the location i at the decision epoch t and a node for the element j of \mathcal{D}^t if $o_j \in f(i, t)$. The capacity of the edge is equal to R_j^t and the cost of the edge is $-C_{i,o_j,d_j}^t$. The flow on these edges is given by x_{ij}^t .
- An edge is created between the element j of \mathcal{D}^t and the node corresponding to the location i at decision epoch t' ($t' > t$), if $d_j = i$ and $\delta_j^{t,t'} = 1$ ($\forall j, i, i', t, t' \delta_{ij}^{t,t'} = \delta_{i'j}^{t,t'}$, therefore we define $\delta_j^{t,t'} = \delta_{ij}^{t,t'} \quad \forall i, j, t, t'$). The capacity of the edge is R_j^t and the cost is 0. We denote the flow on these edges by y_j^t .
- An edge is created between the node corresponding to the location i at decision epoch t and the node corresponding to the location i at decision epoch $t + 1$ to have the flow of unassigned servers, i.e., to ensure that the unassigned servers from the location i at decision epoch t remain in the same location at next decision epoch. The capacity of this edge is equal to $\sum_i \mathcal{N}_i^1$ and the cost is 0. The flow on these edges is denoted by w_i^t .
- From source node S , we create an edge to all the location nodes at the decision epoch 1. The capacity of edge between the source node S and the location i node is \mathcal{N}_i^1 (the number of initial servers in location i) and the cost is 0.
- If the requests present in the element j of $\mathcal{D}^t, \forall t$ complete at a decision epoch $> M$, then we create an edge between the element j of \mathcal{D}^t to the sink T . The capacity of the edge is R_j^t and the cost is 0.
- We also create edges from the location nodes at decision epoch M (last decision epoch) to the sink node T . The capacity of the edge is $\sum_i \mathcal{N}_i^1$ and the cost is 0.

Given this network, we can view the similarity of the two optimization models in Table A.1. The transformation uses slack variables (w_i^t) and the intermediate variables (y_j^t) to convert the original U-OLYMPIAD constraints into the flow and capacity constraints present in the min cost flow model. As the min cost flow is

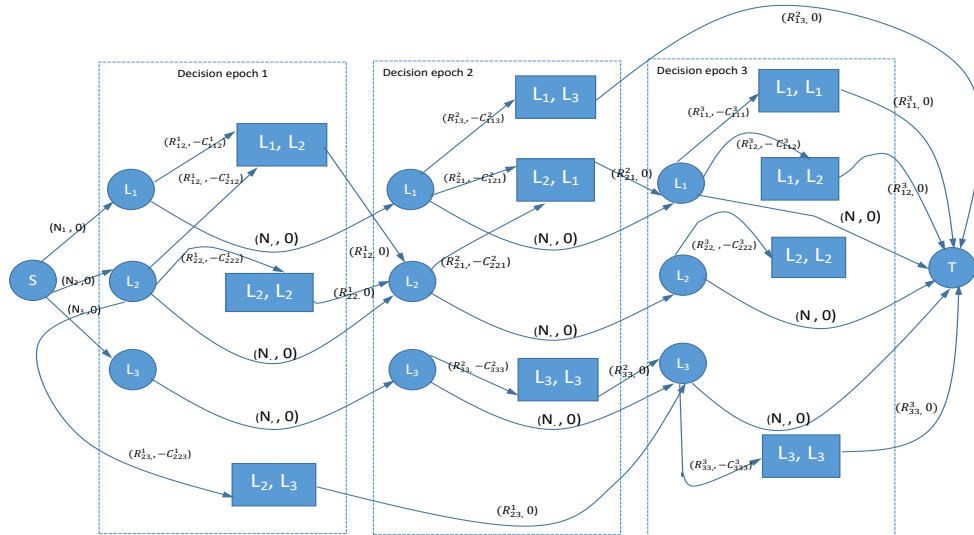
U-OFLYMPIAD($\mathcal{L}, \mathcal{D}, \mathcal{N}, \mathcal{C}, f, g, T, \delta, M$):	MinCostFlow: $G(V, E)$:
Objective	
$\min \sum_{t=1}^M \sum_{i \in \mathcal{L}} \sum_{\substack{j \in \mathcal{D}^t, \\ o_j \in g(i,t)}} -1 \cdot \mathcal{C}_{i,o_j,d_j}^t \cdot x_{ij}^t$ $+ \sum_{t=1}^M \sum_{i \in \mathcal{L}} w_i^t \cdot 0 + \sum_{t=1}^M \sum_{j \in \mathcal{D}^t} y_j^t \cdot 0 \quad (\text{A.1})$	$\min \sum_{(u,v) \in E} c_{uv} \cdot f_{uv} \quad (\text{A.2})$
Flow Constraints	
$\sum_{\substack{j \in \mathcal{D}^1, \\ o_j \in g(i,1)}} x_{ij}^1 + w_i^1 = \mathcal{N}_i^1 \quad \forall i \quad (\text{A.3})$ $y_j^t - \sum_{i \in f(o_j,t)} x_{ij}^t = 0 \quad \forall t, j \in \mathcal{D}^t \quad (\text{A.4})$ $\sum_{\substack{j \in \mathcal{D}^t, \\ o_j \in g(i,t)}} x_{ij}^t + w_i^t - w_i^{t-1}$ $- \sum_{t'=1}^{t-1} \sum_{\substack{j \in \mathcal{D}^{t'}, \\ d_j=i}} \delta_j^{t',t} \cdot y_j^{t'} = 0 \quad \forall i, t > 1 \quad (\text{A.5})$ $-1 \cdot \sum_{i \in \mathcal{L}} w_i^M + \sum_{t'=1}^M \sum_{\substack{j \in \mathcal{D}^{t'}, \\ d_j=i}} \delta_j^{t',M+1} \cdot y_j^{t'}$ $= -1 \cdot \sum_{i \in \mathcal{L}} \mathcal{N}_i^1 \quad (\text{A.6})$	$\sum_{k (j,k) \in E} f_{jk} - \sum_{i (i,j) \in E} f_{ij} = b_j \quad \forall j \in V \quad (\text{A.7})$
Capacity Constraints	
$0 \leq y_j^t \leq R_j^t \quad \forall t, j \in \mathcal{D}^t \quad (\text{A.8})$ $0 \leq x_{ij}^t \leq R_j^t \quad \forall i, t, j \in \mathcal{D}^t \quad (\text{A.9})$ $0 \leq w_i^t \leq \sum_{i \in \mathcal{L}} \mathcal{N}_i^1 \quad \forall i, t \quad (\text{A.10})$	$0 \leq f_{ij} \leq u_{ij} \quad \forall (i,j) \in E \quad (\text{A.11})$

Table A.1: MinCost Flow and U-OFLYMPIAD

polynomial time solvable and gives integral flow values for the integer capacities, we can solve the U-OFLYMPIAD also in polynomial time to get integer solutions.

■

Example 3. The example network with three locations and $M=3$ is shown in the Figure A.1. S and T denote the source and sink node. Circular nodes represent the server location nodes and rectangular nodes represent the request nodes between locations. Requests between the locations L_2 and L_3 at decision epoch 1 complete at decision epoch 3. Requests between the locations L_1 and L_3 complete at decision epoch 4 and as $M=3$, they are connected to sink node T . N denotes the total number of servers available initially, i.e., $\mathcal{N} = \mathcal{N}_1^1 + \mathcal{N}_2^1 + \mathcal{N}_3^1$. Each arc contains 2 values with the first value representing capacity and the second value representing the cost.

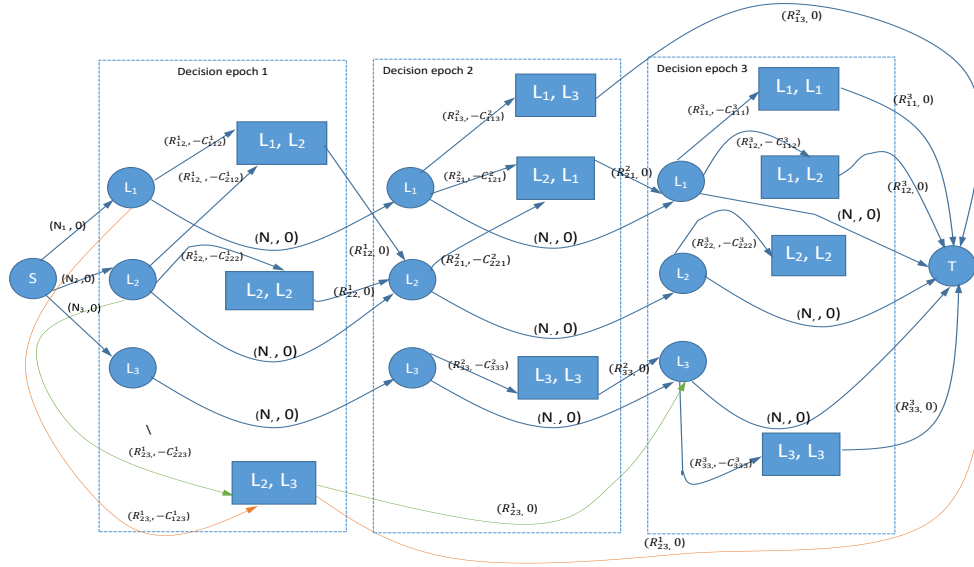


(a)

Figure A.1: Example min cost flow network for U-OFLYMPIAD

In general, depending on the server assigned, a request can have a maximum of $\lceil \frac{\tau}{\Delta} \rceil + 1$ different completion decision epochs. Therefore, in the network constructed in the Proposition 1, we will have edges between the decision epoch t request nodes to the decision epoch $t', t' + 1, \dots, t' + \lceil \frac{\tau}{\Delta} \rceil$ location nodes with the flow on these edges conditioned to be equal to the flow on the edges from the server node to request nodes.

In the example shown in the Figure A.1, suppose if the L_2 server is assigned to the request between locations L_2 and L_3 , then the request completes at decision epoch 3 and if the L_1 server is assigned to the request between locations L_2 and L_3 , the request will complete at decision epoch 4. The modified network is shown in Figure A.2. Now the node L_2, L_3 at decision epoch 1 will have the capacity R_{23}^1 with the condition that flow on 2 green edges should be equal and flow on 2 orange edges should be equal. Proposition 2 shows that in the above case where there are conditional flows, the U-OFLYMPIAD is NP-hard.



(a)

Figure A.2: Modified Example flow network for U-OFLYMPIAD

A.2 Proof of Proposition 2

Proposition 2: If $\exists j, i, i', t, t'$ s.t. $\delta_{ij}^{t,t'} \neq \delta_{i'j}^{t,t'}$, then the U-OFLYMPIAD is NP-hard.

Proof: To show that the U-OFLYMPIAD is NP-hard in general case, we reduce the well known 3-SAT problem to U-OFLYMPIAD. We construct an instance of the U-OFLYMPIAD for any arbitrary instance of 3-SAT with L clauses and V variables. We show that we obtain the optimal value for the U-OFLYMPIAD if and only if there exists an assignment to variables in 3-SAT formula such that all the clauses in

3-SAT will evaluate to true.

For any arbitrary instance of 3-SAT with L clauses and V variables, we construct an instance of U-OFLYMPIAD as follows:

- We create an U-OFLYMPIAD problem instance with $4L + 1$ decision epochs, i.e., $M=4L + 1$. At first decision epoch, we create $2V$ requests. At remaining $4L$ decision epochs, we create one request each.
- We create $2V+1$ locations, denoted by L_0 and $L_i, L'_i, i \leq V$. So the set of locations $\mathcal{L} = \{L_0, L_i, L'_i, i \leq V\}$.
- Initially locations L_i have one server each and locations L_0 and L'_i have zero servers.
- We create $2V$ requests at first decision epoch. Each location L_i is the origin location of two requests with one request having destination in L_i and one request having destination in L'_i . Requests having origin in L_i at decision epoch 1, can only be assigned server from the location L_i . Therefore, $D^1 = \{\langle L_i, L_i, 1 \rangle, \langle L_i, L'_i, 1 \rangle\}$ and $f(L_i, 1) = \{L_i\}$. All the requests at the first decision epoch have unit revenue.
- If location i server is assigned to the request between location pairs $\langle L_i, L_i \rangle$ then the variable x_i is true else if it is assigned to the request between location pairs $\langle L_i, L'_i \rangle$ then x_i is false. Requests served at decision epoch 1 determines the value of variables x_i . Therefore, at second decision epoch, server will be available in location L_i if x_i is true, and will be available in the location L'_i if x_i is false.
- If there are L clauses, we create L requests with request corresponding to k^{th} clause at decision epoch $4k - 2$. Each of these requests have destination in the location L_0 . The origin location of these requests is the location corresponding to the first literal of the k^{th} clause.

In addition, if the k^{th} clause has literal x_i then the location L_i server can be assigned to the request at decision epoch $4k - 2$ and if the k^{th} clause has literal $\neg x_i$, then the location L'_i can be assigned to the request. On the other hand, if the

k^{th} clause does not contain literal x_i then location L_i server can not be assigned to the request at decision epoch $4k - 2$. In short, request at the decision epoch $4k - 2$ can be assigned server only from 3 locations which correspond to the literals present in the k^{th} clause. Therefore, the request at decision epoch $4k - 2$ will have at least one server available if clause evaluates to true, i.e., if at least one of the literals has true value.

On assigning a server from the location corresponding to the first literal, request completes at decision epoch $4k - 1$ earning revenue 1, on assigning a server from the location corresponding to second literal, request completes at decision epoch $4k$ earning revenue 2 and on assigning a server from the location corresponding to the third literal, request completes at decision epoch $4k + 1$ earning revenue 3.

- At decision epoch $4k - 1$, a request is present between location L_0 and the location corresponding to first literal of k^{th} clause with revenue 3. At decision epoch $4k$, a request is present between location L_0 and the location corresponding to second literal of k^{th} clause with revenue 2. Similarly, at the decision epoch $4k + 1$, a request is present between locations L_0 and the location corresponding to the third literal of k^{th} clause with revenue 1.

The requests at decision epoch $4k - 1$, $4k$ and $4k + 1$ can be assigned a server only from location L_0 . The server will be available in location L_0 only if the request at decision epoch $4k - 2$ is served. As there is only one request available at the decision epoch $4k - 2$, maximum one of these three requests can be served. The maximum revenue which can be earned by serving the requests between decision epochs $4k - 2$ to $4k + 1$ is 4.

After serving the request at decision epoch $4k - 2$ and one of the requests at decision epochs $4k - 1$, $4k$ or $4k + 1$ such that total revenue is 4, the availability of servers in the locations at decision epoch $4k + 2$ will be same as decision epoch $4k - 2$. Therefore, next clause will be evaluated for the same assignment of variables.

We now show that there is an assignment of values to the variables in the 3-SAT instance so that the formula evaluates to true if and only if there exists a solution to the U-OFLYMPIAD problem with objective value $V + 4L$.

