11-2022

Efficient navigation for constrained shortest path with adaptive expansion control

Wenwen XIA

Yuchen LI
Singapore Management University, yuchenli@smu.edu.sg

Wentian GUO

Shenghong LI

Citation

XIA, Wenwen; LI, Yuchen; GUO, Wentian; and LI, Shenghong. Efficient navigation for constrained shortest path with adaptive expansion control. (2022). *2022 IEEE International Conference on Data Mining (ICDM): Orlando, November 22 - December 1: Proceedings*. 588-597.
Available at: https://ink.library.smu.edu.sg/sis_research/7782

2022 IEEE International Conference on Data Mining (ICDM)

pp. 588-597. DOI: 10.1109/ICDM54844.2022.00069

Efficient Navigation for Constrained Shortest Path with Adaptive Expansion Control

Wenwen Xia[1]*, Yuchen Li[2], Wentian Guo[3], Shenghong Li[4]
[1,4] *Shanghai Jiao Tong University*, [4]MoE Key Lab of Artificial Intelligence, AIInstitute
[2]Singapore Management University
[3]Meta Platforms, Inc.
{xiawenwen, shli}@sjtu.edu.cn, yuchenli@smu.edu.cn, wentianguo@fb.com,

***Abstract*—In many route planning applications, finding constrained shortest paths (CSP) is an important and fundamental problem. CSP aims to find the shortest path between two nodes on a graph while satisfying a path constraint. Solving CSPs requires a large search space and is prohibitively slow on large graphs, even with the state-of-the-art parallel solution on GPUs. The reason lies in the lack of effective navigational information and pruning strategies in the search procedure. In this paper, we propose SPEC, a *S*hortest *P*ath *E*nhanced approach for solving the exact *C*SP problem. Our design rationales of SPEC rely on the observation that the shortest path (SP) provides valuable information in the search procedure of CSP. Hence, we propose a label priority that distinguishes promising candidate paths based on SP. We further devise efficient pruning and teleporting strategies utilizing SP lengths and costs, which eliminates unfeasible paths at an early stage. Furthermore, we observe that the expansion number at each search iteration affects the overall performance significantly. Thus, we devise an adaptive controller based on reinforcement learning. We also show that SPEC works seamlessly with the parallel implementation. Extensive experimental results on 8 read-world graphs reveal that single thread SPEC achieves an order of magnitude speedup over the state-of-the-art GPU-based method. The parallel implementation boosts SPEC 3 to 5 times further.**

***Index Terms*—constrained shortest path, shortest path**

I. Introduction

The constrained shortest path (CSP) problem is an essential component in many applications. Given a graph on which each edge has two properties, i.e., the length and cost, CSP finds the shortest path between two nodes *source* and *target* without exceeding a cost limit. For example, in the vehicle routing problem [1], the length of an edge is the road length, while the cost could be the road charge. The goal of CSP is to find the shortest path under a road charge limit. Many applications impose rigorous constraints and seek exact solutions, such as Quality of Service (QoS) package routing with delay constraints on Internet networks [2], [3], battery-limited drone deliveries [4], etc. In these applications, approximate solutions that may exceed the cost limit are not favored because of service quality or security concerns [5]. In this paper, we focus on the exact CSP problem and consider one cost constraint, which is the most common case in practical applications. Solving CSP on large graphs efficiently is both theoretically NP-hard and empirically challenging [1], [6], [7].

*This work was done when Wenwen Xia was visiting Singapore Management University.

The seminal method for CSP is the labeling algorithm (LA) based on dynamic programming [8], which has been well studied and extended to solve the CSP problem and its variants [9]–[11]. In LA, each path from the source node to a visited node v is denoted as a **label** l_v on v. l_v stores its length and cost values. LA maintains a frontier (a priority queue) to store labels for expansion. At each step, the top label l_t is popped and new labels are created by extending the length and cost of l_t along all outgoing edges of t. These new labels are pruned or inserted back into the frontier. The algorithm terminates until the frontier is empty. The details of LA will be presented in Sec. II-A. Recently, Lu et al. propose a framework named Vine that utilizes the massively parallel Graphics Processing Units (GPUs) to accelerate LA and delivers the state-of-the-art performance for CSP [7]. However, the efficiency of Vine is still unsatisfactory for real-time applications on large graphs. For example, it takes several seconds to find one CSP in a graph with only a few thousand nodes. Furthermore, the reliance on hardware accelerators like GPUs also imposes limitations in cases where such resource is inaccessible for space and cost concerns [2], [12].

Motivations. There are two key challenges in accelerating LA. First, it is unclear how to choose an efficient label expansion order to navigate the search procedure faster. Second, it is difficult to prune feasible yet sub-optimal labels effectively at an early stage. Existing methods only expand the labels with small length or cost values and prune unpromising labels that have larger length and cost than existing ones [7], [8]. However, the *unconstrained* shortest path (SP) information between nodes in the search procedure has been entirely overlooked. In fact, the SP information provides valuable hints to optimize the expansion order and significantly reduces the search space. The reason is SP length is often a good (if not best) criterion to quantify a label's quality. SP length and cost values could be used for pruning which avoids a large number of sub-optimal labels at an early stage. Furthermore, the overhead of querying the SP information is almost negligible for CSP processing as the SP queries can be solved 5 to 6 orders of magnitudes faster than CSP queries (Figure 12 in Appendix). Thus, the benefit of employing SP information in the searching outweighs the overheads of processing SP queries for CSP.

Solution. Based on the motivations, we propose the SP Enhanced CSP (SPEC) algorithm that exploits the SP information for efficient label expansion and effective label pruning. To devise an efficient label expansion order, we rely on a new type of label priority that considers not only the label length from the source but also the SP length to the target node. This label priority can reduce the iteration number of the frontier expansion and thus navigate the search faster. To effectively prune the sub-optimal labels, SPEC progressively maintains a current best CSP length as the pruning threshold. We can safely prune a label if it cannot achieve a valid and better solution, e.g., if the shortest path in terms of length is larger than the current best length, or if the shortest path in terms of cost exceeds the global cost limit. To facilitate both the label expansion and pruning, we adapt the state-of-the-art indexing method for the SP algorithms to efficiently compute the SP in terms of length and cost.

Moreover, we need to carefully choose the number of labels expanded in each iteration as it could affect the overall performance as well. A straightforward approach as in the existing solutions is to expand the top label in the frontier since it has the highest priority. However, counter-intuitively, we find that this approach might be sub-optimal because it can cause SPEC stuck in expanding the labels that seem promising but are actually unfeasible. On the other side, expanding too many labels will lead to a redundant workload. Hence, we formalize the label expansion process as a sequential decision making procedure and design a reinforcement learning (RL)-based adaptive expansion controller. The RL controller takes statistics of the frontier expansion and returns an action as the expansion number. To further boost the performance, we propose a parallel scheme that works seamlessly with all proposed techniques.

Summary of Contributions.

- We propose SPEC, an exact CSP solving method. We devise an efficient expansion priority and prune strategies by utilizing the SP information which has been entirely overlooked in the literature.
- We devise a novel RL controller to adaptively expand the label frontier for reducing overheads on unpromising labels. Our techniques fit nicely with the parallel implementation.
- Extensive experiments on 8 real-world graphs show that single-thread SPEC is on average 30× faster compared to the state-of-the-art GPU-accelerated approach [7]. Multi-thread SPEC is 3∼4× faster than single thread SPEC. The implementation of SPEC and datasets are released [1].