The “if” direction: Suppose there exists a solution with the objective value $V + 4L$. As maximum revenue which can be earned by serving the requests between decision epochs $4k - 2$ and $4k + 1$ is 4 and the maximum revenue earned at first decision epoch is V , it means that V requests are assigned a server at first decision epoch earning a total revenue V and one request is served at each of the decision epochs $4k - 2, \forall k = 1..L$ and one request is served from every three decision epochs $4k - 1, 4k$ and $4k + 1$ earning a total revenue of $4L$. The variable x_i is set to true if at first decision epoch location L_i server is assigned to the request having destination in location L_i otherwise it is set to false.

This will be a solution to 3-SAT instance as all the request at decision epoch $4k - 2$ are served (i.e., all clauses are true) and as the revenue earned between decision epochs $4k - 2$ and $4k + 1$ is 4, at decision epoch $4k + 2$ the servers availability is same as at decision epoch 2.

So if there is a solution to the U-OFLYMPIAD problem, we can find an assignment for 3-SAT instance.

The “only if” direction: Suppose there is an assignment of values to the variables such that the 3-SAT formula evaluates to true. So at decision epoch 1, if x_i is true, we assign L_i server to request having destination in location L_i otherwise it is assigned to the request having destination in location L'_i . Therefore, revenue earned at decision epoch 1 will be V . Now, as the 3-SAT formula evaluates to true, at decision epoch 2, we will have at least one server available to serve the request. If the first literal of the first clause is true we assign it to request at decision epoch 2 and serve the request at decision epoch 3 earning a revenue of 4. If first literal is false but second literal is true, then we assign it to request at decision epoch 2 and serve the request at decision epoch 4 earning a revenue of 4. If first and second literal are false but the third literal is true then we assign it to request at decision epoch 2

and serve the request at decision epoch 5 earning a revenue of 4. Therefore we can serve request at decision epoch 2 and one of the requests at decision epoch 3, 4 or 5 and earn a total revenue of 4. At decision epoch 6 the servers, the availability of servers will be same as at decision epoch 2. As all the clauses of 3-SAT evaluate to true, the second clause will also be true and at decision epoch 6 we will have at least one server available to serve the request. Therefore, for each clause in 3-SAT, we will serve 2 requests earning a revenue of 4 resulting in objective value of the U-OFLYMPIAD to be $V + 4L$ ■.

Example 4. We show the graphical representation in the Figure A.3 for an example 3-SAT clause $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$. Initially, there is one server available in location L_1, L_2 and L_3 denoted by flow of 1 from the source S . At each decision epoch, the circular nodes represent the server location nodes and rectangular nodes represent the request nodes with vertex capacity as the number of requests between the location pairs. The edge between the server node and the request node at a decision epoch represent that server can be assigned to the request. The revenue obtained on assigning a server to the request is marked on the edge. The value of flow on the edge will represent the number of location servers assigned to request. At first decision epoch, depending on the assignment of server to the requests, the value of x_1, x_2, x_3 will be 1 or 0 (true or false). At second decision epoch, the request node has edges from L_1, L_2 and L_3 . That is the server will be assigned to the request if one of these locations has a server available. Also as the capacity of node is 1, only one of the servers from these locations will be assigned a request. If server of L_1 is assigned to the request at decision epoch 2, the black edges represent the movement of server. After serving the request at decision epoch 3, the server will become available in the location L_1 again. Similarly green edges show the movement of server if server from location L_2 is assigned and orange edges show the movement of server, if server from location L_3 is assigned. Unassigned servers at decision epoch 2, will remain in the same locations and their movement is represented through dotted lines in the graph. At decision epoch 6, the server

distribution in the locations will be same as decision epoch 2 as the servers which moved between decision epoch 2 and 6 came back into the same location at decision epoch 6.

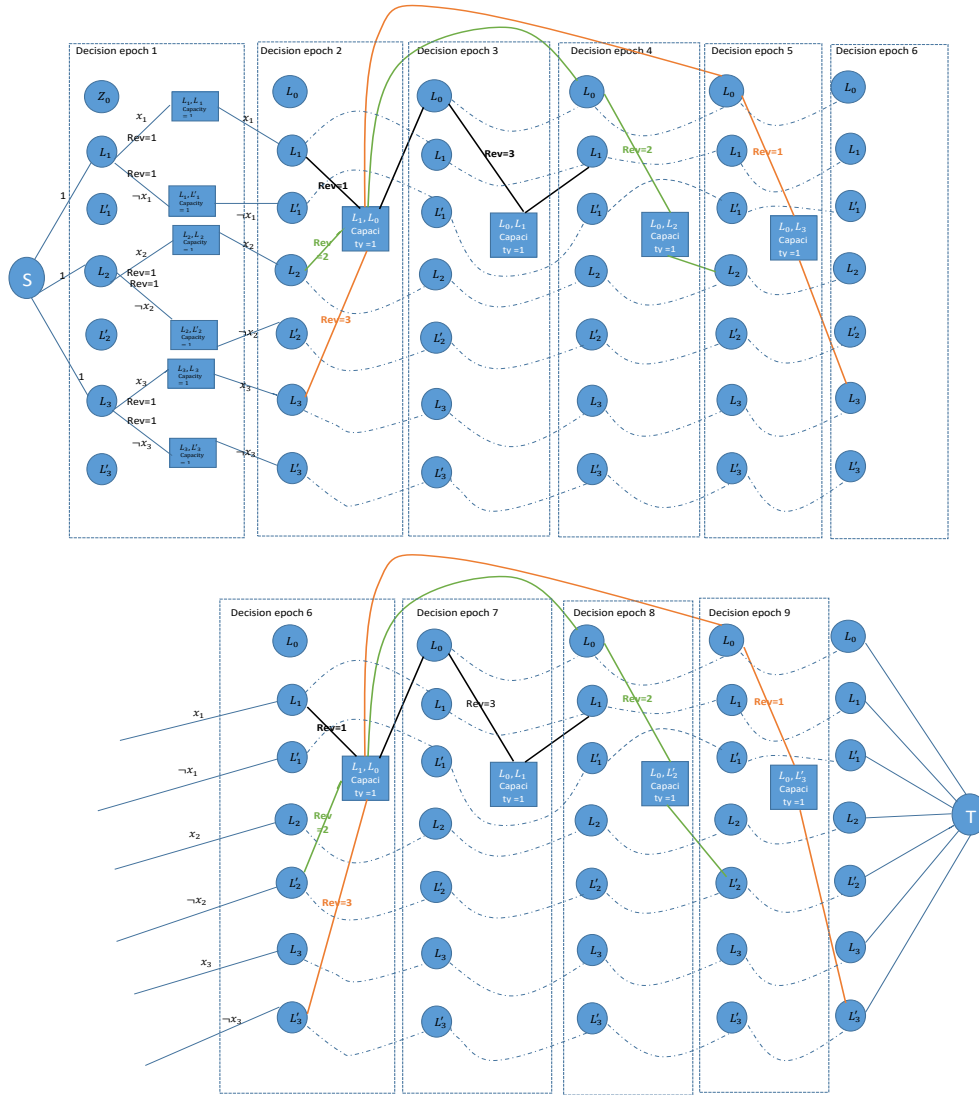


Figure A.3: Example clause - $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$

A.2.1 Proof of Proposition 3

Proposition 3: Solving TSS for more than one sample is an NP-hard problem irrespective of the δ values.

Proof: To show that solving TSS for more than one sample is NP-hard, we re-

duce the 3-SAT problem to TSS. We construct an instance of **TSS** for any arbitrary instance of 3-SAT with L clauses and V variables. We show that we obtain the optimal value of $V + 1$ for **TSS** if and only if there exists an assignment of variables such that the 3-SAT clause evaluates to true.

Each clause of 3-SAT corresponds to one sample in **TSS**. Each sample has one request, i.e., if the clause evaluates to true, request in the sample is served by a server else it will not be served. The first stage requests are created so that they consider all possible (true/false) values for the variables. Intuitively, the first stage decides the assignment of variables present in clauses and the second stage evaluates the clauses for those variable assignments.

The detailed steps are as follows:

- If there are V variables L clauses we create $2V + L$ locations, denoted by $\{L_i, L'_i, i \leq V\}$ and $\{L_{sk}, k = 1, 2, \dots, L\}$.
- Initially locations L_i have one server each and other locations have zero servers.
- We create $2V$ requests at the first stage. Each location L_i is the origin location of two requests with one request having destination in L_i and one request having destination in L'_i . Requests having origin in location L_i can only be assigned servers from locations L_i . Therefore, $D^1 = \{\langle L_i, L_i, 1 \rangle, \langle L_i, L'_i, 1 \rangle \mid i \leq V\}$, $f(L_i, 1) = \{L_i\}$. All the requests have unit revenue.
- If the location L_i server is assigned to the request having destination in location L_i then the variable x_i is true else if it is assigned to the request having destination in the location L'_i then x_i is false. Requests served at the first decision epoch determine the value of variables x_i . Therefore, at the second stage, server will be available in the location L_i if x_i is true, and will be available in location L'_i if x_i is false.
- If there are n clauses, we create n requests (one request in each sample). $|\xi^D| = L$ and $\xi_2^{D,k}$ has one element. Request in k^{th} sample has origin in the location L_{sk} and destination in $L_1, \xi_2^{D,k} = \{\langle L_{sk}, L_1, 1 \rangle\}$. All requests have unit revenue.
- If the k^{th} clause has literal x_i then location L_i server can be assigned to the

request in sample k , and if the k^{th} clause has literal $\neg x_i$, then location L'_i can be assigned to the request in sample k . On the other hand, if the k^{th} clause does not contain literal x_i then location L_i server can not be assigned to the request in sample k . In short, request in k^{th} sample can be assigned server only from 3 locations which correspond to literals present in k^{th} clause.

- Now, the request in sample k will have at least one server available if clause evaluates to true, i.e., if one of the literals has true value.

We now show that there is an assignment of values to the variables in the 3-SAT instance so that the formula evaluates to true if and only if there exists a solution to **TSS** with objective value $V + 1$.

The “if” direction: Suppose there exists a **TSS** solution with the objective value $V + 1$, it means that V requests are assigned a server at the first stage and in all the L samples requests are assigned a server. The variable x_i is set to true if at the first stage location L_i server is assigned to the request having destination in location L_i otherwise it is set to false. This will be a solution to the 3-SAT instance as requests in all the samples are served so at least one of the literals in each clause is set to true. So if there is a solution to **TSS** instance, we can find an assignment for 3-SAT instance.

The “only if” direction: Suppose there is an assignment of values to the variables such that the 3-SAT formula evaluates to true. So if x_i is true, we assign L_i server to request having destination in location L_i otherwise it is assigned to the request having destination in location L'_i . Now, in each sample for each request, we will have at least one server available so we can serve all the L requests at the second stage. Therefore objective value of **TSS** will be $V + \frac{1}{L} \cdot L = V + 1$. ■

A.3 Proof of Proposition 4

Proposition 4 *In U-OLYMPIAD without sample information and adversarial behavior from environment, when maximizing the number of requests satisfied for a*

fixed number of servers N , the competitive ratio, c for any deterministic b -stage algorithm (i.e., with information available up to the b^{th} decision epoch) in a M -decision epoch ($M \geq b$) problem is

$$c \leq \frac{1}{M - b + 1}$$

Proof. Before we describe the key elements of the proof, we first provide the key terms that will be used in this proof:

- ALG denotes the value of the best deterministic b -stage algorithm over M decision epochs.
- OPT denotes the value obtained by an M -Stage optimal algorithm over M decision epochs.
- N is the number of servers available.
- Let OPT_b denote the value of optimal solution for first b decision epochs (i.e., the maximum number of requests which can be served in first b decision epochs).

In order to show the upper bound on competitive ratio, we will consider different cases on values that can be taken by o . Since we are computing competitive ratio (least value of $\frac{ALG(I)}{OPT(I)}$), we identify the least value of the numerator and the highest value of the denominator.

- (1) $OPT_b \geq N$, i.e., the number of requests served in the first b decision epochs is greater than N

As ALG denotes the number of requests served by the best deterministic b -stage algorithm over M decision epochs. Therefore, at the very least, it can obtain the

optimal solution for b decision epochs and hence:

$$ALG \geq OPT_b$$

Since OPT is the value obtained by an M -stage optimal algorithm, it can potentially serve N requests for every one of the remaining $(M - b)$ time steps.

Therefore,

$$OPT \leq (M - b) \cdot N + OPT_b$$

Hence, we have:

$$c \leq \left(\frac{OPT_b}{OPT_b + N \cdot (M - b)} \right) = \left(\frac{1}{1 + \frac{(M-b)}{\frac{OPT_b}{N}}} \right)$$

The above expression will be minimum when $\frac{OPT_b}{N}$ is minimum. As $OPT_b \geq N$, the minimum value of $\frac{OPT_b}{N}$ is 1. Therefore,

$$c \leq \frac{1}{M - b + 1}$$

(2) $OPT_b < N$, i.e., the number of requests served in first b decision epochs is lower than N

As the number of requests served in first b decision epochs is lower than N , there will be some servers which did not move from their initial position. So if optimal algorithm uses these servers to serve requests, deterministic algorithm can also serve requests using them, As the minimum value of ALG and OPT is OPT_b , we take $ALG = OPT_b + x$ and $OPT = OPT_b + y$. Assume the

competitive ratio in this case is better than $\frac{1}{1+M-b}$. Therefore,

$$\frac{OPT_b + x}{OPT_b + y} < \frac{1}{1 + M - b} \quad (\text{A.12})$$

$$\implies (OPT_b + x) \cdot (M - b) + OPT_b + x < OPT_b + y \quad (\text{A.13})$$

$$\implies y > OPT_b \cdot (M - b) + x \cdot (M - b + 1) \quad (\text{A.14})$$

As the maximum number of requests served in remaining $M - b$ decision epochs is $N \cdot (M - b)$,

$$y \leq N \cdot (M - b) \quad (\text{A.15})$$

From equation (A.14) and (A.15),

$$\begin{aligned} & OPT_b \cdot (M - b) + x \cdot (M - b + 1) < N \cdot (M - b) \\ \implies x & < (N - OPT_b) \cdot \frac{M - b}{M - b + 1} \\ \implies x & < N - OPT_b \\ \implies x + OPT_b & < N \end{aligned}$$

As the value of ALG in all M decision epochs is less than N , $N - (OPT_b + x)$ servers did not move from their initial position. Therefore, even for the optimal algorithm, $N - (OPT_b + x)$ servers will not be moving. So,

$$\begin{aligned} & y \leq (OPT_b + x) \cdot (M - b) \\ \implies OPT_b + y & \leq (OPT_b + x) \cdot (M - b) + OPT_b \\ \implies \frac{OPT_b + y}{OPT_b + x} & \leq (M - b) + \frac{OPT_b}{OPT_b + x} \\ \implies \frac{OPT_b + x}{OPT_b + y} & \geq \frac{1}{\frac{OPT_b}{OPT_b + x} + (M - b)} \end{aligned}$$

RHS $> \frac{1}{M-b+1}$. But we assumed $\frac{OPT_{b+x}}{OPT_{b+y}} < \frac{1}{M-b+1}$ which is a contradiction. Thus,

$$c \leq \frac{1}{M-b+1} \blacksquare$$

We can also extend the above reasoning for the case when objective is to maximize revenue. If revenue of any request $C \in \{C_{min}, C_{max}\}$, then at first b stages, requests with revenue C_{min} are served by both deterministic and optimal algorithm and for the remaining decision epochs requests with revenue C_{max} are served by optimal algorithm. Therefore, the competitive ratio will be $\frac{C_{min}}{C_{min}+(M-b) \cdot C_{max}}$.

The competitive ratio is low mainly due to the assumption that server only moves when it is assigned to a request. Therefore, the adversary can take more advantage by creating requests in the locations which are not reachable from the server position.

It should be possible to improve the competitive ratio on removing this assumption. By taking decision to move randomly to another location if no request is available at the current stage, we may improve the competitive ratio against an online adversary (i.e., the adversary who is not aware of the output of random decision).

A.4 Proof of Proposition 5

Proposition 5: In U-OLYMPIAD with sample information and stochastic behaviour from environment according to the samples, when maximizing the number of requests satisfied, the expected competitive ratio, c_μ , of the TSS algorithm is

$$c_\mu \leq \frac{3}{4 \cdot (M-1)} + \frac{3}{4 \cdot M}$$

where M is the number of decision epochs ($M \geq 3$).

Proof: To prove the upper bound on the expected competitive ratio, we construct a worst case distribution and show the value of expected competitive ratio for that distribution. The value of expected competitive ratio for any specific distribution is

upper bound on the true value.

We assume that there is a positive probability of having a request at second decision epoch ¹. As the **TSS** only knows distribution for the next decision epoch, we create an instance such that **TSS** can not serve any requests after first two decision epochs.

We construct a worst case instance with one server and two requests at first two decision epochs. Initially, the server is located in location L_1 , the two requests are request 1, $r_1 = \langle L_1, L_1 \rangle$ and request 2, $r_2 = \langle L_1, L_2 \rangle$. Requests originating in location L_1 can only be served by server in location L_1 . Similarly request originating in location L_2 can only be served by server in L_2 .

As per the distribution at decision epoch 2, the probability that request has origin and destination in L_1 is P_1 and probability that request has origin and destination in L_2 is P_2 and the probability is zero for all other location pairs. P_1 and P_2 are independent of each other. Also as per the distribution, at subsequent decision epochs, there is zero probability of a request having origin in location L_1 and 1 probability of requests having origin and destination at L_2 .

The **TSS** algorithm will make the first stage assignment based on the expected number of requests served, i.e., it will maximize $1 + \max(P_1, P_2)$. If $P_1 \geq P_2$, **TSS** will assign the server to request 1 at decision epoch 1 and will serve request at decision epoch 2 with probability P_1 . **TSS** will not be able to serve any more requests at subsequent decision epoch.

The M-stage optimal algorithm will assign server to request 2 at first decision epoch. At second decision epoch, the M-stage optimal algorithm will be able to serve the request if it originates in location L_2 otherwise it will not. The M-stage algorithm will be able to serve requests at all subsequent decision epochs². Therefore,

¹If the probability of having a request at second decision epoch is 0, **TSS** will not have any future information available and will be as good as deterministic one-stage algorithm. In case of zero probability, from Proposition 4, the competitive ratio will be $\frac{1}{M}$.

²The number of decision epochs (M) is greater than or equal to 3

the expected competitive ratio is given by

$$c_\mu \leq P_1 \cdot (1 - P_2) \cdot \frac{2}{M-1} + P_1 \cdot P_2 \cdot \frac{2}{M} + (1 - P_1) \cdot P_2 \cdot \frac{1}{M} + (1 - P_1) \cdot (1 - P_2) \cdot \frac{1}{M-1} \quad (\text{A.16})$$

The four terms correspond to four possible cases of drawing 2 requests from the given distribution. Rearranging the terms, we get

$$c_\mu \leq (1 + P_1) \cdot \frac{1}{M-1} - P_2 \cdot (1 + P_1) \cdot \left(\frac{1}{M-1} - \frac{1}{M} \right) \quad (\text{A.17})$$

From equation (A.17), as $\frac{1}{M-1} > \frac{1}{M}$, for a fix value of P_1 , on increasing the value of P_2 the expected competitive ratio decreases. As $P_1 \geq P_2$, to minimize the expected value of competitive ratio, we take $P_2 = P_1$. Substituting $P_2 = P_1$ in equation (A.17), we get

$$c_\mu \leq (1 + P_1) \cdot \frac{1}{M-1} - P_1 \cdot (1 + P_1) \cdot \left(\frac{1}{M-1} - \frac{1}{M} \right)$$

The above expression will be minimum when the derivative (with respect to P_1) is 0, i.e.,

$$\begin{aligned} (1) \cdot \frac{1}{M-1} - (1 + 2 \cdot P_1) \cdot \left(\frac{1}{M-1} - \frac{1}{M} \right) &= 0 \\ \implies 2 \cdot P_1 \cdot \left(\frac{1}{M-1} - \frac{1}{M} \right) &= \frac{1}{M} \\ \implies P_1 &= \frac{M-1}{2} \end{aligned}$$

As $0 \leq P_1 \leq 1$, and M is a positive integer, the possible value of P_1 are 0, 0.5 and 1.0. Since, we have a positive probability of having request at second decision epoch, $P_1 = 0$ is not possible. Therefore, the competitive ratio will be minimum when $P_1 = 0.5$ or 1. On substituting $P_1 = P_2 = 1.0$, in equation (A.17), we get $\frac{2}{M}$ and on substituting $P_1 = P_2 = 0.5$, in equation (A.17), we get $\frac{3}{4 \cdot (M-1)} + \frac{3}{4 \cdot M}$

As $\frac{2}{M} > \frac{3}{4 \cdot (M-1)} + \frac{3}{4 \cdot M}$, for $M \geq 3$, therefore

$$c_\mu \leq \frac{3}{4 \cdot (M-1)} + \frac{3}{4 \cdot M} \quad (\text{A.18})$$

As there exists an instance for which the expected competitive ratio of **TSS** can not be more than $\frac{3}{4 \cdot (M-1)} + \frac{3}{4 \cdot M}$, we can say that $\frac{3}{4 \cdot (M-1)} + \frac{3}{4 \cdot M}$ is the upper bound on the expected competitive ratio of **TSS**. ■

Appendix B

Supplementary material for the Chapter 4

B.1 Neural Network and Training Specifics

Location embeddings – Generated separately using a two-layer neural network that attempts to estimate the travel times between two locations. As locations are part of a street network, instead of this, node2vec or any other similar method can be used to learn the embedding for each location. $h(l)$ is used to denote the embedding of location l .

Inputs: As mentioned in chapter 4, the state of server i is represented using tuple $\langle p^i, t, L^i \rangle$, where p^i denotes the current location of server, t denotes the current time and L^i denotes the ordered list of future locations (pickup/dropoff location of currently assigned requests) along with the cut off time by which the location should be visited. Therefore, $L^i = [(l_1, t_1), (l_2, t_2), \dots, (l_n, t_n)]$ where l_1, l_2, \dots denotes the list of locations and t_1, t_2, \dots denotes the corresponding cutoff times. The cutoff time for visiting current location is t . We use the location embeddings of each location generate following input.

$$A^i = [(h(p^i), t), (h(l_1), t_1), (h(l_2), t_2), \dots, (h(l_n), t_n)].$$

Additionally, we add information about the current decision epoch (t), the num-

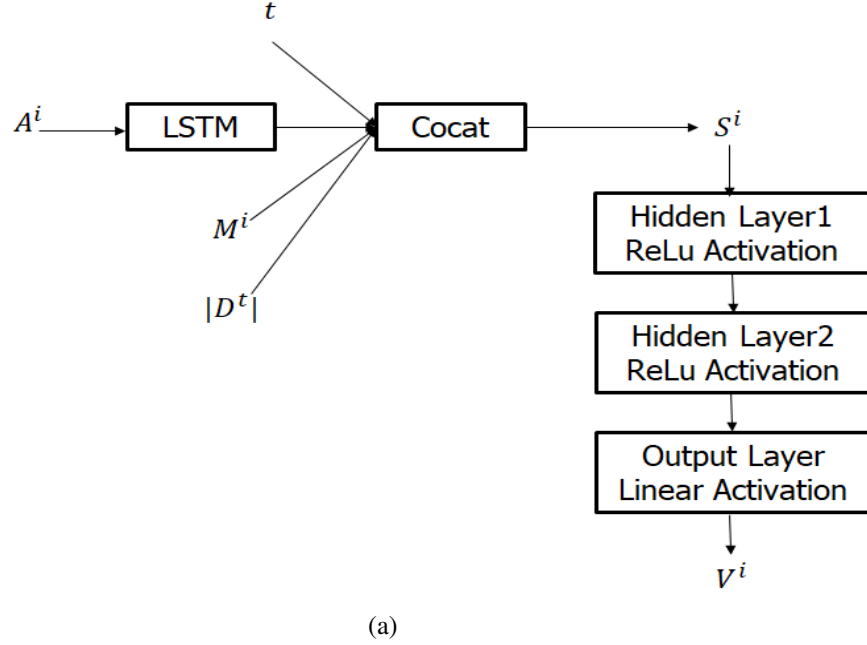


Figure B.1: Neural Network Architecture

ber of servers in the vicinity of server i (M^i) and the total number of requests that arrived in the epoch \mathcal{D}^t . Due to the constraint that a single request can only be assigned to a single server, multiple agents compete for the same request. As a consequence, the value of being in a given state is dependent on the competition it faces from other agents when it is in that state. Adding the information about other servers and the number of current requests stabilizes learning significantly. The complete architecture of the neural network which is used to predict the V-value is shown in the Figure B.1a. The loss considered is the mean squared error and it is minimized using the Adam optimizer using default initial parameters. We need to explore despite having deterministic transition and reward functions because the action space, in our model, is stochastic.

This value function over individual servers is learned offline. When the approach is running online, we compute the assignment (of customer requests to servers) that maximizes the value function computed in the offline phase.

B.2 Details of Baseline algorithm

There are some differences in our implementation of the approach by (Alonso-Mora, Samaranayake, et al., 2017). We refer to our implementation as Baseline and the approach by Alonso *et.al.* as Alonso approach. The differences are highlighted in table B.1. The difference in generation of feasible trips is due to following practical considerations

1. Training time: To effectively train RL algorithms, we need a large number of experiences. To generate samples to train from, we must run our approach for some training days. Using (Alonso-Mora, Samaranayake, et al., 2017)’s strategy for generating feasible trips takes significantly longer than the modification we propose. Given that our training time already takes multiple days, this is not viable. During test time, we maintain the same strategy to ensure coherence with what our value function is trained on.
2. Limitation due to academic computational resources: Our problem is completely parallelisable across different servers and so, in commercial set-ups, the consideration above would not stay relevant. In our case, however, we are bound by academic infrastructure.

This is not a limitation for our approach. We expect the results of both the baseline and our approach improve proportionally if the feasible trips are generated as proposed in the paper by (Alonso-Mora, Samaranayake, et al., 2017).

B.2.1 Rebalancing

Rebalancing empty servers has a significant impact on the number of requests served (Wallar, Van Der Zee, Alonso-Mora, & Rus, 2018). Similar to (Alonso-Mora, Samaranayake, et al., 2017), we perform a re-balancing of unassigned servers to high demand areas after each batch assignment. But unlike them we do not perform rebalancing by using only current unserved requests. This is because by using

Alonso approach	Baseline
Generation of Feasible Trips	
<p>1. Generate RV graph by checking feasibility of each request with each server. Keep only 30 closest servers for each request.</p> <p>2. Perform exhaustive search for up to 4 requests (in the server currently and in the proposed trip). For more requests, check if the request can be inserted into the current server path.</p> <p>3. The exploration of feasible trips for a server is stopped when a time limit of 0.2 seconds is reached.</p>	<p>1. Same</p> <p>2. Insert request into current server path, irrespective of number of requests, for faster computation.</p> <p>3. Exploration is stopped when feasibility constraints are evaluated 150 times. This is done to make the performance independent of the processing speed.</p>
Rebalancing Strategy	
Number of servers rebalanced is min(unassigned servers, unassigned requests).	All unassigned servers are rebalanced

Table B.1: Differences between Baseline and Alonso Approach

RebalanceServers(t):	
min	$\sum_{j \in \mathcal{D}^t} \sum_{i \in \mathcal{V}_u^t} \mathcal{T}(p_i, o_j) \cdot m_{ij}^t$ (B.1)
<i>subject to</i>	$\sum_{i \in \mathcal{V}_u^t} m_{ij}^t \leq n_j^t \quad \forall j \in \mathcal{D}^t$ (B.2)
	$\sum_{j \in \mathcal{D}^t} m_{ij}^t = 1 \quad \forall i \in \mathcal{V}_u^t$ (B.3)
	$0 \leq m_{ij}^t \leq 1 \quad \forall i, j$ (B.4)

Table B.2: Optimization Formulation for Rebalancing unassigned servers

only current unserved requests for rebalancing, number of servers rebalanced will be minimum of unassigned servers and requests leaving majority of servers not being rebalanced in case of low demand scenarios. This means that servers that could be stuck in areas where requests are infrequent. Our approach differs from (Wallar et al., 2018) as we do not use the concept of 'regions' which are disjoint sets of locations. We work with individual locations, instead.

Therefore, we sample $\min(500, |\mathcal{V}|)$ requests from the number of requests seen so far and rebalance all servers to move to the areas of these sampled request by performing the optimization provided in table B.2.

Let \mathcal{V}_u^t denotes the set of unassigned servers at decision epoch t and \mathcal{D}^t denotes the set of sampled customer requests (as described above). m_{ij}^t is a binary variable indicating that server i is moving towards customer request j . The objective of the linear optimization program is to minimize the sum of travel times. We use $\mathcal{T}(p_i, o_j)$ to denote the time taken to travel from initial location p_i of server i to the origin o_j of request j . Constraint B.2 ensures that each server is assigned to exactly one request. Constraint B.3 ensures that each customer request is assigned to exactly n_j^t servers where $n_j^t = \lfloor \frac{|\mathcal{V}_u^t|}{500} \rfloor$ or $\lceil \frac{|\mathcal{V}_u^t|}{500} \rceil$ such that $\sum_{j \in \mathcal{D}^t} n_j^t = |\mathcal{V}_u^t|$.