The rest of this paper is organized as follows. In Sec. II, we elaborate the CSP problem and related work. In Sec. III, we elaborate the SP-based label priorities and pruning strategies in SPEC. In Sec. IV, we present the adaptive RL controller and the parallel implementation. In Sec. V and Sec. VI, we present experimental setup and results. We conclude the paper in Sec. VII.

[1]https://github.com/xiawenwen49/SPEC

II. Background and Related work

In this section, we first elaborate on the definition of the CSP problem and the details of the labeling algorithm with a toy example. Then we summarize related works of both SP and CSP.

A. CSP and The Labeling Algorithm

Definition 1 (CSP). *Given a directed graph $G = (V, E)$ where V is the node set and E is the edge set, each edge e is associated with a length $e.l$ and cost $e.c$. We assume lengths and costs are nonnegative. A path p consists of a sequence of edges, $p =< e_1, e_2, \cdots, e_{|p|} >$. The path length $p.l$ and path cost $p.c$ can be computed as $\sum_{i=1}^{|p|} e_i.l$ and $\sum_{i=1}^{|p|} e_i.c$ respectively. Given a* ***query*** *(u, v, c) with source u, target v, and a cost limit c, let $\mathbb{P}$ denotes all paths from u to v with path cost no larger than c. The CSP problem aims to find the shortest path in $\mathbb{P}$, i.e., $p^* = \mathrm{argmin}_{p\in\mathbb{P}}\, p.l$.*

The Labeling Algorithm. The most commonly used algorithm for exact CSP is the labeling algorithm, also called Sky-Dijk [7], [8]. In the process of solving a query (u, v, c), a label l_k for node k is defined as a path's information from source u to k. l_k only stores the path's length, cost, and node k, i.e., $l_k = (l, c, k)$ where $l = p.l, c = p.c$. p indicates the path that l_k represents. To prune labels that are deemed sub-optimal, label dominance is defined as follows.

Definition 2 (Dominance). *Label l_k dominates label l_j if $l_k.l \le l_j.l$ and $l_k.c \le l_j.c$.*

The labeling algorithm maintains a non-dominant label list at each node and a global frontier that stores labels for expansion. Labels in the frontier (a priority queue) are maintained in ascending order of their cost/length. At each step, the top label l_k is popped and expanded along the out edges of node k, i.e., extending the label length and cost with an edge's length and cost. If a newly generated label l_j is not dominated by existing labels on node v_j and doesn't exceed the cost budget, it will be inserted back into the frontier and the non-dominant label list on v_j, otherwise discarded. The expanding process terminates until the frontier is empty.

Example 1. *Consider the graph in Figure 1, and assume the query is $(v_1, v_5, 50)$. We plot the frontier's labels at each step in Figure 2. The frontier is initialized with label $(0, 0, v_1)$. At the 1st step, $(2, 10, v_2)$ and $(1, 30, v_3)$ are generated along edges (v_1, v_2) and (v_1, v_3). At the 2nd step, $(7, 20, v_4)$ and $(3, 20, v_3)$ are generated from $(2, 10, v_2)$ along edge (v_2, v_4) and (v_2, v_3). At the 3rd step, $(11, 40, v_5)$ is generated from $(7, 20, v_4)$, and it then stops since the target v_5 is found. At the 4th step, $(4, 30, v_4)$ is generated from $(3, 20, v_3)$ along edge (v_3, v_4). $(6, 80, v_5)$ is discarded because 80 exceeds the cost limit 50. At the 5th step, $(8, 50, v_5)$ is generated from $(4, 30, v_4)$ along edge (v_4, v_5) and the target is found. At the 6th step, $(4, 90, v_5)$ is generated from $(1, 30, v_3)$ but is discarded since 90 exceeds the cost limit of 50. Finally, $(8, 50, v_5)$ is returned as the solution.*

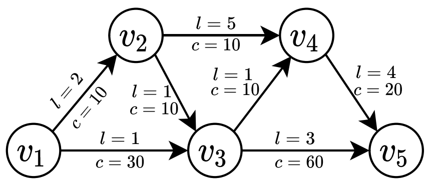


Fig. 1. A sample graph.

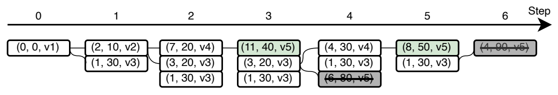


Fig. 2. Processing the example graph in Figure 1. The source is v_1, target is v_5, and cost budget is set to 50. Green labels are feasible solutions and will stop expanding, while gray ones are pruned.

B. Related Work

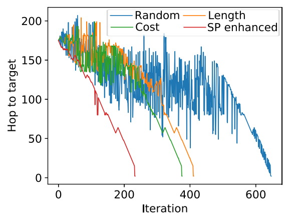
The CSP problem was firstly proposed in [13], where CSP is formulated as an integer linear programming (ILP) problem. [14], [15] also solve the exact CSP with ILP solvers. Nonetheless, these methods hardly scale to large networks. [8] proposes the labeling algorithm as described in Sec. II-A, which has good empirical performance [16]. [17] proposes a kth SP algorithm, which can also be adapted to solve the exact CSP problem, e.g., iteratively computing the next SP solution until the cost limit is satisfied. [18] proposed a forest-based index structure analogous to the two-hop labeling. While their index construction induces large space cost. To scale CSP to large networks, many approximate CSP solutions are proposed by trading off solution quality for speed [1], [6], [19], [20]. The approximate method COLA [1] builds indices by partitioning the graph and selecting specific paths between boundary nodes. However, approximate solutions introduce errors and cannot be applied for the exact CSP. To accelerate the exact CSP algorithm, Lu et al. [7] propose Vine, which utilizes GPUs to parallelize the expansion process of the labeling algorithm. With the help of tens of thousands of GPU threads, Vine delivers state-of-the-art performance for the CSP problem. However, Vine expands labels without distinguishing their priorities and expands all labels at each iteration. In contrast, we devise highly efficient pruning strategies by taking advantage of SP information computed on the labels.

III. SP Enhanced Accelerations

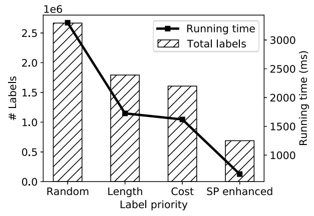
In this section, we first present the effect of label priority and present our devised SP-based label priorities. Then we introduce our pruning and teleporting strategies using SP. Finally, we briefly present our adaptions of the state-of-the-art SP algorithm. We provide a pseudo-code of SPEC in the Appendix Algorithm 1.

A. Label Priority

The labeling algorithm iteratively expands the label with the highest priority. Thus, it is critical to carefully design the label priority to distinguish promising labels and expand those early to find the optimal solution quickly. Existing priority values are set based on the length or cost from the source to the label nodes. In this work, we propose to incorporate



(a) Hops to target in searching



(b) Label number and running time

Fig. 3. Hops from the top label in the frontier to the target node w.r.t iterations and their totally expanded label number and running time comparison.

the SP information from a label node to the target into the label priority. The SP information enables navigation support for the expansion process to reach the target faster. Although computing SP brings additional costs, we find that the benefits significantly outweigh the overhead.

In the following, we present the motivating experiment to illustrate the significance of the label priority. For comparison, we list four priorities in the following.