Appendix C

Supplementary material for the Chapter 8

<p>LPBatch:</p> $\max \sum_{t \in T} \sum_{u \in \mathcal{U}} \sum_{v \in \mathcal{V}} w_{u,v}^t \cdot x_{u,v}^t$ $s.t. \quad \sum_{u \in \mathcal{U}} x_{u,v}^t \leq q_v^t \quad \forall v, t \quad (\text{C.1})$ $\sum_{t' < t} \sum_{v' \in \mathcal{V}} x_{u,v'}^{t'} \cdot Pr[c_{u,v'}^{t'} > t - t'] + \sum_{v \in \mathcal{V}} x_{u,v}^t \leq 1, \forall u, t \quad (\text{C.2})$ $0 \leq x_{u,v}^t \leq 1 \quad \forall u, v, t \quad (\text{C.3})$
--

Table C.1: Optimization Formulation - Unit-Capacity Batch Arrival

C.1 Proof of Proposition 6

Proposition 11. *The optimal value of the LPBatch provides a valid upper bound on the offline optimal value.*

Proof: To show that the optimal value of the LP provides a valid upper bound on the offline optimal value, we prove that the expected value of the optimal matching is less than or equal to the optimal value of the LP. We have two distributions,

first one corresponding to the arrival of vertices (p_v^t) and another corresponding to the number of rounds of unavailability ($c_{u,v}^t$). To prove that the optimal value of LP provides a valid upper bound in the presence of both distributions, similar to earlier works (Sankararaman, 2019), we need to make an assumption that the optimal assignment only depends on the arrival distribution and is independent of the value of $c_{u,v}^t$ ¹.

Consider a realization of arrivals denoted by sequence a . Let $m_v^t(a)$ denotes the number of vertices of type v arriving in round t for arrival sequence a . Similarly, consider a realization of the number of rounds of unavailability denoted by sequence r , where $c_{u,v}^t(r)$ denotes the number of rounds for which server u becomes unavailable on matching with vertex of type v in sequence r . Let $\delta(c_{u,v}^{t'}(r) > t - t')$ is an indicator variable denoting that the number of rounds for which server u becomes unavailable on being assigned to vertex of type v in round t' is greater than $t - t'$ rounds in sequence r . Now, for any arrival sequence a and any realization of the number of rounds of unavailability r , the offline solution can be computed by solving the optimization program in Table C.2².

As mentioned before, we make an assumption that the optimal assignment is independent of the realization of the number of rounds of unavailability, i.e., it only depends on a and not on r . Therefore, we use $x^*(a)$ to denote the optimal solution of the formulation in Table C.2 for sequence a and r . The expected number of times edge (u, v) is matched at t is given by $\sum_a x_{u,v}^{*,t}(a) \cdot P(a)$. To prove that, the LP in Table C.1 is a valid upper bound on the offline optimal value, we show that $\forall u, v, t \sum_a x_{u,v}^{*,t}(a) \cdot P(a)$ is a feasible solution to the LP.

The optimization in Table C.2 is solved for each sequence a and r . As $x^*(a)$ denotes the optimal assignment for the sequence a and r , therefore, $x^*(a)$ satisfies the constraints of the formulation in Table C.2. Hence, we get

¹Instead of this, we can also assume that $c_{u,v}^t$ is a known constant which is a weaker assumption than assuming that the optimal assignment is independent of distribution. The proof is similar for both assumptions.

²We index each of $m_v^t, x_{u,v}^t$ by a and $c_{u,v}^t$ by r to denote the instance for sequence a and r .

$$\sum_{u \in \mathcal{U}} x_{u,v}^{*,t}(a) \leq m_v^t(a) \quad \forall v, t \quad (\text{C.4})$$

$$\sum_{t' < t} \sum_{v' \in \mathcal{V}} x_{u,v'}^{*,t'}(a) \cdot \delta(c_{u,v'}^{t'}(r) > t - t') + \sum_{v \in \mathcal{V}} x_{u,v}^{*,t}(a) \leq 1 \quad \forall u, t \quad (\text{C.5})$$

Multiplying Equations (C.4) and (C.5) by $P(a) \cdot P(r)$ ³, probability of sequence a and r , and performing summation over a and r , we get

$$\sum_{u \in \mathcal{U}} \sum_a \sum_r x_{u,v}^{*,t}(a) \cdot P(a) \cdot P(r) \leq \sum_a \sum_r m_v^t(a) \cdot P(a) \cdot P(r) \quad \forall v, t \quad (\text{C.6})$$

$$\begin{aligned} \sum_{t' < t} \sum_{v' \in \mathcal{V}} \sum_a \sum_r x_{u,v'}^{*,t'}(a) \cdot \delta(c_{u,v'}^{t'}(r) > t - t') \cdot P(a) \cdot P(r) \\ + \sum_{v \in \mathcal{V}} \sum_a \sum_r x_{u,v}^{*,t}(a) \cdot P(a) \cdot P(r) \leq \sum_a \sum_r 1 \cdot P(a) \cdot P(r) \quad \forall u, t \end{aligned} \quad (\text{C.7})$$

$\sum_a \sum_r 1 \cdot P(a) \cdot P(r) = 1$ and $\sum_a \sum_r m_v^t(a) \cdot P(a) \cdot P(r) = \sum_a m_v^t \cdot P(a)$ denotes the expected number of vertices of type v arriving in round t . Therefore,

$$\sum_a m_v^t \cdot P(a) = q_v^t$$

As x^* is independent of r , therefore,

$$\sum_a \sum_r x_{u,v}^{*,t}(a) \cdot P(a) \cdot P(r) = \sum_a x_{u,v}^{*,t}(a) \cdot P(a)$$

On substituting these values, we get,

³ a and r are independently drawn from the distributions.

$\max \sum_{t \in T} \sum_{u \in \mathcal{U}} \sum_{v \in \mathcal{V}} w_{u,v}^t \cdot x_{u,v}^t(a) \tag{C.10}$
$s.t. \sum_{u \in \mathcal{U}} x_{u,v}^t(a) \leq m_v^t(a) \quad \forall v, t \tag{C.11}$
$\sum_{t' < t} \sum_{v' \in \mathcal{V}} x_{u,v'}^{t'}(a) \cdot \delta(c_{u,v'}^{t'}(r) > t - t') + \sum_{v \in \mathcal{V}} x_{u,v}^t(a) \leq 1 \quad \forall u, t \tag{C.12}$
$x_{u,v}^t(a) \in \{0, 1\} \tag{C.13}$

Table C.2: Optimization Formulation - Batch Arrival - For a fixed sequence a and r

$$\sum_{u \in \mathcal{U}} \sum_a x_{u,v}^{*,t}(a) \cdot P(a) \leq q_v^t \quad \forall v, t \tag{C.8}$$

$$\begin{aligned} \sum_{t' < t} \sum_{v' \in \mathcal{V}} \sum_a x_{u,v'}^{*,t'}(a) \cdot P(a) \cdot \sum_r \delta(c_{u,v'}^{t'}(r) > t - t') \cdot P(r) \\ + \sum_{v \in \mathcal{V}} \sum_a x_{u,v}^{*,t}(a) \cdot P(a) \leq 1 \quad \forall u, t \end{aligned} \tag{C.9}$$

As,

$$\sum_r \delta(c_{u,v'}^{t'}(r) > t - t') \cdot P(r) = Pr(c_{u,v'}^{t'} > t - t')$$

Therefore, the Equations (C.8) and (C.9) are same as the constraints of the optimization formulation in Table C.1 with $x_{u,v}^t = \sum_a x_{u,v}^{*,t}(a) \cdot P(a)$. Therefore, $\sum_a x^*(a) \cdot P(a)$ is a feasible solution to the LP in Table C.1. As the value of the optimal solution is greater than or equal to the value of any feasible solution, hence it is proved that the optimal value of LP provides a valid upper bound on the offline optimal value.

C.2 Proof of Proposition 7

Proposition 12. *The online algorithm ADAPBatch is $\frac{1}{2}$ competitive.*

Proof: The proof proceeds in two steps:

1. We first prove that the maximum value of γ for which the assignment rule of ADAPBatch is valid is $\frac{1}{2}$.
2. Next, we prove that the online algorithm is γ competitive. As the maximum value of γ for which the assignment rule is valid is $\frac{1}{2}$, therefore, the online algorithm is $\frac{1}{2}$ competitive.

Computing Maximum value of γ for which the assignment rule is valid:

Since $x_{u,v}^{*,t} \geq 0, \forall u, v, t$, the assignment rule will always generate a positive value. Therefore, the only condition which should be satisfied for the batch assignment rule to be valid is,

$$\frac{x_{u,v}^{*,t} \cdot \gamma}{b^t \cdot p_v^t \cdot \beta_{u,i}^t} \leq 1 \quad \forall u, v, i, t \quad (\text{C.14})$$

Using the expression $q_v^t = b^t \cdot p_v^t$ in Constraint (C.1) of the optimization formulation in Table C.1, we have,

$$\sum_u x_{u,v}^{*,t} \leq b^t \cdot p_v^t \implies x_{u,v}^{*,t} \leq b^t \cdot p_v^t$$

Substituting this in Equation (C.14) and rearranging terms, we get

$$\beta_{u,i}^t \geq \gamma, \forall u, i, t \quad (\text{C.15})$$

Therefore, we focus on finding the value of $\beta_{u,i}^t, \forall u, i, t$. We use mathematical induction to prove that $\beta_{u,i}^t \geq 1 - \gamma, \forall u, i, t$.

Proving $\beta_{u,i}^t \geq 1 - \gamma, \forall u, i, t$: $\beta_{u,i}^t$ denotes the probability that the server u is safe in round t while considering the i^{th} vertex. Initially, as all the servers are available, therefore,

$$\beta_{u,1}^1 = 1 \geq 1 - \gamma, \forall u$$

Please note that $\beta_{u,i}^t \geq \beta_{u,i-1}^t, \forall i > 0$, therefore, we only show the computation for the value of β_{u,b^1}^1 .

$\beta_{u,i}^t$ for any i and t is computed by computing the probability that u is assigned to any v before round t step i such that it has not rejoined the system yet.

$$\beta_{u,b^1}^1 = 1 - \sum_v \sum_{i=1}^{b^1-1} \beta_{u,i}^1 \cdot p_v^1 \cdot \frac{x_{u,v}^{*,1} \cdot \gamma}{b^1 \cdot p_v^1 \cdot \beta_{u,i}^1} = 1 - (b^1-1) \cdot \sum_v \frac{x_{u,v}^{*,1} \cdot \gamma}{b^1} \geq 1 - \gamma (As b^1 \geq 1)$$

As it is valid for $t = 1$, we prove it by induction. Assume that it is valid for all $t' < t$, i.e., in online case the edge (u, v) is matched $x_{u,v}^{*,t} \cdot \gamma$ times in round t .

Therefore,

$$\beta_{u,1}^t = 1 - \sum_v \sum_{t' < t} (x_{u,v}^{*,t'} \cdot \gamma \cdot Pr(c_{u,v}^{t'} > t - t'))$$

From Equation (C.2) of the optimization program in Table C.1, we have

$$\sum_v \sum_{t' < t} (x_{u,v}^{*,t'} \cdot \gamma \cdot Pr(c_{u,v}^{t'} > t - t')) \leq \gamma - \sum_v x_{u,v}^{*,t} \cdot \gamma$$

Multiplying both sides of the above equation by -1, we get

$$-1 \cdot \sum_v \sum_{t' < t} (x_{u,v}^{*,t'} \cdot \gamma \cdot Pr(c_{u,v}^{t'} > t - t')) \geq -1 \cdot (\gamma - \sum_v x_{u,v}^{*,t} \cdot \gamma)$$

Adding 1 on both sides in above equation, we get

$$1 - \sum_v \sum_{t' < t} (x_{u,v}^{*,t'} \cdot \gamma \cdot Pr(c_{u,v}^{t'} > t - t')) \geq 1 - \gamma + \sum_v x_{u,v}^{*,t} \cdot \gamma$$

$$1 - \sum_v \sum_{t' < t} (x_{u,v}^{*,t'} \cdot \gamma \cdot Pr(c_{u,v}^{t'} > t - t')) \geq 1 - \gamma + \sum_v x_{u,v}^{*,t} \cdot \gamma$$

Therefore,

$$\beta_{u,1}^t \geq 1 - \gamma + \sum_v x_{u,v}^{*,t} \cdot \gamma \geq 1 - \gamma$$

Similarly, we compute β_{u,b^t}^t , by computing the probability that u is assigned to any v before round t step i such that it has not rejoined the system yet.

$$\begin{aligned}\beta_{u,b^t}^t &= 1 - \sum_v \sum_{t' < t} (x_{u,v}^{*,t'} \cdot \gamma \cdot Pr(c_{u,v}^{t'} > t - t')) - \sum_v \sum_{i=1}^{b^t-1} \beta_{u,i}^t \cdot p_v^t \cdot \frac{x_{u,v}^{*,t} \cdot \gamma}{b^t \cdot p_v^t \cdot \beta_{u,i}^t} \\ \implies \beta_{u,b^t}^t &\geq 1 - \gamma + \sum_v x_{u,v}^{*,t} \cdot \gamma - \sum_v (b^t - 1) \cdot \frac{x_{u,v}^{*,t} \cdot \gamma}{b^t} \geq 1 - \gamma \quad (\text{C.16})\end{aligned}$$

Therefore, $\beta_{u,i}^t \geq 1 - \gamma, \forall u, i, t$. From Equation (C.15), for assignment rule to be valid $\beta_{u,i}^t \geq \gamma$. Therefore, the maximum possible value of γ is the solution of equation $1 - \gamma = \gamma \implies \gamma = \frac{1}{2}$.

Online Algorithm ADAPBatch is γ competitive: Let $M_{u,v,i}^t$ is an indicator variable denoting that the server u is tried for assignment to vertex of type v in round t while processing i^{th} vertex of the batch. Let $N_{u,i}^t$ is an indicator variable denoting that the server u is available in round t while processing i^{th} vertex of the batch and $O_{v,i}^t$ denotes that the vertex of type v is processed as i^{th} vertex in round t . - Therefore, probability that the server u is assigned to the vertex of type v while processing i^{th} vertex of the batch in round t is given by

$$P[M_{u,v,i}^t = 1] \cdot P[N_{u,i}^t = 1] \cdot P[O_{v,i}^t = 1] = \frac{x_{u,v}^{*,t} \cdot \gamma}{b^t \cdot p_v^t \cdot \beta_{u,i}^t} \cdot \beta_{u,i}^t \cdot p_v^t = \frac{x_{u,v}^{*,t} \cdot \gamma}{b^t}$$

Expected Number of times vertex of type v is matched to u in round $t =$

$$b^t \cdot \frac{x_{u,v}^{*,t} \cdot \gamma}{b^t} = x_{u,v}^{*,t} \cdot \gamma$$

i.e., in online case the edge (u, v) is matched $x_{u,v}^{*,t} \cdot \gamma$ times in round t .

As each edge is made with the probability $x_{u,v}^{*,t} \cdot \gamma$ and the maximum value of γ for which the assignment rule is valid is $\frac{1}{2}$, therefore, using Lemma 1 the competitive ratio of the algorithm is $\gamma = \frac{1}{2}$.

Lemma 1. (Sankararaman, 2019) Let x^* denote the optimal solution to LP in Table C.1. Suppose we have that for every edge (u, v) at any round t , $\Pr[(u, v) \text{ is included in the matching}] \geq \gamma \cdot x_{u,v}^{*,t}$ then the competitive ratio is at least γ .

C.3 Expected Number of times group of type v^g can be formed in round t

Let $q_{v^g}^t$ denote the expected number of times group of type v^g can be formed in round t . As each of the b^t vertices are sampled independently from a categorical distribution p_v^t , $\forall v$, we can consider it as having b^t trials and use $Y_{v,i}$ as an indicator variable denoting that v is sampled in the i^{th} trial (as the i^{th} vertex or not, out of b^t vertices). Please note that $Y_{v,i} \cdot Y_{v',i} = 0, v \neq v'$ and if $i \neq j$, $Y_{v,i}$ and $Y_{v',j}$ are independent.

We first consider a simple case, where $v^g = (v, v'), v \neq v'$. Therefore, if any of two different vertices are of type v and v' then we can form the group of type v^g . So, we have

$$q_{v^g}^t = E \left[\sum_{i,j;i \neq j} Y_{v,i} \cdot Y_{v',j} \right]$$

By linearity of expectation, we get

$$q_{v^g}^t = \sum_{i,j;i \neq j} E[Y_{v,i} \cdot Y_{v',j}] \tag{C.17}$$

As each vertex is independently sampled, the event of sampling i^{th} vertex and

j^{th} vertex are independent of each other. Therefore,

$$\begin{aligned}
 q_{v^g}^t &= \sum_{i,j;i \neq j} E[Y_{v,i}^t] \cdot E[Y_{v',j}^t] \\
 E[Y_{v,i}^t] &= p_v^t \\
 E[Y_{v',j}^t] &= p_{v'}^t \\
 q_{v^g}^t &= \sum_{i,j;i \neq j} p_v^t \cdot p_{v'}^t \\
 q_{v^g}^t &= b^t \cdot (b^t - 1) p_v^t \cdot p_{v'}^t
 \end{aligned}$$

Similarly if we have $v^g = (v, v)$, we can form the group if any of the two vertices are of type v

$$q_{v^g}^t = E \left[\sum_{i,j;i < j} Y_{v,i} \cdot Y_{v,j} \right]$$

By linearity of expectation

$$q_{v^g}^t = \sum_{i,j;i < j} E[Y_{v,i} \cdot Y_{v,j}]$$

As each vertex is independently sampled, event of sampling i^{th} vertex and j^{th} vertex are independent. Therefore,

$$\begin{aligned}
 q_{v^g}^t &= \sum_{i,j;i < j} E[Y_{v,i}^t] \cdot E[Y_{v,j}^t] \\
 E[Y_{v,i}^t] &= p_v^t \\
 q_{v^g}^t &= \sum_{i,j;i < j} p_v^t \cdot p_v^t \\
 q_{v^g}^t &= \frac{b^t \cdot (b^t - 1)}{2} \cdot p_v^t \cdot p_v^t
 \end{aligned}$$

Now extending the above reasoning to any group of type v^g , a group of type v^g can be formed if we have n_{v,v^g} vertices of type v for each $v \in v^g$.

$$q_{v^g}^t = E \left[\prod_{v \in v^g} \sum_{i_1^v, i_2^v, \dots, i_{n_{v,v^g}}^v} Y_{v, i_1^v} \cdot Y_{v, i_2^v} \cdot \dots \cdot Y_{v, i_{n_{v,v^g}}^v} \right] \quad (\text{C.18})$$

In the above expression, $i_l^v < i_m^v$ if $m > l$ (Please refer to above derivation for simple case when $v^g = (v, v)$.)

When $i_l^v = i_m^{v'}$ the product of Y_{v, i_l^v} and $Y_{v', i_m^{v'}}$ will be 0. Also $E[Y_{v, i_l^v}] = p_v^t, \forall l$, therefore, the expression in Equation (C.18) is equivalent to

$$WS \cdot \prod_{v \in v^g} (p_v^t)^{n_{v,v^g}}$$

where WS denotes the number of ways to select the i_l^v indices in Equation (C.18) such that the the indicator variables multiplied provide a value 1. Therefore, if group of type $v^g = (v_1, v_2, \dots, v_k)$ then this is equivalent to selecting n_{v_1, v^g} vertices of type v_1 from b^t followed by selecting n_{v_2, v^g} vertices from $b^t - n_{v_1, v^g}$ vertices and so on. Therefore, WS for $v^g = (v_1, v_2, \dots, v_k)$ is computed as follows

$$WS = \frac{(b^t)!}{(b^t - n_{v_1, v^g})! \cdot (n_{v_1, v^g})!} \cdot \frac{(b^t - n_{v_1, v^g})!}{(b^t - n_{v_1, v^g} - n_{v_2, v^g})! \cdot (n_{v_2, v^g})!} \dots \frac{(b^t - \sum_{i=1}^{k-1} n_{v_i, v^g})!}{(n_{v_k, v^g})! \cdot (b^t - \sum_{i=1}^k n_{v_i, v^g})!} \quad (\text{C.19})$$

$$WS = \frac{\prod_{i=0}^{|v^g|} (b^t - i)}{\prod_{i=1}^k (n_{v_i, v^g})!}$$

Now for any v^g WS can be written as follows

$$WS = \frac{\prod_{i=0}^{|v^g|} (b^t - i)}{\prod_{v \in v^g} (n_{v, v^g})!} \quad (\text{C.20})$$

Therefore, $q_{v^g} = \frac{\prod_{i=0}^{|v^g|} (b^t - i)}{\prod_{v \in v^g} (n_{v, v^g})!} \cdot \prod_{v \in v^g} (p_v^t)^{n_{v, v^g}}$

LPSHare:

$$\max \sum_{t \in T} \sum_{u \in \mathcal{U}} \sum_{v^g \in \mathcal{V}^g} w_{u,v^g}^t \cdot x_{u,v^g}^t \quad (\text{C.21})$$

$$\begin{aligned} \text{s.t.} \quad & \sum_{t' < t} \sum_{v^{g'} \in \mathcal{V}^{g'}} x_{u,v^{g'}}^{t'} \cdot Pr[c_{u,v^{g'}}^{t'} > t - t'] + \\ & + \sum_{v^g \in \mathcal{V}^g} x_{u,v^g}^t \leq 1 \quad \forall u, t \end{aligned} \quad (\text{C.22})$$

$$\sum_{v^g; v \in v^g} \sum_{u \in \mathcal{U}} n_{v,v^g} \cdot x_{u,v^g}^t \leq q_v^t \quad \forall v, t \quad (\text{C.23})$$

$$\sum_{u \in \mathcal{U}} x_{u,v^g}^t \leq q_{v^g}^t \quad \forall v^g, t \quad (\text{C.24})$$

$$0 \leq x_{u,v^g}^t \leq 1 \quad \forall u, v^g, t \quad (\text{C.25})$$

Table C.3: Optimization Formulation - Multi-Capacity

C.4 Details about the Optimization Formulation

LPSHare

Let the number of vertices of type v available in round t is m_v^t . We use $m_{v^g}^t$ to denote the number of groups of type v^g which can be formed in round t . n_{v,v^g} denotes the number of vertices of type v present in the group of type v^g . Let x_{u,v^g}^t denotes the assignment of server u to the group of type v^g . Let y_{v,v^g}^t denotes the flow on the edge from v to v^g where v is a part of the group of type v^g . Therefore, we will have following flow preservation constraints:

$$\sum_{t'} \sum_{v^{g'} \in \mathcal{V}^{g'}} x_{u,v^{g'}}^{t'} \cdot Pr(c_{u,v^{g'}}^{t'} > t - t') + \sum_{v^g \in \mathcal{V}^g} x_{u,v^g}^t \leq 1 \quad \forall u, t \quad (\text{C.26})$$

$$\sum_{u \in \mathcal{U}} x_{u,v^g}^t \leq m_{v^g}^t \quad \forall v^g, t \quad (\text{C.27})$$

$$\sum_{v^g; v \in v^g} y_{v,v^g}^t \cdot n_{v,v^g} \leq m_v^t \quad \forall v, t \quad (\text{C.28})$$

$$\sum_{v \in v^g} n_{v,v^g} \cdot y_{v,v^g}^t \leq \text{sizeof}(v^g) \cdot m_{v^g}^t \quad \forall v^g, t \quad (\text{C.29})$$

$$y_{v,v^g}^t = \sum_{u \in \mathcal{U}} x_{u,v^g}^t \quad \forall v^g; v \in v^g, t \quad (\text{C.30})$$

The Equation (C.30) is the conditional equal flow constraint which states that the total incoming flow to a group will be equal to the total outgoing flow to each of the vertex which is a part of the group.

Substituting Equation (C.30) in Equation (C.28), we get

$$\sum_{v^g; v \in v^g} \sum_{u \in \mathcal{U}} x_{u,v^g}^t \leq m_v^t \quad \forall v, t \quad (\text{C.31})$$

As $m_{v^g}^t \geq \min(m_v^t)$, if m_v^t is an integer, therefore, Equation (C.27) is redundant in the presence of Equation (C.31).

Similarly on substituting Equation (C.30) in Equation (C.29), we get,

$$\sum_{v \in v^g} n_{v,v^g} \sum_{u \in \mathcal{U}} x_{u,v^g}^t \leq \text{sizeof}(v^g) \cdot m_{v^g}^t \quad \forall v^g, t \quad (\text{C.32})$$

$$\sum_{u \in \mathcal{U}} x_{u,v^g}^t \leq m_{v^g}^t \quad \forall v^g, t \quad (\text{C.33})$$

This is same as Equation (C.27) which is redundant.

Therefore, we get the optimization formulation provided in Table C.4. Please note that we keep the redundant constraint in the optimization formulation.

We replace $m_{v^g}^t$ and m_v^t by $q_{v^g}^t$ and q_v^t which denote the expected number of groups/vertices. The equations remains same as Equations (C.26) - (C.30).

But unlike earlier case, we can not say that Equation (C.27) is redundant in presence of Equation (C.31).

This is because q_v^t can lie between 0 and 1.

Therefore, we get the optimization formulation presented in Table C.3.

C.5 Proof of Proposition 8

Proposition 13. *The optimal value of the LPShare provides a valid upper bound on the offline optimal value.*

Proof: The proof is similar to the proof of Proposition 6.

To show that the LP provides a valid upper bound on the offline optimal solution, we prove that the expected value of matching is less than or equal to the optimal solution of LP. We have two distributions, one corresponding to the arrival of vertices (p_v^t) and another corresponding to the number of rounds of unavailability (c_{u,v^g}^t). To prove that the optimal value of the LP provides a valid upper bound in presence of both distributions, similar to earlier works (Sankararaman, 2019), we need to make an assumption that the optimal assignment only depends on the arrival distribution and is independent of the value of c_{u,v^g}^t .⁴

Consider a realization of arrivals denoted by sequence a . Let $m_v^t(a)$ denotes the number of vertices of type v arriving in round t for arrival sequence a and $m_{v^g}^t(a)$ denotes the number of groups of type v^g which can be formed in round t in the arrival sequence a . Similarly, consider a realization of the number of rounds of unavailability denoted by sequence r , where $c_{u,v^g}^t(r)$ denotes the number of rounds for which server u becomes unavailable on matching with group of type v^g in sequence r . $\delta(c_{u,v^g}^t(r) > t - t')$ is an indicator variable denoting that the number of rounds for which server u becomes unavailable on being assigned to group of type v^g in round t' is greater than $t - t'$. Now, for any arrival sequence a and any realization of the number of rounds of unavailability r , the offline solution can be computed by solving the optimization program in Table C.4⁵.

As mentioned before, we make an assumption that the optimal assignment is independent of the realization of the number of rounds of unavailability, i.e., it only depends on a and not r . Therefore, we use $x^*(a)$ to denote the optimal solution of the formulation in Table C.4 for sequence a and r . The expected number of times (u, v^g) is matched at t is given by $\sum_a x_{u,v^g}^{*,t}(a) \cdot P(a)$. To prove that, the optimal value of LP in Table C.3 is a valid upper bound on the optimal solution, We show that $\forall u, v^g, t \sum_a x_{u,v^g}^{*,t}(a) \cdot P(a)$ is a feasible solution to the LP.

⁴Instead of this, we can also assume that c_{u,v^g}^t is a known constant which is a weaker assumption than assuming that the optimal assignment is independent of distribution of the number of rounds of unavailability. The proof is similar for both assumptions.

⁵We index each of $m_v^t, m_{v^g}^t, x_{u,v}^t$ by a and c_{u,v^g}^t by r to denote the instance for sequence a and r .

The optimization in Table C.4 is solved for each sequence a and r . As we used $x^*(a)$ to denote the optimal solution for the sequence a and r , therefore, $x^*(a)$ satisfies the constraints of formulation in Table C.4. Hence, we get

$$\sum_{t' < t} \sum_{v^{g'} \in \mathcal{V}^g} x_{u, v^{g'}}^{*, t'}(a) \cdot \delta(c_{u, v^{g'}}^{t'}(r) > t - t') + \sum_{v^g \in \mathcal{V}^g} x_{u, v^g}^{*, t}(a) \leq 1 \quad \forall u, t \quad (\text{C.34})$$

$$\sum_{v^g; v \in v^g} \sum_{u \in \mathcal{U}} n_{v, v^g} \cdot x_{u, v^g}^{*, t}(a) \leq m_v^t(a) \quad \forall v, t \quad (\text{C.35})$$

$$\sum_{u \in \mathcal{U}} x_{u, v^g}^{*, t}(a) \leq m_{v^g}^t(a) \quad \forall v^g, t \quad (\text{C.36})$$

Multiplying Equations (C.34),(C.35) and (C.36) by $P(a) \cdot P(r)$, probability of sequence a and r , and performing summation, we get

$$\begin{aligned} \sum_a \sum_r \sum_{t' < t} \sum_{v^{g'} \in \mathcal{V}^g} x_{u, v^{g'}}^{*, t'}(a) \cdot \delta(c_{u, v^{g'}}^{t'}(r) > t - t') \cdot P(a) \cdot P(r) + \\ \sum_a \sum_r \sum_{v^g \in \mathcal{V}^g} x_{u, v^g}^{*, t}(a) \cdot P(a) \cdot P(r) \leq \sum_a \sum_r 1 \cdot P(a) \cdot P(r) \quad \forall u, t \end{aligned} \quad (\text{C.37})$$

$$\begin{aligned} \sum_a \sum_r \sum_{v^g; v \in v^g} \sum_{u \in \mathcal{U}} n_{v, v^g} \cdot x_{u, v^g}^{*, t}(a) \cdot P(a) \cdot P(r) \leq \\ \sum_a \sum_r m_v^t(a) \cdot P(a) \cdot P(r) \quad \forall v, t \end{aligned} \quad (\text{C.38})$$

$$\sum_a \sum_r \sum_{u \in \mathcal{U}} x_{u, v^g}^{*, t}(a) \cdot P(a) \cdot P(r) \leq \sum_a \sum_r m_{v^g}^t(a) \cdot P(a) \cdot P(r) \quad \forall v^g \quad (\text{C.39})$$