1) Random priority. We assign a random value to a label as the priority, i.e., ignoring label differences as a baseline.
2) Length priority. For a label l_k, we adopt $l_k.l$ as the priority. The priority is given to those with small length [7], [8].
3) Cost priority. For a label l_k, we adopt $l_k.c$ as the priority. The priority is given to those with small cost [7], [8].
4) SP enhanced priority. We set the priority of l_k as $l_k.l+\bar{d}_{kv}$. $\bar{d}_{kv}$ is the shortest length from k to target v.

In Figure 3(a), we plot the hop number of the top element in the frontier to the target node w.r.t iterations on the NE graph. We observe that for random priority, length priority, and cost priority, the hop number fluctuates significantly, and the random priority has the largest fluctuation. In contrast, SP enhanced priority shows a stable and drastic decrease of the hop number. Hence, SP enhanced priority has much fewer iterations incurred to reach the target. Figure 3(b) shows the total number of expanded labels and average query time of the four priorities. SP enhanced priority achieves the smallest number of labels expanded as well as the lowest running time. The experiment confirms that label priority plays a significant role in efficient CSP processing and the SP information provides critical guidance on the label expansion process.

In addition to considering SP information for the length property, we can also consider SP information on edge costs. For node k and v, if we compute the shortest path using length, we obtain the **minimal length** $\bar{d}_{kv}$ and the corresponding **maximum cost** $\bar{c}_{kv}$ that k, v require. While we can also compute the shortest path using cost, in this way we obtain the **minimal cost** $\tilde{c}_{kv}$ that k, v require to have a path and the corresponding **maximum length** $\tilde{d}_{kv}$. Hence we could devise three SP enhanced label priorities as follows. All these priorities could be regarded as **estimations of a label's final length** when reaching the target node.

1) $P1 : l_k.l+\bar{d}_{kv}$, i.e., an optimistic estimation that prioritizes the labels with shorter lengths by ignoring the cost.

2) $P2 : l_k.l + \tilde{d}_{kv}$, i.e, a pessimistic estimation that prioritizes the labels with shorter lengths by using the smallest cost.
3) $P3 : l_k.l + \bar{d}_{kv} + (\tilde{d}_{kv} - \bar{d}_{kv}) \times \frac{\bar{c}_{kv}-(c-l_k.c)}{\bar{c}_{kv}-\tilde{c}_{kv}}$, a linear interpolation of $P1$ and $P2$.

Note that a smaller value indicates a higher priority. We will study the experimental performance of these priorities in Sec. VI-C. Besides, we discuss how to compute $\bar{d}_{kv}, \bar{c}_{kv}, \tilde{c}_{kv}, \tilde{d}_{kv}$ efficiently in Sec. III-C.

B. Pruning and Teleporting

In addition to the label priority, the SP information enables efficient pruning and teleporting strategies as well. Previous methods mainly utilize the dominance relation (Def. 2) for pruning [7], [8]. However, with the help of SP information, we maintain a **current best** [2] and could prune labels using the current best and SP lengths/costs to the target efficiently.

For a query (u, v, c) and a label l_k (located on node k), we define the **start point** of l_k as $(l_k.c + \tilde{c}_{kv}, l_k.l + \tilde{d}_{kv})$ and the **end point** of l_k as $(l_k.c + \bar{c}_{kv}, l_k.l + \bar{d}_{kv})$. Both are understood as coordinates in the 2-dimensional cost-length coordinate system. Note the definitions of $\bar{d}_{kv}, \bar{c}_{kv}, \tilde{d}_{kv}, \tilde{c}_{kv}$ are in Sec. III-A. A sketch of the start and end point is shown in Figure 4(a).

The pruning and teleporting are based on the locations of a label's start and end point in the cost-length coordinate system. We initialize the current best to $\tilde{d}_{u,v}$. For query (u, v, c), if the minimal cost $\tilde{c}_{u,v}$ exceeds c, the query is infeasible. If the maximum cost $\bar{c}_{u,v}$ doesn't exceed c, we immediately return $\bar{d}_{u,v}$ as the solution. Otherwise, the pruning and teleporting strategies are as follows.

1) For the endpoint, if $l_k.l + \bar{d}_{kv}$ is larger than the current best, we **prune** the label. Because $l_k.l + \bar{d}_{kv}$ is the smallest possible length from l_k and is already worse than the current best. (Figure 4(b), case 1.)
2) For the end point, if $l_k.l + \bar{d}_{kv}$ is smaller than the current best, and the cost $l_k.c + \bar{c}_{kv}$ is no larger than the cost limit, we update the current best to $l_k.l + \bar{d}_{kv}$, i.e., **teleporting** it to the target. Because it is the smallest length expanding from this label without exceeding the cost limit. (Figure 4(b), case 2.)
3) For the start point, if $l_k.c + \tilde{c}_{kv}$ is larger than the cost limit, we **prune** the label. Because $l_k.c + \tilde{c}_{kv}$ is the minimal cost expanding from this label and exceeds the cost limit. (Figure 4(c), case 3.)
4) For the start point, if $l_k.l + \tilde{d}_{kv}$ is smaller than the current best and $l_k.c + \tilde{c}_{kv}$ is no larger than the cost limit, we make a new label which is **teleported** to the target with length $l_k.l + \tilde{d}_{kv}$. The current best length is updated correspondingly. Note that we will not prune the label because we may find better solutions expanding from this label. (Figure 4(c), case 2.)

For the last case, i.e., the start point is in the area of Figure 4(c) case 1 and the end point is in the area of Figure 4(b) case 3, we preserve the label because it may expand to feasible paths which have smaller lengths than the current best without exceeding the cost limit as well. The correctness and complexity analysis are shown in the Appendix.

[2]Current best is the minimum CSP length of currently found feasible paths.

C. Efficient SP Processing for CSP

Our approach relies on efficient SP processing for computing $\bar{d}_{kv}$, $\bar{c}_{kv}$, $\tilde{d}_{kv}$, $\tilde{c}_{kv}$. If there exist multiple shortest length paths, the smallest cost on these paths should be returned. Likewise, if there exist multiple shortest cost paths, the smallest length should be returned. Although existing SP algorithms can return either the shortest length or the minimum cost efficiently, they only consider one metric. While we require both ones simultaneously.

We adapt the state-of-the-art SP algorithm H2H [21] and summarize our adaptations as follows:

1) We built two indices: one for original length information and the other for the corresponding cost.
2) In index construction and query processing, if the new length is smaller than the old one, both the length and cost will be updated. If the new length is equal to the old one, the cost will be updated only if the new cost is smaller than the old one. A mirror operation is applied to minimum cost index construction and query processing.

With the above adaptation, the cost of invoking SP in SPEC only takes up 5% to 10% of the total running time.

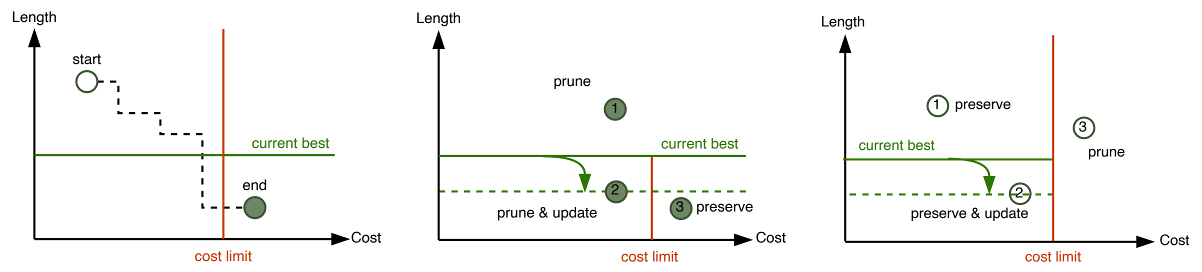
IV. Adaptive expansion control

Assigning priorities to labels facilitates distinguishing labels that are more promising. An intuitive strategy is to always expand the label with the highest priority, i.e., the top one in the frontier, which seems to be the most efficient. However, counter-intuitively, we find that this approach is often suboptimal because it can cause the search exploration process stuck in expanding the labels that seem promising but are actually unfeasible. Furthermore, expanding too many labels will lead to redundant labels generated. We elaborate on this intuition with an experiment as follows.

A. Observations

As shown in the toy example of Figure 5(a), S and T indicate the source and target node respectively. l_1, l_2 are labels in the 1st iteration, and l_3, l_4, l_5, l_6 are in the 2nd iteration. Labels are arranged such that higher ones have larger priorities. Besides, we assume l_5 is a label that will induce an update on the current best (highlighted in red), and l_3, l_4 will be pruned in future steps. At the current iteration, assuming the expansion number is set to 2, l_3 and l_4 are expanded. Then, all their descendant labels will be pruned finally and the current best will not decrease, which is unsatisfactory. If the expansion number is set to 4, the current best will be updated. However, more labels will be generated from l_6 at this step which may not be pruned as well even using the new pruning threshold. An appropriate expansion number should be 3 for this example.

Our exploratory experiments confirm the findings above. Figure 5(b) shows a comparison of the current best and frontier

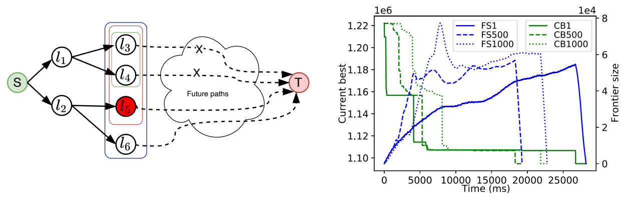


(a) Start and end point (b) For end point (c) For start point

Fig. 4. Pruning and teleporting strategies. The start and end correspond to two paths from a label node to the target node with the smallest cost and the smallest length respectively. The coordinates of start and end points could be computed assisted by the shortest path indices.

size for a query on NE with expansion number set to 1 (EP1), 500 (EP500), and 1000 (EP1000). We can find that EP1 finds the final CSP the slowest and has the largest running time. A large number of EP500 finds the CSP the fastest and has the smallest running time. While an even larger number EP1000 is worse than EP500 as well.

Although we may find an optimal expansion number for one query, it may not be suitable for queries with different nodes and cost constraints. Adjusting different static expansion strategies for different query sets is impractical because of the tuning cost. Instead, considering that each iteration's expansion impacts the overall search procedure which reveals a sequential decision making characteristic, we propose to learn an adaptive strategy in an end-to-end manner using the reinforcement learning method to serve different queries.



(a) Adaptive expansion intuition (b) Expansion number comparison

Fig. 5. The effect of different expansion numbers. In Figure 5(b), FS is the frontier size and CB is the current best.

B. RL-based expansion control

In the following, we elaborate our specifications of state, action, reward and optimization objective:

State s. We collect query's information and the current running statistics as informative signals. Based on the preliminary study, we list state features in Table I. QI and RI indicate query information and running information respectively.

Action a. We set the actions as discrete numbers that increase geometrically, i.e., 16, 32, 64, 128, etc. The action space can cater to various expansion demands for different queries.

Reward r. Our goal is to optimize the overall running time, which is constituted by each iteration's time. Hence, we take the negative running time during each iteration as the reward signal. Note that the reward decaying factor γ is set to 1. In this way, maximizing the cumulative reward in RL is consistent with our goal of minimizing the overall running time. Based on our experimental studies, we find that the running time of one iteration may fluctuate because of CPU overloads and

TABLE I
LIST OF STATE FEATURES

Type	Symbol	Description
QI	c	Cost limit
	L_{uv}	The shortest length between u, v
RI	T_{last}	Last iteration running time
	F	Frontier size
	ΔF	Frontier size change from last iteration
	S_{curr}	Current best value
	$l_{\text{top}}.priority$	Frontier top label's priority
	$l_{\text{top}}.l$	Frontier top label's length
	$l_{\text{top}}.c$	Frontier top label's cost
	ΔS_{top}	Length gap between top label's priority and current best
	c_{r}	Frontier top label's remaining cost

thread schedules. Hence, we combine the running time with the number of new unpruned labels as a combined reward to reduce variance. λ is set in [0.1, 0.5] empirically.

$$R_{\text{iter}} = -T_{\text{iter}} - \lambda N_{\text{iter}} \tag{1}$$

Optimization objective. We adopt the Deep Q-network (DQN) [22] as the RL agent. For a transition tuple $(s, a, r, s^{'})$, the optimization objective is the Bellman equation error using target network and policy network. The target network's estimation is $V_{\text{target}} = r + \max_{a'} Q_{\text{target}}(s^{'}, a^{'})$, while the policy network's estimation is $V_{\text{policy}} = Q_{\text{policy}}(s, a)$. We adopt smooth L1 loss as the error metric as follows, where β is set to 1.0 by default.

$$L = \begin{cases} 0.5(V_{\text{target}} - V_{\text{policy}})^2/\beta, & |V_{\text{target}} - V_{\text{policy}}| < \beta \\ |V_{\text{target}} - V_{\text{policy}}| - 0.5\beta, & \text{otherwise} \end{cases} \tag{2}$$

C. Parallel optimization

Once the expansion number is determined at each iteration, we could parallelize the expansion process to boost the efficiency based on modern multi-core CPUs. We devise a thread pool to manage parallel threads. Additionally, we make two optimizations for the parallel implementation.

(1) Each thread maintains a local non-dominant list, which stores newly unpruned labels in this thread, instead of directly writing back to the global list. This approach reduces synchronization overheads and memory write contention when parallel threads write labels back to the global non-dominant list.

(2) After all threads expand their local non-dominant lists, we merge and prune new labels from all threads by adopting the dominance relation. Because a parallel expansion will expand

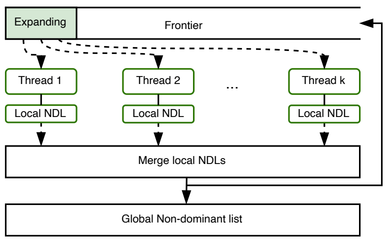


Fig. 6. Parallel execution sketch. Local NDL indicates a non-dominant label list maintained in each thread independently.

TABLE II
List of dataset statistics.

Type	Dataset	—V—	—E—	Size
Road	New York City (NY)	264346	730100	14MB
	Florida (FLA)	1070376	2687902	55MB
	Northwest USA (NW)	1207945	2820774	60MB
	Northeast USA (NE)	1524453	3868020	81MB
Internet	Router Connection (Router)	2113	13264	1MB
	Tech-Internet (Internet)	40164	170246	3MB
IF	Open flights (Flights)	2905	31290	1MB
	Power grid (Power)	4941	13188	1MB

more labels than a single thread as some labels dominated in a single thread may not be dominated in a parallel scenario [7]. With the merge step, we avoid pushing back redundant labels into the frontier.

The execution flow is sketched in Figure 6. We will describe more detailed settings in Sec. V, e.g., thread pool size, the overload of each thread, etc.