$\sum_a \sum_r 1 \cdot P(a) \cdot P(r) = 1$ and $\sum_a \sum_r m_v^t(a) \cdot P(a) \cdot P(r) = \sum_a m_v^t(a) \cdot P(a)$ denotes the expected number of vertices of type v arriving in round t and $\sum_a \sum_r m_{v^g}^t(a) \cdot P(a) \cdot P(r) = \sum_a m_{v^g}^t(a) \cdot P(a)$ denotes the expected number of groups of type v^g formed in round t . Therefore,

$$\sum_a m_v^t(a) \cdot P(a) = q_v^t$$

$$\sum_a m_{v^g}^t(a) \cdot P(a) = q_{v^g}^t$$

Also, as x^* is independent of r , therefore,

$$\sum_a \sum_r x_{u,v^g}^{*,t}(a) \cdot P(a) \cdot P(r) = \sum_a x_{u,v^g}^{*,t}(a) \cdot P(a)$$

On substituting these values, we get,

$$\begin{aligned} & \sum_a \sum_{t' < t} \sum_{v^{g'} \in \mathcal{V}^g} x_{u,v^{g'}}^{*,t}(a) \cdot P(a) \sum_r \delta(c_{u,v^{g'}}^{t'}(r) > t - t') \cdot P(r) \\ & + \sum_a \sum_{v^g \in \mathcal{V}^g} x_{u,v^g}^{*,t}(a) \cdot P(a) \leq 1 \quad \forall u, t \end{aligned} \quad (\text{C.40})$$

$$\sum_a \sum_{v^g; v \in v^g} \sum_{u \in \mathcal{U}} n_{v,v^g} \cdot x_{u,v^g}^{*,t}(a) \cdot P(a) \leq q_v^t \quad \forall v, t \quad (\text{C.41})$$

$$\sum_a \sum_{u \in \mathcal{U}} x_{u,v^g}^{*,t}(a) \cdot P(a) \leq q_{v^g}^t \quad \forall v^g, t \quad (\text{C.42})$$

As,

$$\sum_r \delta(c_{u,v^{g'}}^{t'}(r) > t - t') \cdot P(r) = Pr(c_{u,v^{g'}}^{t'} > t - t')$$

On substituting these values, Equations (C.40),(C.41) and (C.42) become similar to the constraints of the optimization formulation in Table C.3 with $x_{u,v^g}^t = \sum_a x_{u,v^g}^{*,t}(a) \cdot P(a)$. Therefore, $\sum_a x^*(a) \cdot P(a)$ is a feasible solution to LP in Table C.3. Hence it is proved that the LP provides a valid upper bound on the optimal solution.

$$\begin{aligned}
\max \quad & \sum_{t \in T} \sum_{u \in \mathcal{U}} \sum_{v^g \in \mathcal{V}^g} w_{u,v^g}^t \cdot x_{u,v^g}^t(a) & (C.43) \\
s.t. \quad & \sum_{t' < t} \sum_{v^{g'} \in \mathcal{V}^{g'}} x_{u,v^{g'}}^{t'}(a) \cdot \delta(c_{u,v^{g'}}^{t'}(r) > t - t') + \sum_{v^g \in \mathcal{V}^g} x_{u,v^g}^t(a) \leq 1 \quad \forall u, t & (C.44) \\
& \sum_{v^g; v \in v^g} \sum_{u \in \mathcal{U}} n_{v,v^g} \cdot x_{u,v^g}^t(a) \leq m_v^t(a) \quad \forall v, t & (C.45) \\
& \sum_{u \in \mathcal{U}} x_{u,v^g}^t(a) \leq m_{v^g}^t(a) \quad \forall v^g & (C.46) \\
& x_{u,v^g}^t(a) \in \{0, 1\} \quad \forall u, v^g, t & (C.47)
\end{aligned}$$

Table C.4: Optimization Formulation - Multi-Capacity - For a fixed sequence a and r

C.6 Example showing the groups considered at different steps

Example 3. Suppose $\mathcal{V} = \{v_1, v_2\}$, $b^t = 3$. Out of the three incoming vertices - two vertices are of type v_1 and one vertex is of type v_2 . To distinguish between two vertices of type v_1 , we refer them by $v_1(1)$ and $v_1(2)$. On random shuffling of these three vertices, they are present in the following order:

$$\text{Sequence} : (v_1(1), v_2, v_1(2))$$

Step (1, 2) represent that the first and second vertex in the above sequence is considered. We define an ordering over types of vertices. In this example, let $v_1 \prec v_2$. So, whenever we are considering group formed with these two types of vertices, we will always consider (v_1, v_2) and not (v_2, v_1) . Therefore, in this example, we will consider the group at (1, 2) and (3, 2) not at (2, 1) or (2, 3). This ensures that we are processing each group only once.

Step	v^g	Step	v^g	Step	v^g
(1,1)	$v_1(1)$	(2,1)	$(v_2, v_1(1))$	(3,1)	$(v_1(2), v_1(1))$
(1,2)	$(v_1(1), v_2)$	(2,2)	v_2	(3,2)	$(v_1(2), v_2)$
(1,3)	$(v_1(1), v_1(2))$	(2,3)	$(v_2, v_1(2))$	(3,3)	$v_1(2)$

In another arrival sequence, two vertices are of type v_2 and one vertex is of type

v_1 . To distinguish between these two vertices of type v_2 , we refer to them by $v_2(1)$ and $v_2(2)$. On random shuffling of these three vertices, they are present in following order:

$$\text{Sequence} : (v_2(1), v_1, v_2(2))$$

In this case the groups will be processed as shown in the below table,

Step	v^g	Step	v^g	Step	v^g
(1,1)	$v_2(1)$	(2,1)	$(v_1, v_2(1))$	(3,1)	$(v_2(2), v_2(1))$
(1,2)	$(v_2(1), v_1)$	(2,2)	v_1	(3,2)	$(v_2(2), v_1)$
(1,3)	$v_2(1), v_2(2)$	(2,3)	$(v_1, v_2(2))$	(3,3)	$v_2(2)$

So across these 2 sequences, we can see that the group of type (v_1, v_2) is processed at 4 places: $(1, 2)(2, 1), (2, 3)(3, 2)$ – Similarly if we create more sequences, we will observe that the group of type (v_1, v_2) can be considered at 6 places (all places except $(1, 1)(2, 2)(3, 3)$).

C.7 Complete proof of Proposition 9

Proof of Proposition 9: We now show the detailed steps of finding the maximum value of $\frac{\prod_{v \in v^g} (p_v^t)^{n_{v, v^g}}}{P_{v^g, (i, j)}^t}$ used in the proof. For any values of i, j and t , we can compute $P_{v^g, (i, j)}^t$ in terms of the probabilities of individual vertices in the group being available. Let $P_{v, i, (i, j)}^t$ denotes the probability that the vertex of type v with label i is available in round t at step (i, j) . Similarly, $P_{v, j, (i, j)}^t$ denotes the probability that the vertex of type v with label j is available in round t at step (i, j) . Since the two vertices with labels i and j are independent, the probability that both of them are available for assignment at t is just a product of the two vertices being available. Therefore, probability that a group of type v^g is available in round t at step (i, j) is:

$$P_{v^g, (i, j)}^t = \begin{cases} P_{v, i, (i, j)}^t \cdot P_{v', j, (i, j)}^t & \text{where } v^g = (v, v'), i \neq j, \\ P_{v, i, (i, j)}^t \cdot P_{v, j, (i, j)}^t & \text{where } v^g = (v, v), i \neq j, \\ P_{v, i, (i, j)}^t & \text{where } v^g = (v), i = j \end{cases} \quad (\text{C.48})$$

$P_{v,i,(1,1)}^t$ and $P_{v,j,(1,1)}^t$ will be equal to $p_v^t \forall v, t, i, j$. From Equation (C.48), we can see that to compute $P_{v^g,(i,j)}^t$, we need to compute $P_{v,i,(i,j)}^t$ and $P_{v,j,(i,j)}^t \forall v, t, i, j$. If $P_{v,i,(i,j)}^{t,-}$ denotes the probability that the vertex with label i is assigned prior to step (i, j) then

$$P_{v,i,(i,j)}^t = p_v^t - P_{v,i,(i,j)}^{t,-} \quad (\text{C.49})$$

Using the assignment rule in Equation (8.8) and considering the steps where vertex of type v with label i is used, we get,

$$P_{v,i,(i,j)}^t = p_v^t - P_{v,i,(i,j)}^{t,-} \quad (\text{C.50})$$

$$\begin{aligned} P_{v,i,(i,j)}^t &= p_v^t \\ &- \sum_{u \in \mathcal{U}} \sum_{v^g; v \in v^g} \sum_{j'=1}^{j-1} \frac{x_{u,v^g}^t \cdot \gamma}{h_{v^g}^t \cdot P_{v^g,(i,j')}^t \cdot \beta_{u,(i,j')}^t} \cdot P_{v^g,(i,j')}^t \cdot \beta_{u,(i,j')}^t \cdot s_{v,v^g,i,(i,j')} \\ &- \sum_{u \in \mathcal{U}} \sum_{v^g; v \in v^g} \sum_{i'=1}^{i-1} \frac{x_{u,v^g}^t \cdot \gamma}{h_{v^g}^t \cdot P_{v^g,(i',j)}^t \cdot \beta_{u,(i',j)}^t} \cdot P_{v^g,(i',j)}^t \cdot \beta_{u,(i',j)}^t \cdot s_{v,v^g,i,(i',j)} \end{aligned} \quad (\text{C.51})$$

where, $s_{v,v^g,i,(i,j')}$ is a binary constant denoting that the vertex $v \in v^g$ is considered as a vertex with label i in step (i, j') or not. Please note that while the group of type v^g will be considered at $h_{v^g}^t$ steps, when we are computing $P_{v,i,(i,j)}^t$, we will not be considering all $h_{v^g}^t$ steps but only the steps where v^g is formed with v as a vertex with label i . This is because we are computing the probability of vertex of type v with label i being available at step (i, j) . Therefore, in the Equation (C.51), if $|v^g| = 2$, at most $b^t - 1$ steps will affect the computation of $P_{v,i,(i,j)}^t$ and if $|v^g| = 1$ then only one step will affect the computation⁶. Let $e_{v^g,i,(i,j)}^t$ denote the maximum number of steps (for group of type v^g) which can affect the computation of $P_{v,i,(i,j)}^t$, then

$$P_{v,i,(i,j)}^t \geq p_v^t - \sum_u \sum_{v^g, v \in v^g} \frac{x_{u,v^g}^t \cdot \gamma \cdot e_{v^g,i,(i,j)}^t}{h_{v^g}^t} \quad (\text{C.52})$$

⁶Please refer to the example present in the Section C.6 for more clarity

For $\kappa = 2$, we have

$$\frac{e^{t}_{v^g, v, i, (i, j)}}{h_{v^g}^t} = \begin{cases} \frac{2}{b^t} & \text{if } |v^g| = 2 \text{ and } v^g = (v, v) \\ \frac{1}{b^t} & \text{if } |v^g| = 2 \text{ and } v^g = (v, v') \\ \frac{1}{b^t} & \text{if } |v^g| = 1 \text{ and } v^g = (v) \end{cases}$$

Therefore, $\frac{e^{t}_{v^g, v, i, (i, j)}}{h_{v^g}^t} = \frac{n_{v, v^g}}{b^t}$. Substituting this in Equation (C.52), we get

$$P_{v, i, (i, j)}^t \geq p_v^t - \sum_u \sum_{v^g; v \in v^g} \frac{x_{u, v^g}^t \cdot \gamma \cdot n_{v, v^g}}{b^t} \quad (\text{C.53})$$

From the Constraint (8.5) in the optimization program of Table 8.1, we have,

$$\sum_{v^g; v \in v^g} \sum_{u \in \mathcal{U}} n_{v, v^g} \cdot x_{u, v^g}^t \leq b^t \cdot p_v^t. \text{ Therefore,}$$

$$P_{v, i, (i, j)}^t \geq p_v^t - \gamma \cdot p_v^t \implies P_{v, i, (i, j)}^t \geq (1 - \gamma) \cdot p_v^t \quad (\text{C.54})$$

Similarly, we can show that $P_{v, j, (i, j)}^t \geq (1 - \gamma) \cdot p_v^t$. Substituting (C.54) in Equation (C.48),

$$P_{v^g, (i, j)}^t \geq \begin{cases} (1 - \gamma)^2 \cdot \prod_{v \in v^g} (p_v^t)^{n_{v, v^g}} & \text{if } |v^g| = 2 \\ (1 - \gamma) \cdot \prod_{v \in v^g} (p_v^t)^{n_{v, v^g}} & \text{if } |v^g| = 1 \end{cases} \quad (\text{C.55})$$

Rearranging terms, we get ⁷

$$\frac{\prod_{v \in v^g} (p_v^t)^{n_{v, v^g}}}{P_{v^g, (i, j)}^t} \leq \frac{1}{(1 - \gamma)^2}, \forall v^g, t, i, j \quad (\text{C.56})$$

This is because $\frac{1}{1 - \gamma} \leq \frac{1}{(1 - \gamma)^2}$. Equation (C.56) is same as the Equation (8.11) used in the proof of Proposition 9 .

Proving $\beta_{u, (i, j)}^t \geq 1 - \gamma$:

To prove that $\beta_{u, (i, j)}^t \geq 1 - \gamma$, we use mathematical induction, at $t = 1$, initially

⁷As all the above computation is valid for all value of i, j and t

all u are available, therefore,

$$\beta_{u,(1,1)}^1 = 1$$

Please note that $\beta_{u,(i,j)}^t$ keeps on decreasing as i, j increases for fixed u and t , therefore, we only show for the value of $\beta_{u,(b^1,b^1)}^1$.

$$\begin{aligned} \beta_{u,(b^1,b^1)}^1 &= 1 - \sum_{v^g} \sum_{i=1}^{b^1-1} \sum_{j=1}^{b^1} \frac{x_{u,v^g}^{*,1} \cdot \gamma}{h_{v^g}^1 \cdot P_{v^g,(i,j)}^1 \cdot \beta_{u,(i,j)}^1} \cdot P_{v^g,(i,j)}^1 \cdot \beta_{u,(i,j)}^1 \\ &\quad - \sum_{v^g} \sum_{j=1}^{b^1-1} \frac{x_{u,v^g}^{*,1} \cdot \gamma}{h_{v^g}^1 \cdot P_{v^g,(i,j)}^1 \cdot \beta_{u,(i,j)}^1} \cdot P_{v^g,(i,j)}^1 \cdot \beta_{u,(i,j)}^1 \end{aligned} \quad (\text{C.57})$$

As mentioned before in ADAPShare- κ description, each group will be considered for assignment at $h_{v^g}^t$ steps, and at step (b^t, b^t) , a single vertex will be considered, therefore,

$$\beta_{u,(b^1,b^1)}^1 \geq 1 - \sum_{v^g} (x_{u,v^g}^{*,1} \cdot \gamma) \quad (\text{C.58})$$

As maximum value of $\sum_{v^g} x_{u,v^g}^{*,t}$ is 1. Therefore, $\beta_{u,(b^1,b^1)}^1 \geq 1 - \gamma$.