V. Experimental Setup

Environment. We run SPEC and other baselines that only require CPUs on a machine with an AMD EPYC 7643 48-Core processor. Note that Vine runs on GPUs, and the GPU specification is NVIDIA Tesla P100 GPU with 12GB global memory and 56SMs. Vine is complied with NVIDIA's *nvcc* compiler (version 7.5) [3], and other CPU algorithms are complied with *gcc 9.3* with *O3* flag. In all experiments, the graph data is pre-loaded into the CPU or GPU memory. We run all experiments five times and average the results.

Datasets. We evaluate SPEC and the baselines on 8 read-world graph datasets as listed in Table II. These datasets are used in previous works [1], [7]. We include three types of graphs: (1) road network, (2) internet topology, and (3) infrastructure network. Road networks are obtained from the 9th DIMAC Challenge [23]. Internet and infrastructure datasets are obtained from the network data repository [24]. On road networks, length and cost indicate road length and travel cost. For internet and infrastructure networks, only the topology information is obtainable, without length and cost attached. Hence for each edge, we randomly generate two integers in the range $[100, 10000]$ as length and cost respectively.

Query sets. For road networks, we adopt the same query set as used in [7]. While for internet and infrastructure networks, we adopt the following rule to generate queries, which is used in previous works [1], [7]. We firstly generate query node pairs randomly and then divide them into three sets Q1, Q2, and Q3, according to their hop lengths. The ranges are $[d_{min}/8, d_{min}/4]$, $[d_{min}/4, d_{min}/2]$, and $[d_{min}/2], +\infty]$, for Q1, Q2, and Q3 respectively. d_{min} is the graph diameter. $Q1$ is designed to be with a relatively small search space, Q2 is moderate, and Q3 is large. Constraints are also generated randomly between the minimum and the maximum cost of the two query nodes. Each query set contains 100 queries. We report the average running time of each query set.

Baselines. We compare the following methods that could solve the exact CSP problem: (1) our single thread SPEC (**SPEC-s**), (2) our multi-thread SPEC (**SPEC-m**), (3) labeling algorithm (**LA**), [8] (4) **Vine**, [7], (5) k-th shortest path algorithm (**KSP**) [17], [25], [26]. KSP solves the exact CSP algorithm by finding the next SP until the solution satisfies the cost limit. We implement the labeling algorithm and adopt the label's length as the priority. For other baselines, we use their released codes and default settings.

SPEC settings. For all experiments, we set $l_k.l + \bar{d}_{kv}$ as the default priority unless specified otherwise. For the RL controller, we set 7 discrete actions, i.e., 16, 32, 64, 128, 256, 512, 1024. We set the replay buffer size to 5×10^5 steps, and the target network update frequency to 2×10^4 steps. Here each step represents one iteration during solving a query. For our multi-thread SPEC, we set the thread pool size to 70, and each thread's default *overload*, i.e., the number of labels distributed to each thread for expansion, is set to 10.

Besides, we list the index space and build time of all datasets in Appendix. We also elaborate RL training scheme, Q-network architectures, and inference time in Appendix.

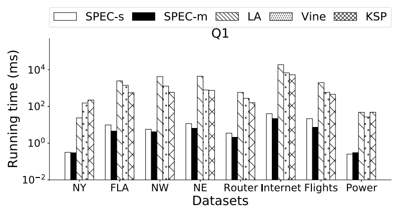
VI. Experimental Results

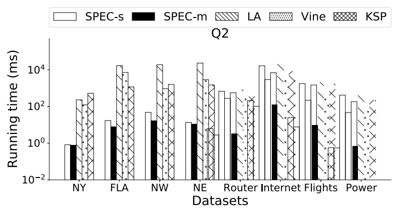
In this section, we report experiment results, including overall performance comparison (Sec. VI-A), pruning and teleporting effect (Sec. VI-B), label priorities (Sec. VI-C), adaptive expansion control effect (Sec. VI-D), and parallel overload effect (Sec. VI-E).

A. Overall Performance Comparison

Figure 7 shows all methods' average running time on each query set. Note that the y-axis is set in a logarithm scale. For small query sets like Q1, the speedup of SPEC-s over Vine ranges over several dozens to several hundreds. For example, SPEC-s achieves the smallest speedup of $27\times$ on the Flights graph and the largest speedup of $485\times$ on the NY graph. The speedup of SPEC-s over Vine on Q1 is $162\times$ averaged on all datasets. For large queries like Q3, the relative speedup of SPEC-s is smaller than that on Q1. For example, SPEC-s achieves $3.2\times$ on NW and $125\times$ on FLA. The average speedup of SPEC-s is $40\times$ over all datasets on Q3. For moderate Q2, the speedup also lies in the middle of Q1 and Q3, i.e., the smallest and the largest speedup is $9\times$ on Flights and $418\times$ on FLA respectively with an average of $118\times$. For other baselines, the running time of LA is prohibitively large even for moderate graph and query sets, as shown in Figure 7(b) and surges for large query sets as shown in Figure 7(c). For

[3]Newer NVCC version results in errors for Vine.


(a) Q1


(b) Q2

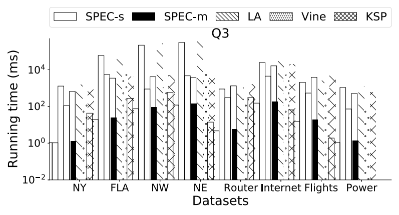

(c) Q3

Fig. 7. Overall performance comparison for three query sets on all datasets.

KSP, the average running time is slightly better than that of Vine on small Q1 sets while much larger on moderate and large query sets.

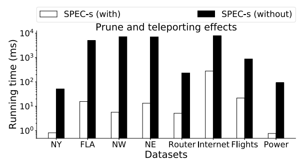
Parallel Performance. We can find that SPEC-m obtains $3\times$ to $5\times$ speedup over SPEC-s for queries on large datasets, which require substantial computation time. For example, on the NE graph with Q3 set, the average time of SPEC-s is 690ms, while SPEC-m is 142ms, achieving a $4.85\times$ speedup. For small datasets, the speedup decreases. This is because small datasets have short running times and the parallelism benefit becomes less significant. For example, on the Power graph with Q3 set, the running time of SPEC-m and SPEC-s is 1.31ms and 2.26ms respectively, with a minor speedup. We find that the overload of each thread also has a considerable influence on the improvements. We will discuss the effects of this parameter in Sec. VI-E.

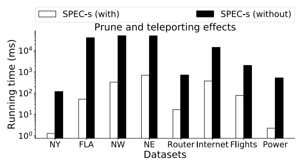
B. Impact of Pruning Strategies

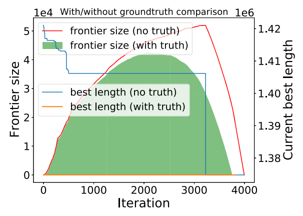
In this section, we discuss the effects of our pruning and teleporting strategies in Sec. III-B. We compare the single thread SPEC-s performance with and without these pruning and teleporting strategies. Two versions are denoted by SPEC-s (with) and SPEC-s (without) respectively. We conduct comparisons on all datasets for Q2 and Q3. Results are shown in Figure 8. We can find that our pruning and teleporting strategies have a substantial influence on the overall running time on all datasets. The SPEC-s (with) runs about 2 to 3 magnitudes faster than SPEC-s (without) depending on datasets. For example, speedups for Q2 on all datasets of SPEC-s (with) over SPEC-s (without) are $64\times$, $317\times$, $1263\times$, $525\times$, $45\times$, $28\times$, $40\times$, $121\times$ respectively, with an average of $301\times$. While the average speedup for Q3 is $179\times$. This comparison reveals the effectiveness of our pruning and teleporting strategies, especially for large graphs like NW and NE, on which many unpromising labels are pruned at the early stages of the expansion process.

To illustrate the pruning effects more intuitively, we show two queries' statistics (on the NE graph) in Figure 8(c) and 8(d). Figure 8(c) and 8(d) plot the frontier size and current best w.r.t iteration number, corresponding to the left and right y-axis, respectively. For each query, we have two settings. The *without truth* indicates a regular setting of SPEC-s, i.e., gradually updating the current best. While the *with truth* means we set the current best to the groundtruth CSP value of that query at the beginning. For query 1 in Figure 8(c), we can find that the frontier size of *without truth* and *with truth* are similar until the half of iterations. Then *without truth* has a slightly larger frontier size than *with truth*. The frontier size of *without truth* decreases rapidly after finding the final CSP solution at about 3200 iterations. The final iteration numbers of *without truth* and *with truth* have a considerably small gap, i.e., about 300 iterations w.r.t a total of 4000, and the overall running time is analogous as well. For query 2 in Figure 8(d), the red line (*without truth*) is quite close to the green area (*with truth*), which means that the gradually updating of current best has pruned almost all labels that could be pruned by the final ground truth length.

Figure 8(c) and Figure 8(d) reveal that: (1) there still exist considerable labels that cannot be pruned in the search procedure even if we know the groundtruth for pruning at the beginning; (2) our pruning and teleporting strategies lead to a pruning performance comparable with that of using the groundtruth for pruning initially. Hence, in order to take advantage of available computing resources like parallelism, a promising direction is to enable bidirectional expansion , i.e., expanding from both the source and target, so that a better pruning strategy and performance can be devised. We leave this as future work.


(a) Performance on Q2


(b) Performance on Q3


(c) Query case 1

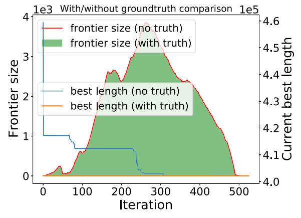
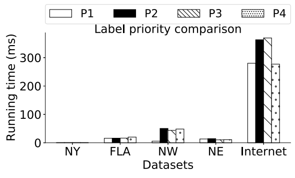

(d) Query case 2

Fig. 8. Pruning/teleporting performance comparison and illustration of two cases on the NE graph between with and without ground-truth CSP value initially. For Figure 8(c) and 8(d), frontier size and length correspond to the left and right y-axis respectively.

C. Label Priority Effects

In this section, we investigate the effects of different label priorities. We compare 4 priorities, i.e., $P1$, $P2$, $P3$ are three


(a) Comparison on Q2

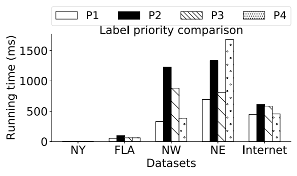

(b) Comparison on Q3

Fig. 9. Label priorities comparison.

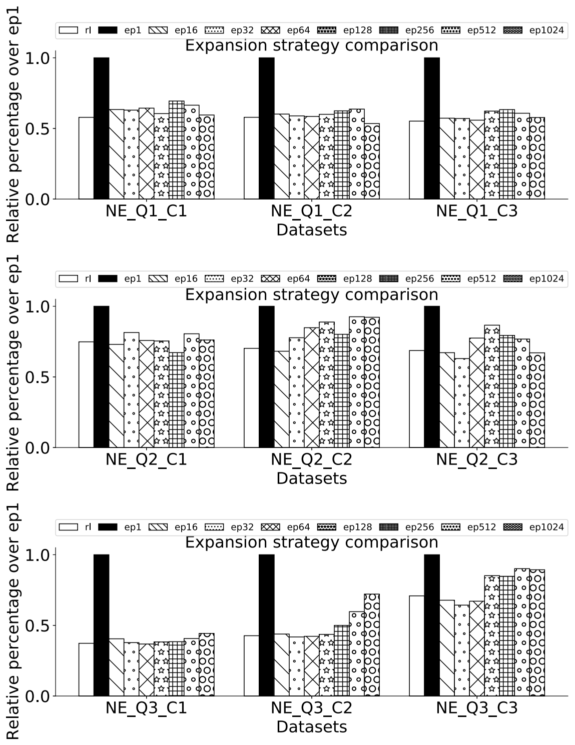
SP enhanced priorities discussed in Sec. III-A. We set $P4$ as the label length, i.e., $l_k.l$. The results on five relatively large networks are shown in Figure 9. Comparisons are conducted for Q2 (Figure 9(a)) and Q3 (Figure 9(b)) on the NE graph with the single thread SPEC-s. Note that all variants include the pruning and teleporting strategies.

We can see that for easy queries like Q2 on NY, FLA, and Internet, the difference between $P1$ and $P4$ is not so obvious. However, for hard queries like Q3 on NE, the gap between SP enhanced priority $P1$ and length-based $P4$ becomes significant, e.g., SPEC-s ($P1$) runs in less than 750ms on average while SPEC-s ($P4$) exceeds 1700ms. $P1$ achieves the best performance for most cases. Another interesting observation is the relatively inferior performance of $P2$ and $P3$, which are also SP enhanced priorities. We can find $P3$ performs slightly worse than $P1$, while $P2$ is much worse than $P1$. Note that $P1$ is an optimistic estimation of a label's final length ignoring the future cost needed. The reason comes from the fact that a real CSP solution is *near* the SP path, i.e., CSP and SP may have some overlapped path nodes. Hence the SP length priority $P1$ works well as a signal for selecting promising labels that will probably appear on the final CSP path, which is also supported by Figure 3(a).

D. Adaptive Control Effects

In this section, we study the performance of adaptive expansion control compared with static strategies. We consider 8 heuristic settings denoted by epi, where i is set to 1, 16, 32, 64, 128, 256, 512, 1024, respectively. i indicates how many labels are expanded in each iteration. All experiments are based on the single thread SPEC-s. To compare rl with static strategies, we vary the queries by showing results for Q1, Q2, and Q3 with small cost C1, medium cost C2 and large cost C3. C1, C2, and C3 refer to the 20%, 50%, and 80% values within the cost range specified in the experiment setup, respectively. We demonstrate the results on the NE graph as it is the largest dataset in our experiments and the dataset comes with real road length and travel cost. The running time is divided by that of $ep1$ to show the relative running time. A smaller value indicates a better performance compared with $ep1$. For all settings, we use the same RL controller. Results are shown in Figure 10(a).

We can find that different query sets prefer different static strategies. For example, on Q2_C1, $ep128$ achieves the best performance. On Q2_C2, $ep16$ is the best. While on Q3_C3, $ep32$ outperforms others. For most cases, the rl strategy achieves comparable performance with the best static strategy


(a) Expansion strategy comparison

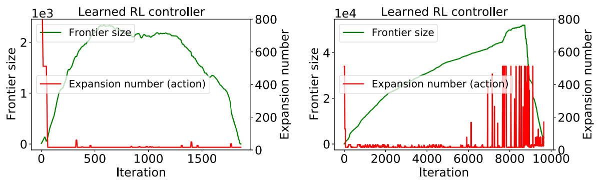

(b) RL actions (case 1) (c) RL actions (case 2)

Fig. 10. Expansion strategies and learned adaptive expansion numbers.

and outperforms the best static strategy on several cases, e.g., Q1_C1 and Q1_C3. In all cases, the rl strategy outperforms the $ep1$ significantly.