As it is valid for $t = 1$, we prove it by induction. Assume that it is valid for all $t' < t$, i.e., in online case the server u is matched $x_{u,v^g}^{*,t} \cdot \gamma$ times in round t to group of type v^g . Therefore,

$$\beta_{u,(1,1)}^t = 1 - \sum_{v^{g'}} \sum_{t' < t} x_{u,v^{g'}}^{*,t'} \cdot \gamma \cdot Pr(c_{u,v^{g'}}^{t'} \geq t - t') \quad (\text{C.59})$$

From Equation (C.22) of the optimization program in Table C.3, we have

$$\sum_{v^{g'}} \sum_{t' < t} (x_{u,v^{g'}}^{*,t'} \cdot \gamma \cdot Pr(c_{u,v^{g'}}^{t'} > t - t')) \leq \gamma - \sum_{v^g} x_{u,v^g}^{*,t} \cdot \gamma$$

Multiplying both sides of the above equation by -1, we get

$$-1 \cdot \sum_{v^{g'}} \sum_{t' < t} (x_{u,v^{g'}}^{*,t'} \cdot \gamma \cdot Pr(c_{u,v^{g'}}^{t'} > t - t')) \geq -1 \cdot (\gamma - \sum_{v^g} x_{u,v^g}^{*,t} \cdot \gamma)$$

Adding 1 on both sides in above equation, we get

$$1 - \sum_{v^{g'}} \sum_{t' < t} (x_{u,v^{g'}}^{*,t'} \cdot \gamma \cdot Pr(c_{u,v^{g'}}^{t'} > t - t')) \geq 1 - \gamma + \sum_{v^g} x_{u,v^g}^{*,t} \cdot \gamma$$

Therefore,

$$\beta_{u,(1,1)}^t \geq 1 - \gamma + \sum_{v^g} x_{u,v^g}^{*,t} \cdot \gamma \geq 1 - \gamma$$

$$\begin{aligned} \beta_{u,(b^t,b^t)}^t &= \beta_{u,(1,1)}^t - \sum_{v^g} \sum_{i=1}^{b^t-1} \sum_{j=1}^{b^t} \frac{x_{u,v^g}^{*,t} \cdot \gamma}{h_{v^g}^t \cdot P_{v^g,(i,j)}^t \cdot \beta_{u,(i,j)}^t} \cdot P_{v^g,(i,j)}^t \cdot \beta_{u,(i,j)}^t \\ &\quad - \sum_{v^g} \sum_{j=1}^{b^t-1} \frac{x_{u,v^g}^{*,t} \cdot \gamma}{h_{v^g}^t \cdot P_{v^g,(i,j)}^t \cdot \beta_{u,(i,j)}^t} \cdot P_{v^g,(i,j)}^t \cdot \beta_{u,(i,j)}^t \end{aligned} \quad (C.60)$$

Similar to $t = 1$ case

$$\begin{aligned} \beta_{u,(b^t,b^t)}^t &= \beta_{u,(1,1)}^t - \sum_{v^g} (x_{u,v^g}^{*,t} \cdot \gamma) \\ \beta_{u,(b^t,b^t)}^t &\geq 1 - \gamma + \sum_{v^g} x_{u,v^g}^{*,t} \cdot \gamma - \sum_{v^g} (x_{u,v^g}^{*,t} \cdot \gamma) \end{aligned} \quad (C.61)$$

$$\beta_{u,(b^t,b^t)}^t \geq 1 - \gamma \quad (C.62)$$

Therefore,

$$\beta_{u,(i,j)}^t \geq (1 - \gamma), \forall t, i, j$$

C.8 Proof of Corollary 1

Proposition 14. *The online algorithm ADAPShare is γ competitive (The value of γ is the solution to the equation $(1 - \gamma)^{(\kappa+1)} = \gamma$).*

Proof: Using Proposition 10, we can show that ADAPShare- κ is γ competitive. Therefore, we need to find the maximum value of γ for which the assignment rule for ADAPShare- κ is valid. We highlight the differences in comparison to proof of Proposition 9.

We can now group κ vertices together from the b^t vertices arriving in round t , therefore, we define $(b^t)^\kappa$ steps in our algorithm.

Similar to ADAPShare-2, Step $(i_1, i_2, \dots, i_\kappa)$ ($i_1 \neq i_2 \neq \dots \neq i_\kappa$) denotes that the group formed by vertex at $i_1^{th}, i_2^{th} \dots$ and $\dots i_\kappa^{th}$ label is considered. Step (i, i, \dots, i) denotes that a group of size 1 with one vertex at i^{th} position is considered. Similarly, we can define steps where groups of size 2 to $\kappa - 1$ are considered. Group of size s will be considered at $\prod_{i=0}^{s-1} (b^t - i)$ steps. There will be $(b^t)^\kappa - \sum_{s=1}^{\kappa} \prod_{i=0}^s (b^t - i)$ steps where algorithm does not do anything, i.e., none of the groups is considered for assignment at these steps.

Similar to ADAPShare-2 case, we find the maximum value of $\frac{\prod_{v \in v^g} (p_v^t)^{n_{v,v^g}}}{P_{v^g, (i_1, i_2, \dots, i_\kappa)}^t}$ and then show that $\beta_{u, (i_1, i_2, \dots, i_\kappa)}^t \geq \frac{\gamma \cdot \prod_{v \in v^g} (p_v^t)^{n_{v,v^g}}}{P_{v^g, (i_1, i_2, \dots, i_\kappa)}^t}$ for this maximum value.

Please note that

$$P_{v^g, (i_1, i_2, \dots, i_\kappa)}^t = \prod_{v \in v^g} P_{v, i_v, (i_1, i_2, \dots, i_\kappa)}^t.$$
 Therefore, we compute $P_{v, i_v, (i_1, i_2, \dots, i_\kappa)}^t \forall v$ where $P_{v, i_v, (i_1, \dots, i_\kappa)}^t$ denotes the probability that vertex of type v labeled as i_v^{th} vertex is available at step (i_1, \dots, i_κ) . Please note that $P_{v, i, (1, 1, \dots, 1)}^t = p_v^t \forall i$.

$$\begin{aligned}
P_{v,i_v,(i_1,\dots,i_\kappa)}^t &= p_v^t - \\
&\sum_u \sum_{v^{g'}; v \in v^{g'}} \sum_{j_1=1}^{i_1-1} \dots \sum_{j_\kappa=1}^{i_\kappa-1} \frac{x_{u,v^{g'}}^t \cdot \gamma}{h_{v^{g'}}^t \cdot P_{v^{g'},(j_1,\dots,j_\kappa)}^t \cdot \beta_{u,(j_1,\dots,j_\kappa)}^t} \cdot P_{v^{g'},(j_1,\dots,j_\kappa)}^t \cdot \beta_{u,(j_1,\dots,j_\kappa)}^t
\end{aligned} \tag{C.63}$$

Now, similar to $\kappa = 2$ case, in the above equation we only consider the steps where vertex of type v has label i_v . Let $e_{v^g,v,i_v,(i_1,\dots,i_\kappa)}^t$ denote the maximum number of steps (for group of type v^g) which can affect the computation of $P_{v,i_v,(i_1,\dots,i_\kappa)}^t$, then

$$\begin{aligned}
P_{v,i_v,(i_1,\dots,i_\kappa)}^t &= p_v^t - \\
&\sum_u \sum_{v^{g'}; v \in v^{g'}} e_{v^g,v,i_v,(i_1,\dots,i_\kappa)}^t \cdot \frac{x_{u,v^{g'}}^t \cdot \gamma}{h_{v^{g'}}^t}
\end{aligned} \tag{C.64}$$

And $e_{v^g,v,i_v,(i_1,\dots,i_\kappa)}^t = \frac{\prod_{i=1}^{|v^g|} (b^t - i)}{(\prod_{v' \in v^g; v \neq v'} (n_{v',v^g})!) (n_{v,v^g} - 1)!}$

Therefore, $\frac{e_{v^g,v,i_v,(i_1,\dots,i_\kappa)}^t}{h_{v^g}^t} = \frac{n_{v,v^g}}{b^t}$

Therefore, we can get following by proceeding in similar way as the analysis for $\kappa = 2$

$$P_{v,i_v,(i_1,\dots,i_\kappa)}^t \geq p_v^t - \gamma \cdot p_v^t \tag{C.65}$$

$$P_{v,i_v,(i_1,\dots,i_\kappa)}^t \geq (1 - \gamma) \cdot p_v^t \tag{C.66}$$

Similar to $\kappa = 2$, we can use mathematical induction to show $\beta_{u,(i_1,\dots,i_\kappa)}^t \geq \frac{\gamma}{(1-\gamma)^\kappa}$. Therefore, maximum value of γ for which assignment rule is valid is the solution to the equation, $\gamma = (1 - \gamma)^{\kappa+1}$

Algorithm 13 ADAPShare- $\kappa(\gamma)$

- 1: **for** $t < T$ **do**
 - 2: Generate b^t uniform random numbers and sort the vertices in order of generated random numbers. Label the vertices from 1 to b^t .
 - 3: $i_1 = 1, i_2 = 1, \dots, i_\kappa = 1$
 - 4: **while** $i_1 \leq b^t \parallel i_2 \leq b^t \parallel \dots \parallel i_\kappa \leq b^t$ **do**
 - 5: $v^g =$ group formed at step $i_1, i_2, \dots, i_\kappa$ based on the labels assigned to the vertices.
 - 6: **if** v^g is a valid group **then**
 - 7: If $\mathcal{E}_{*,v^g,t} \neq \emptyset$, then choose $(u, v^g) \in \mathcal{E}_{*,v^g,t}$ with probability p where
$$p = \frac{x_{u,v^g}^{*,t} \cdot \gamma}{h_{v^g}^t \cdot P_{v^g,(i_1,i_2,\dots,i_\kappa)}^t \cdot \beta_{u,(i_1,\dots,i_\kappa)}^t}$$
 - 8: Update $\mathcal{E}_{*,*,t}$, available groups based on the group considered in previous steps.
 - 9: Increment the step $i_1 = 1, i_2 = 1, \dots, i_\kappa$
-

References

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems*. Retrieved from <https://www.tensorflow.org/> (Software available from tensorflow.org)
- Abbasi, Y., Bartlett, P. L., Kanade, V., Seldin, Y., & Szepesvári, C. (2013). Online learning in Markov decision processes with adversarially chosen transition probability distributions. In *Advances in neural information processing systems* (pp. 2508–2516).
- Agatz, N., Erera, A. L., Savelsbergh, M. W., & Wang, X. (2011). Dynamic ride-sharing: A simulation study in metro atlanta. *Procedia-Social and Behavioral Sciences*, 17, 532–550.
- Aggarwal, G., Goel, G., Karande, C., & Mehta, A. (2011). Online vertex-weighted bipartite matching and single-bid budgeted allocations. In *Proceedings of the twenty-second annual acm-siam symposium on discrete algorithms* (pp. 1253–1264).
- Alonso-Mora, J., Samaranayake, S., Wallar, A., Frazzoli, E., & Rus, D. (2017). On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3), 462–467.

- Alonso-Mora, J., Wallar, A., & Rus, D. (2017). Predictive routing for autonomous mobility-on-demand systems with ride-sharing. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 3583–3590).
- Ashlagi, I., Burq, M., Dutta, C., Jaillet, P., Saberi, A., & Sholley, C. (2018). Maximum weight online matching with deadlines. *arXiv preprint arXiv:1808.03526*.
- Ashlagi, I., Burq, M., Dutta, C., Jaillet, P., Saberi, A., & Sholley, C. (2019). Edge weighted online windowed matching. In *Proceedings of the 2019 ACM conference on economics and computation* (pp. 729–742).
- Banerjee, S., Johari, R., & Riquelme, C. (2015). Pricing in ride-sharing platforms: A queueing-theoretic approach. In *Proceedings of the sixteenth ACM conference on economics and computation* (pp. 639–639).
- Banerjee, S., Johari, R., & Riquelme, C. (2016). Dynamic pricing in ridesharing platforms. *ACM SIGecom Exchanges*, 15(1), 65–70.
- Beeline. (2016). *Beeline singapore - book seat on the buses*. <https://www.beeline.sg/>.
- Beineke, L. W. (1980). The four color problem: Assaults and conquest (thomas saaty and paul kainen). *SIAM Review*, 22(2), 241–243.
- Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik*, 4(1), 238–252.
- Bent, R., & Van Hentenryck, P. (2004). Regrets only! online stochastic optimization under time constraints. In *Aaai* (Vol. 4, pp. 501–506).
- Bent, R. W., & Van Hentenryck, P. (2004). Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Operations Research*, 52(6), 977–987.
- Bertsimas, D., Jaillet, P., & Martin, S. (2019). Online vehicle routing: The edge of optimization in large-scale applications. *Operations Research*, 67(1), 143–162.
- Bertsimas, D., & Tsitsiklis, J. N. (1997). *Introduction to linear optimization*

(Vol. 6). Athena Scientific Belmont, MA.