To understand the expansion strategy learned by the RL controller more intuitively and shed some light into the expansion procedure, we plot the actions of the controller and the frontier size w.r.t iterations for two query cases on NE graph, as shown in Figure 10(b) and 10(c). The green line indicates the frontier size and the red line shows the expansion number given by the RL controller. We can find that the expansion controller learns to give different strategies for different queries. For case 1, the expansion number is large at the beginning and remains at a low level till termination. For case 2, the expansion number firstly decreases rapidly as in case 1. The controller outputs large actions near the end of the search procedure. We can find that the increase of actions is accompanied by the rapid decrease of frontier size. Because the final CSP solution may have been found and most of the remaining labels in the frontier could be pruned. Therefore the controller prefers large actions to decrease the total iterations and the overall running time. Based on the strategies learned by the expansion controller, the expansion number should be relatively small when the frontier size is large for heavy queries. Once high-quality solutions are found for pruning, we may increase the expansion number for quick termination.

E. Influence of Parallel Thread Overload

We study each thread's overload in SPEC-m, i.e., number of labels distributed to each thread to expand, which is an important parameter affecting the number of threads used and parallelization cost. We plot the average running time and the number of threads utilized under different overload settings for Q3 on the NE graph in Figure 11. *Single* indicates SPEC-s, while *OLi* means SPEC-m with overload set to i.

From Figure 11, we can find that SPEC-m with overload set to 2, 5, or 10 have much less running time compared with the single thread version, i.e., about 150ms vs 700ms. We can also notice that too many or too few threads will not obtain the best result. For example, *OL1* uses 70 threads while requiring about 290ms, while *OL50* uses about 2 threads with an average of 450ms. Small overload leads to a non-negligible thread invocation cost compared to the time of expanding the assigned labels. A large overload, e.g., *OL30, OL50*, under-utilizes thread resources. Hence, in our experiments, we set 10 as the default overload.

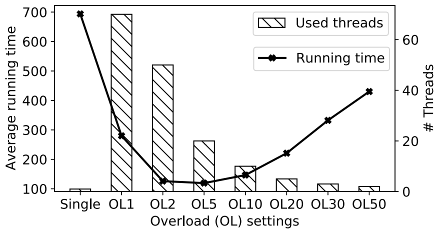


Fig. 11. Effects of parallel thread overload for Q3 on NE.

VII. Conclusions

In this paper, We devise highly efficient pruning strategies by leveraging SP information. Our SP enhanced label priority facilitates finding better feasible solutions faster, which could be used to prune more sub-optimal labels. Furthermore, we devise the adaptive RL-based expansion controller, which improves the overall performance further compared with static expansion strategies. Single CPU thread SPEC-s outperforms current state-of-the-art GPU-accelerated Vine by more than one order of magnitude. The parallelized SPEC-m outperforms SPEC-s about 3 to 5 times further. As future works, we plan to study the processing of multiple CSPs under concurrent graph processing frameworks [27].

Acknowledgements This research work is funded by the National Nature Science Foundation of China under Grant 61971283, and Shanghai Municipal Science and Technology Major Project under Grant 2021SHZDZX0102. Yuchen Li's work is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (Award No.: MOE2019-T2-2-065).

References

[1] S. Wang, X. Xiao, Y. Yang, and W. Lin, "Effective indexing for approximate constrained shortest path queries on large road networks," in *VLDB*, 2016.

[2] A. Alanazi and K. Elleithy, "Real-time qos routing protocols in wireless multimedia sensor networks: Study and analysis," *Sensors*, vol. 15, no. 9, pp. 22 209–22 233, 2015.

[3] I. Awan and M. Younas, "Towards qos in internet of things for delay sensitive information," in *ICMWIS*, 2013.

[4] K. Dorling, J. Heinrichs, G. G. Messier, and S. Magierowski, "Vehicle Routing Problems for Drone Delivery," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 1, pp. 70–85, 2017.

[5] L. D. P. Pugliese and F. Guerriero, "A survey of resource constrained shortest path problems: Exact solution approaches," *Networks*, vol. 62, no. 3, pp. 183–200, 2013.

[6] R. Hassin, "Approximation schemes for the restricted shortest path problem," *Mathematics of Operations research*, vol. 17, no. 1, pp. 36–42, 1992.

[7] S. Lu, B. He, Y. Li, and H. Fu, "Accelerating exact constrained shortest paths on gpus," in *VLDB*, 2020.

[8] P. Hansen, "Bicriterion path problems," in *Multiple criteria decision making theory and application*. Springer, 1980, pp. 109–127.

[9] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen, "An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems," *Networks*, vol. 44, no. 3, pp. 216–229, 2004.

[10] S. Irnich and G. Desaulniers, "Shortest path problems with resource constraints," in *Column generation*. Springer, 2005, pp. 33–65.

[11] B. C. Dean, "Algorithms for minimum-cost paths in time-dependent networks with waiting policies," *Networks: An International Journal*, vol. 44, no. 1, pp. 41–46, 2004.

[12] V. Jeauneau, L. Jouanneau, and A. Kotenkoff, "Path planner methods for uavs in real environment," *IFAC-PapersOnLine*, vol. 51, no. 22, pp. 292–297, 2018.

[13] H. C. Joksch, "The shortest route problem with constraints," *Journal of Mathematical Analysis and Applications*, vol. 14, no. 2, pp. 191–197, 1966.

[14] G. Y. Handler and I. Zang, "A dual algorithm for the constrained shortest path problem," *Networks*, vol. 10, no. 4, pp. 293–309, 1980.

[15] K. Mehlhorn and M. Ziegelmann, "Resource constrained shortest paths," in *European Symposium on Algorithms*. Springer, 2000, pp. 326–337.

[16] J. Brumbaugh-Smith and D. Shier, "An empirical investigation of some bicriterion shortest path algorithms," *European Journal of Operational Research*, vol. 43, no. 2, pp. 216–224, 1989.

[17] T. Chondrogiannis, P. Bouros, J. Gamper, U. Leser, and D. B. Blumenthal, "Finding k-shortest paths with limited overlap," *The VLDB Journal*, vol. 29, no. 5, pp. 1023–1047, 2020.

[18] Z. Liu, L. Li, M. Zhang, W. Hua, P. Chao, and X. Zhou, "Efficient constrained shortest path query answering with forest hop labeling," in *ICDE*. IEEE, 2021, pp. 1763–1774.

[19] D. H. Lorenz and D. Raz, "A simple efficient approximation scheme for the restricted shortest path problem," *Operations Research Letters*, vol. 28, no. 5, pp. 213–219, 2001.

[20] G. Tsaggouris and C. Zaroliagis, "Multiobjective optimization: Improved FPTAS for shortest paths and non-linear objectives with applications," *Theory of Computing Systems*, vol. 45, no. 1, pp. 162–186, 2009.

[21] D. Ouyang, L. Qin, L. Chang, X. Lin, Y. Zhang, and Q. Zhu, "When Hierarchy meets 2-hop-labeling: Effiicient shortest distance queries on road networks," in *SIGMOD*, 2018.

[22] V. Mnih1 and H. Koray Kavukcuoglu1*, David Silver1*, Andrei A. Rusu1, Joel Veness1, Marc G. Bellemare1, Alex Graves1, Martin Riedmiller1, Andreas K. Fidjeland1, Georg Ostrovski1, Stig Petersen1, Charles Beattie1, Amir Sadik1, Ioannis Antonoglou1, "Human-level control through deep reinforcement learning," *Nature*, no. 7540, pp. 2315–2321, 2016.