- Biswas, A., Gopalakrishnan, R., Tulabandhula, T., Metrewar, A., Mukherjee, K., & Thangaraj, R. S. (2018). Impact of detour-aware policies on maximizing profit in ridesharing. In *Proceedings of the 10th international workshop on agents in traffic and transportation (att 2018), stockholm, sweden*.
- Blum, A., Sandholm, T., & Zinkevich, M. (2006). Online algorithms for market clearing. *J. ACM*, 53(5), 845–879.
- Boeing, G. (2017). Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65, 126–139.
- Bonifaci, V., Lipmann, M., Stougie, L., et al. (2006). *Online multi-server dial-a-ride problems*. TU/e, Eindhoven University of Technology, Department of Mathematics and
- Borodin, A., & El-Yaniv, R. (1998). *Online computation and competitive analysis*. New York, NY, USA: Cambridge University Press.
- Brown, T. (2016). *Matchmaking in lyft line*. <https://eng.lyft.com/matchmaking-in-lyft-line-691a1a32a008>.
- Chemla, D., Meunier, F., & Calvo, R. W. (2013). Bike sharing systems: Solving the static rebalancing problem. *Discrete Optimization*, 10(2), 120–146.
- Chen, L., Gao, Y., Liu, Z., Xiao, X., Jensen, C. S., & Zhu, Y. (2018). Ptrider: a price-and-time-aware ridesharing system. *Proceedings of the VLDB Endowment*, 11(12), 1938–1941.
- Cheng, P., Xin, H., & Chen, L. (2017). Utility-aware ridesharing on road networks. In *Proceedings of the 2017 acm international conference on management of data* (pp. 1197–1210).
- Contardo, C., Morency, C., & Rousseau, L.-M. (2012). *Balancing a dynamic public bike-sharing system* (Vol. 4). Cirrelt.
- Devanur, N. R., Jain, K., & Kleinberg, R. D. (2013). Randomized primal-dual analysis of ranking for online bipartite matching. In *Proceedings of the twenty-*

- fourth annual acm-siam symposium on discrete algorithms* (pp. 101–107).
- Dickerson, J. P., Sankararaman, K. A., Sarpatwar, K. K., Srinivasan, A., Wu, K.-L., & Xu, P. (2019). Online resource allocation with matching constraints. In *Proceedings of the 18th international conference on autonomous agents and multiagent systems* (pp. 1681–1689).
- Dickerson, J. P., Sankararaman, K. A., Srinivasan, A., & Xu, P. (2017). Allocation problems in ride-sharing platforms: Online matching with offline reusable resources. *arXiv preprint arXiv:1711.08345*.
- Dickerson, J. P., Sankararaman, K. A., Srinivasan, A., & Xu, P. (2018). Allocation problems in ride-sharing platforms: Online matching with offline reusable resources. In *Aaai*.
- Dror, M., Fortin, D., & Roucairol, C. (1998). *Redistribution of self-service electric cars: A case of pickup and delivery* (Unpublished doctoral dissertation). INRIA.
- Dube-Rioux, L., Schmitt, B. H., & Leclerc, F. (1989). Consumers' reactions to waiting: when delays affect the perception of service quality. *ACR North American Advances*.
- Dutta, C. (2018). When hashing met matching: Efficient search for potential matches in ride sharing. *arXiv preprint arXiv:1809.02680*.
- Even-Dar, E., Kakade, S. M., & Mansour, Y. (2009). Online Markov decision processes. *Mathematics of Operations Research*, 34(3), 726–736.
- Feldman, J., Korula, N., Mirrokni, V., Muthukrishnan, S., & Pál, M. (2009). Online ad assignment with free disposal. In *International workshop on internet and network economics* (pp. 374–385).
- Feuerstein, E., & Stougie, L. (2001). On-line single-server dial-a-ride problems. *Theoretical Computer Science*, 268(1), 91–105.
- Fisher, M. L. (1985). An applications oriented guide to lagrangian relaxation. *Interfaces*, 15(2), 10–21.
- Ghiani, G., Manni, E., & Thomas, B. W. (2012). A comparison of anticipatory al-

- gorithms for the dynamic and stochastic traveling salesman problem. *Transportation Science*, 46(3), 374–387.
- Ghosh, S., Trick, M., & Varakantham, P. (2016). Robust repositioning to counter unpredictable demand in bike sharing systems. In *International joint conference on artificial intelligence (ijcai)*.
- Ghosh, S., & Varakantham, P. (2017). Incentivising the use of bike trailers for dynamic repositioning in bike sharing systems. In *International conference on automated planning and scheduling (icaps)*.
- Ghosh, S., Varakantham, P., Adulyasak, Y., & Jaillet, P. (2015). Dynamic redeployment to counter congestion or starvation in vehicle sharing systems. In *International conference on automated planning and scheduling (icaps)*.
- Ghosh, S., Varakantham, P., Adulyasak, Y., & Jaillet, P. (2017). Dynamic repositioning to reduce lost demand in bike sharing systems. *Journal of Artificial Intelligence Research*, 58, 387–430.
- Godfrey, G. A., & Powell, W. B. (2002). An adaptive dynamic programming algorithm for dynamic fleet management, ii: Multiperiod travel times. *Transportation Science*, 36(1), 40–54.
- Grab. (2018). *Grab shuttle plus - on demand shuttle service*. <https://www.grab.com/sg/shuttleplus/>.
- Gunes, C., van Hoes, W.-J., & Tayur, S. (2010). Vehicle routing for food rescue programs: A comparison of different approaches. In *International conference on integration of artificial intelligence (ai) and operations research (or) techniques in constraint programming* (pp. 176–180).
- Hasan, M. H., Van Hentenryck, P., Budak, C., Chen, J., & Chaudhry, C. (2018). Community-based trip sharing for urban commuting.
- Hernández-Pérez, H., & Salazar-González, J.-J. (2004). A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics*, 145(1), 126–139.
- Hernández-Pérez, H., & Salazar-González, J.-J. (2007). The one-commodity

- pickup-and-delivery traveling salesman problem: Inequalities and algorithms. *Networks: An International Journal*, 50(4), 258–272.
- Hoffman, A., & Kruskal, J. (2010). Introduction to integral boundary points of convex polyhedra. *Jünger M et al (eds)*, 50, 1958–2008.
- Hosni, H., Naoum-Sawaya, J., & Artail, H. (2014). The shared-taxi problem: Formulation and solution methods. *Transportation Research Part B: Methodological*, 70, 303–318.
- Huang, Y., Bastani, F., Jin, R., & Wang, X. S. (2014). Large scale real-time ridesharing with service guarantee on road networks. *Proceedings of the VLDB Endowment*, 7(14), 2017–2028.
- Huang, Z., Kang, N., Tang, Z. G., Wu, X., Zhang, Y., & Zhu, X. (2018). How to match when all vertices arrive online. In *Proceedings of the 50th annual acm sigact symposium on theory of computing* (pp. 17–29).
- Jaillet, P., & Lu, X. (2013). Online stochastic matching: New algorithms with better bounds. *Mathematics of Operations Research*, 39(3), 624–646.
- Karp, R. M., Vazirani, U. V., & Vazirani, V. V. (1990a). An optimal algorithm for on-line bipartite matching. In H. Ortiz (Ed.), *Stoc* (p. 352-358). ACM.
- Karp, R. M., Vazirani, U. V., & Vazirani, V. V. (1990b). An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual acm symposium on theory of computing* (pp. 352–358).
- Katriel, I., Kenyon-Mathieu, C., & Upfal, E. (2008). Commitment under uncertainty: Two-stage stochastic matching problems. *Theoretical Computer Science*, 408(2), 213–223.
- Kumar, A., Wu, X., & Zilberstein, S. (2012). Lagrangian relaxation techniques for scalable spatial conservation planning.
- Lee, E., & Singla, S. (2017). Maximum matching in the online batch-arrival model. In *International conference on integer programming and combinatorial optimization* (pp. 355–367).
- Legrain, A., & Jaillet, P. (2016). A stochastic algorithm for online bipartite resource

- allocation problems. *Computers & Operations Research*, 75, 28–37.
- Li, L., Walsh, T. J., & Littman, M. L. (2006). Towards a unified theory of state abstraction for MDPs. In *Isaim*.
- Lipmann, M., Lu, X., de Paepe, W. E., Sitters, R. A., & Stougie, L. (2002). On-line dial-a-ride problems under a restricted information model. In *European symposium on algorithms* (pp. 674–685).
- Lowalekar, M., Varakantham, P., & Jaillet, P. (2018). Online spatio-temporal matching in stochastic and dynamic domains. *Artificial Intelligence*, 261, 71–112.
- Lowalekar, M., Varakantham, P., & Jaillet, P. (2019). ZAC: A zone path construction approach for effective real-time ridesharing. In *Proceedings of the twenty-ninth international conference on automated planning and scheduling, ICAPS 2019, berkeley, ca, usa, july 11-15, 2019*. (pp. 528–538).
- Ma, H., Fang, F., & Parkes, D. C. (2018). Spatio-temporal pricing for ridesharing platforms. *arXiv preprint arXiv:1801.04015*.
- Ma, S., Zheng, Y., & Wolfson, O. (2013). T-share: A large-scale dynamic taxi ridesharing service. In *Data engineering (icde), 2013 ieee 29th international conference on* (pp. 410–421).
- Maister, D. H., et al. (1984). *The psychology of waiting lines*. Harvard Business School Boston, MA.
- Manshadi, V. H., Gharan, S. O., & Saberi, A. (2012). Online stochastic matching: Online actions based on offline statistics. *Mathematics of Operations Research*, 37(4), 559–573.
- Maxwell, M. S., Restrepo, M., Henderson, S. G., & Topaloglu, H. (2010). Approximate dynamic programming for ambulance redeployment. *INFORMS Journal on Computing*, 22(2), 266–281.
- McMahan, H. B., Gordon, G. J., & Blum, A. (2003). Planning in the presence of cost functions controlled by an adversary. In *Icml* (pp. 536–543).
- Mehta, A., et al. (2013). Online matching and ad allocation. *Foundations and Trends® in Theoretical Computer Science*, 8(4), 265–368.

- Mercier, L., & Van Hentenryck, P. (2007). Performance analysis of online anticipatory algorithms for large multistage stochastic integer programs. In *Ijcai* (pp. 1979–1984).
- Mercier, L., & Van Hentenryck, P. (2011). An anytime multistep anticipatory algorithm for online stochastic combinatorial optimization. *Annals of Operations Research*, 184(1), 233–271.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... others (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529.
- Murphy, J. (2013). Benders, nested benders and stochastic programming: An intuitive introduction. *arXiv preprint arXiv:1312.3158*.
- NYYellowTaxi. (2016). *New york yellow taxi dataset*. http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml.
- Parragh, S. N., Doerner, K. F., & Hartl, R. F. (2008). A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, 58(1), 21–51.
- Pelzer, D., Xiao, J., Zehe, D., Lees, M. H., Knoll, A. C., & Aydt, H. (2015). A partition-based match making algorithm for dynamic ridesharing. *IEEE Transactions on Intelligent Transportation Systems*, 16(5), 2587–2598.
- Pfrommer, J., Warrington, J., Schildbach, G., & Morari, M. (2014). Dynamic vehicle redistribution and online price incentives in shared mobility systems. *IEEE Transactions on Intelligent Transportation Systems*, 15(4), 1567–1578.
- Plappert, M., Houthoofd, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., ... Andrychowicz, M. (2017). Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*.
- Powell, W. B. (1996). A stochastic formulation of the dynamic assignment problem, with an application to truckload motor carriers. *Transportation Science*, 30(3), 195–219.
- Powell, W. B. (2007). *Approximate dynamic programming: Solving the curses of dimensionality* (Vol. 703). John Wiley & Sons.

- Raviv, T., Tzur, M., & Forma, I. A. (2013). Static repositioning in a bike-sharing system: models and solution approaches. *EURO Journal on Transportation and Logistics*, 2(3), 187–229.
- Rebennack, S. (2016). Combining sampling-based and scenario-based nestedenders decomposition methods: application to stochastic dual dynamic programming. *Mathematical Programming*, 156(1-2), 343–389.
- Ritzinger, U., Puchinger, J., & Hartl, R. F. (2016). A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research*, 54(1), 215–231.
- Ropke, S., Cordeau, J.-F., & Laporte, G. (2007). Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4), 258–272.
- Russell, S. J., & Zimdars, A. (2003). Q-decomposition for reinforcement learning agents. In *Proceedings of the 20th international conference on machine learning (icml-03)* (pp. 656–663).
- Sankararaman, K. A. (2019). *Sequential decision making with limited resources* (Unpublished doctoral dissertation). University of Maryland, College Park.
- Santi, P., Resta, G., Szell, M., Sobolevsky, S., Strogatz, S. H., & Ratti, C. (2014). Quantifying the benefits of vehicle pooling with shareability networks. *Proceedings of the National Academy of Sciences*, 111(37), 13290–13294.
- Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.
- Schrank, D., Eisele, B., & Lomax, T. (2012). Tti's 2012 urban mobility report. *Texas A&M Transportation Institute. The Texas A&M University System*, 4.
- Schuijbroek, J., Hampshire, R., & van Hoes, W.-J. (2017). Inventory rebalancing and vehicle routing in bike sharing systems. *European Journal of Operational Research*, 257(3), 992–1004.
- Shaheen, S., Cohen, A., Zohdy, I., et al. (2016). *Shared mobility: current practices and guiding principles* (Tech. Rep.). United States. Federal Highway

Administration.

- Shapiro, A. (2006). On complexity of multistage stochastic programs. *Operations Research Letters*, 34(1), 1–8.
- Shapiro, A., Dentcheva, D., & Ruszczyński, A. (2009). *Lectures on stochastic programming: modeling and theory*. SIAM.
- Shotl. (2018). *Shotl on demand shuttles*. <https://shotl.com/>.
- Shu, J., Chou, M. C., Liu, Q., Teo, C.-P., & Wang, I.-L. (2013). Models for effective deployment and redistribution of bicycles within public bicycle-sharing systems. *Operations Research*, 61(6), 1346–1359.
- Simao, H. P., Day, J., George, A. P., Gifford, T., Nienow, J., & Powell, W. B. (2009). An approximate dynamic programming algorithm for large-scale fleet management: A case application. *Transportation Science*, 43(2), 178–197.
- Singapore taxi fare. (n.d.). <https://www.lta.gov.sg/content/ltaweb/en/public-transport/taxis/fares-and-payment-methods.html>.
- Spivey, M. Z., & Powell, W. B. (2004). The dynamic assignment problem. *Transportation Science*, 38(4), 399–419.
- Szepesvári, C. (2010). Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1), 1–103.
- Tang, M., Ow, S., Chen, W., Cao, Y., Lye, K.-w., & Pan, Y. (2017). The data and science behind grabshare carpooling. In *Data science and advanced analytics (dsaa), 2017 ieee international conference on* (pp. 405–411).
- Tong, Y., Zeng, Y., Zhou, Z., Chen, L., Ye, J., & Xu, K. (2018). A unified approach to route planning for shared mobility. *Proceedings of the VLDB Endowment*, 11(11), 1633–1646.
- Topaloglu, H., & Powell, W. B. (2006). Dynamic-programming approximations for stochastic time-staged integer multicommodity-flow problems. *INFORMS Journal on Computing*, 18(1), 31–42.
- United Nations. (2018). *United nations report*. <https://www.un.org/>

development/desa/en/news/population/2018-revision-of-world-urbanization-prospects.html.

- Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Thirtieth aai conference on artificial intelligence*.
- Wallar, A., Van Der Zee, M., Alonso-Mora, J., & Rus, D. (2018). Vehicle rebalancing for mobility-on-demand systems with ride-sharing. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 4539–4546).
- Wang, Y., & Wong, S. C. (2015). Two-sided online bipartite matching and vertex cover: Beating the greedy algorithm. In *Automata, languages, and programming - 42nd international colloquium, ICALP 2015, kyoto, japan, july 6-10, 2015, proceedings, part I* (pp. 1070–1081).
- Widdows, D., Lucas, J., Tang, M., & Wu, W. (2017). Grabshare: The construction of a realtime ridesharing service. In *Intelligent transportation engineering (icite), 2017 2nd IEEE International Conference on* (pp. 138–143).
- Xu, Z., Li, Z., Guan, Q., Zhang, D., Li, Q., Nan, J., ... Ye, J. (2018). Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (pp. 905–913).
- Yu, X., & Shen, S. (2019). An integrated decomposition and approximate dynamic programming approach for on-demand ride pooling. *IEEE Transactions on Intelligent Transportation Systems*.
- Zhang, G., Smilowitz, K., & Erera, A. (2011). Dynamic planning for urban drayage operations. *Transportation Research Part E: Logistics and Transportation Review*, 47(5), 764–777.
- Zheng, L., Chen, L., & Ye, J. (2018). Order dispatch in price-aware ridesharing. *Proceedings of the VLDB Endowment*, 11(8), 853–865.
- Ziegelmann, M. (2001). *Constrained shortest paths and related problems* (Unpublished doctoral dissertation). Universitätsbibliothek.