[23] C. Demetrescu, A. V. Goldberg, and D. S. Johnson, *The shortest path problem: Ninth DIMACS implementation challenge*. American Mathematical Soc., 2009, vol. 74.

[24] R. Rossi and N. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015.

[25] T. Chondrogiannis, P. Bouros, J. Gamper, and U. Leser, "Exact and approximate algorithms for finding k-shortest paths with limited overlap," in *ICEDT*, 2017.

[26] T. Chondrogiannis, P. Bouros, J. Gamper, and Leser, “Alternative routing: k-shortest paths with limited overlap,” in *SIGSPATIAL*, 2015, pp. 1–4.

[27] S. Lu, S. Sun, J. Paul, Y. Li, and B. He, “Cache-efficient fork-processing patterns on large graphs,” in *SIGMOD*, 2021, pp. 1208–1221.

VIII. The SPEC algorithm and index statistics

We conduct a parallel optimization on our adapted H2H algorithm with a 60-threads thread pool. We report the index space and time of the adapted implementation for all datasets in Table III. The SPEC is shown in Algorithm 1.

TABLE III
List of index space and index time on all datasets.

	Dataset							
Properties	NY	FLA	NW	NE	Router	Internet	Flights	Power
Graph size	14MB	55MB	60MB	81MB	1MB	3MB	1MB	1MB
Index size	1.2G	4.9G	6.9G	13G	4.7M	177M	6.2M	3.4M
Index time	212s	441s	553s	280s	1s	4s	1s	1s

IX. Detailed settings and proofs

Algorithm 1: SPEC algorithm

```
Input  : the query $q = (u, v, c)$, graph $G$,
         AdaptedIndex $\Phi$.
Output: the shortest path length $d$ between $u$ and $v$
         with cost not exceed $c$.
1  Initialize $F$ and global label list $GL$, current best $cb$;
2  while $F$ is not empty do
3      $State$ = FeatureExtract($F$, G);
4      $Num$ = RLAgent($State$);
5      $New\_labels$ = Expand($F[0 : Num - 1]$, $G$);
6      Initialize $new\_labels$;
7      forall label $l_k \in New\_labels$ do
8          $\bar{d}, \bar{c}, \tilde{d}, \tilde{c}$ = AdaptedH2H($\Phi$, $k$, $v$, $c$);
9          $prune\_result$ = Prune($l_k$, $GL$, $\bar{d}, \bar{c}, \tilde{d}, \tilde{c}$, $c$, $cb$);
10         if $prune\_result.preserve$ is $True$ then
11             AssignPriorities($l_k$, $\bar{d}, \bar{c}, \tilde{d}, \tilde{c}$ );
12             $new\_labels$.append($l_k$);
13         UpdateCurrentBest($cb$, $prune\_result$);
14     end
15     InsertFrontier($F$, $new\_labels$);
16     InsertGlobalList($GL$, $new\_labels$);
17 end
18 return $d, c$;
```

For the RL controller, we use a MLP with two hidden layers as Q-net’s architecture. The hidden dimensions are both 64, with input dimension 11 and output dimension 7. In our experiments, the average inference time of RL controller is about *10 macro-seconds*. Note that the inference time is independent with datasets.

Theorem 1. *Given a query $q = (u, v, c)$, The algorithm in Alg. 1 could find the shortest path P_q^* with cost not exceeding c (if exists).*

Proof. Since our algorithm is based on the labeling algorithm, which will find all non-dominate labels on all nodes, we only need to prove that the label expansion priority and our pruning strategies will not discard the best label on v, i.e., P_q^*. Assuming the best path $P_q^* = (v_0, v_1, v_2, \cdots, v_k)$ where $v_0 = u$ and $v_k = v$, then for each prefix path $P_q^*[: i] = (v_0, v_1, \cdots, v_i)$, we need to proof that the label corresponding to $P_q^*[: i]$ will be preserved at node v_i after the algorithm termination. We first prove that each $P_q^*[: i]$ will not be dominated by other possible labels on v_i. This can be proven by contradiction. If $P_q^*[: i]$ is dominated by P', then $P' + P_q^*[i+1 :]$ becomes a better path, which contradicts the definition of P_q^*. Firstly, we can find that no matter what labels priority we adopt, all labels $P_q^*[: 0$, $P_q^*[: 1]$, $\cdots$, $P_q^*[: k]$ will be created and preserved on nodes v_0, v_1, $\cdots$, v_k, correspondingly, if no pruning strategies are used. Because the algorithm will expand each label in the frontier until empty by exploring all labels’ neighbors. Furthermore, all these labels $P_q^*[: i]$ will not be dominated by others, as stated above. Hence they will be preserved finally. Secondly, we need to prove that our devised pruning strategies will not prune any $P_q^*[: i]$. Let $P_{\text{curr\_best}}$ denotes the current best solution found during the algorithm operating. For any label $P_q^*[: i]$, we can conclude that: (1) For pruning strategy 1 (Figure 4(b), case 1.), $P_q^*[: i].l + \bar{d}_{v_i,v_k} \leq P_q^*.l \leq P_{\text{curr\_best}}.l$. Because $\bar{d}_{v_i,v_k}$ is the minimal length from v_i to v_k, which is no larger than $P_q^*[i :].l$. Since P_q^* is the best solution from v_0 to v_k, its length is no larger than $P_{\text{curr\_best}}.l$ at any time. Hence, pruning strategy 1 will not discard any label $P_q^*[: i]$. (2) For pruning strategy 2 (Figure 4(c), case 3), $P_q^*[: i].c + \tilde{c}_{v_i,v_k} \leq P_q^*.c \leq c$. Because $\tilde{c}_{v_i,v_k}$ is the smallest cost from v_i to v_k, which is no larger that $P_q^*[i :].c$. Since P_q^* is a feasible solution from v_0 to v_k, its cost is no larger than c. Hence pruning strategy 2 will not discard any label $P_q^*[: i]$ as well. (3) For teleporting strategies, it will not prune existing labels but only update the $P_{\text{curr\_best}}$. As indicated by the above statement, $P_{\text{curr\_best}}$ will not prune $P_q^*[: i]$ in strategy 1. Hence, our pruning strategies will preserve all labels (prefix paths) on the optimal path P_q^*. After the algorithm termination, the best label (the shortest length with a legal cost) on node v_k will be selected as P_q^*. ☐

For the worst-case complexity of Alg. 1, all pruning strategies are ineffective and the controller always decides to expand all labels in the frontier at each iteration. Our algorithm degrades to the naive labeling algorithm and has the same time complexity [8]. Hence the complexity of Alg. 1 is $\mathcal{O}(l_{max}mn \cdot \log l_{max}n)$ where m (resp. n) is the number of edges (resp. nodes), and l_{max} is the maximum edge length.

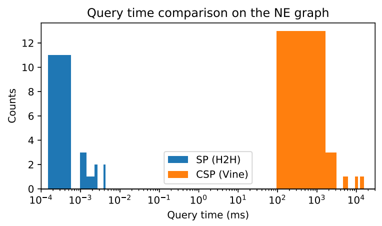


Fig. 12. Query time comparison of state-of-the-art SP method (the H2H [21]) and CSP method (the Vine [7]) on the NE graph for a moderate query set with 20 queries. SP is about 5 to 6 orders of magnitude faster than CSP.