Bounded-Suboptimal Weight-Constrained Shortest-Path Search via Efficient Representation of Paths

Han Zhang[1], **Oren Salzman**[2], **Ariel Felner**[3], **T. K. Satish Kumar**[1], **Sven Koenig**[1]

[1] University of Southern California
[2] Technion - Israel Institute of Technology
[3] Ben-Gurion University
zhan645@usc.edu, osalzman@cs.technion.ac.il, felner@bgu.ac.il, tkskwork@gmail.com, skoenig@usc.edu

Abstract

In the Weight-Constrained Shortest-Path (WCSP) problem, given a graph in which each edge is annotated with a cost and a weight, a start state, and a goal state, the task is to compute a minimum-cost path from the start state to the goal state with weight no larger than a given weight limit. While most existing works have focused on solving the WCSP problem optimally, many real-world situations admit a trade-off between efficiency and a suboptimality bound for the path cost. In this paper, we propose the bounded-suboptimal WCSP algorithm WC-A*pex, which is built on the state-of-the-art approximate bi-objective search algorithm A*pex. WC-A*pex uses an approximate representation of paths with similar costs and weights to compute a $(1+\varepsilon)$-suboptimal path, for a given ε. During its search, WC-A*pex avoids storing all paths explicitly and thereby reduces the search effort while still retaining its $(1+\varepsilon)$-suboptimality bound. On benchmark road networks, our experimental results show that WC-A*pex with $\varepsilon = 0.01$ (i.e., with a guaranteed suboptimality of at most 1%) achieves a speed-up of up to an order of magnitude over WC-A*, a state-of-the-art WCSP algorithm, and its bounded-suboptimal variant.

Introduction and Related Work

In the Weight-Constrained Shortest-Path (WCSP) problem, given a graph in which each edge is annotated with a cost and a weight, a start state, and a goal state, the task is to compute a minimum-cost path from the start state to the goal state with weight no larger than a given weight limit. The WCSP problem appears in many real-world applications. In an electric vehicle domain, the graph represents a road network, and each edge is annotated with a cost corresponding to driving time and a weight corresponding to battery consumption (Baum et al. 2015). A desired route minimizes the driving time without depleting the battery. In a cycling domain, the graph represents a road network, and each edge is annotated with a cost corresponding to cycling time and a weight corresponding to climbing altitude gain (Storandt 2012). A desired route minimizes the cycling time without exceeding a given limit on the total climbing altitude gain.

Combinatorially, the WCSP problem also appears as a subproblem in the context of column generation methods used for solving other problems, such as the shift scheduling problem (Cabrera et al. 2020) and the virtual network embedding problem (Rost 2019). Although many path-finding problems are tractable, the WCSP problem is NP-hard to solve optimally, i.e., it is NP-hard to compute the minimum-cost path within the given weight limit (Handler and Zang 1980; Lorenz and Raz 2001).

The WCSP problem is similar to the Bi-Objective Shortest-Path (BOSP) problem, where each edge is annotated with two costs. Although the task in the BOSP problem is different from the task in the WCSP problem, several techniques of BOSP algorithms can be carried over to WCSP algorithms by treating the weight as the second cost while being cognizant of the weight limit. In fact, WC-A* (Ahmadi et al. 2022a) is a state-of-the-art WCSP algorithm that draws inspiration from BOSP algorithms. WC-A* and its bi-directional variant WC-BA* (Ahmadi et al. 2022b) have been shown to outperform previous state-of-the-art WCSP algorithms Bi-pulse (Cabrera et al. 2020) and RC-BDA* (Thomas, Calogiuri, and Hewitt 2019) in terms of runtime by up to two orders of magnitude on road networks (Ahmadi et al. 2022a,b).

While the algorithms mentioned above focus on solving the WCSP problem optimally, many real-world situations admit—or even encourage—a trade-off between efficiency and a suboptimality bound for the path cost. A bounded-suboptimal WCSP algorithm computes a $(1+\varepsilon)$-suboptimal path, for a given ε. A $(1+\varepsilon)$-suboptimal path has a cost that is no larger than $(1+\varepsilon)$ times the minimum path cost and a weight that is no larger than the weight limit.

There is relatively little work on solving the WCSP problem with bounded-suboptimality guarantees. Cabrera et al. (2020) suggest a general method for converting an optimal WCSP algorithm to a bounded-suboptimal one by terminating the search immediately after the cost of the incumbent solution (i.e., the best solution that the WCSP algorithm has found so far) is proven to be within the given suboptimality bound. Other works on bounded-suboptimal WCSP algorithms (Lorenz and Raz 2001; Ergun, Sinha, and Zhang 2002) are typically based on fully polynomial-time approximation schemes, whose runtimes are polynomial in the size of the graph and $1/\varepsilon$. Unfortunately, these algorithms are still impractical for large graphs, such as road networks, that often have millions of states.

There are many existing works on bounded-suboptimal search algorithms for (unconstrained) shortest-path problems. These algorithms include WA* (Pohl 1970), focal search (Pearl and Kim 1982), and explicit estimation search (Thayer and Ruml 2011). While speeding up the search by allowing suboptimality is intuitive, it is unclear how to do so efficiently for the WCSP problem.

In this paper, we propose a novel bounded-suboptimal WCSP algorithm called WC-A*pex. WC-A*pex takes a WCSP instance and an $\varepsilon \geq 0$ as input and computes a $(1+\varepsilon)$-suboptimal path. WC-A*pex uses techniques from A*pex (Zhang et al. 2022a), a state-of-the-art approximate BOSP algorithm. Unlike other WCSP algorithms, WC-A*pex uses a clever data structure to merge paths with similar costs and weights efficiently (instead of storing them explicitly) during the course of its search. Since paths correspond to search nodes, the merged representation of similar paths reduces the number of node expansions and thereby the overall search effort of WC-A*pex. WC-A*pex thus uses a different technique to speed up the search from existing bounded-suboptimal search algorithms, most of which rely on node expansion orders to guide the search to quickly find a bounded suboptimal solution.

We evaluate WC-A*pex experimentally with different suboptimality bounds and compare it with existing WCSP algorithms on benchmark road networks with 1 to 14 million states and 2 to 34 million edges. The competing WCSP algorithms include WC-A* and our adaptation of it to the bounded-suboptimal variant WC-A*-ε, which terminates the search immediately after the incumbent solution is proven to be $(1+\varepsilon)$-suboptimal. Our experimental results show that WC-A*pex substantially outperforms WC-A* and WC-A*-ε in terms of runtime although they are also based on BOSP algorithms. This demonstrates the power of the merged representation of similar paths used in WC-A*pex. Even with $\varepsilon = 0.01$ (i.e., with a guaranteed suboptimality of at most 1%), WC-A*pex achieves an order-of-magnitude speed-up over WC-A* and WC-A*-ε on the largest road network. In comparison, WC-A*-ε with the same value of ε achieves less than 20% speed-up over WC-A*.

Terminology and Problem Definition

In this section, we formally define the WCSP and BOSP problems. To allow for a uniform notation, we define the cost of an edge as a pair of numbers. In the context of the WCSP problem, the first number indicates the cost, and the second number indicates the weight. In the context of the BOSP problem, both numbers represent the cost.

We use **boldface** font to denote pairs and p_i, $i \in \{1,2\}$, to denote the i-th component of a pair $\mathbf{p}$. The addition of two pairs $\mathbf{p}$ and $\mathbf{p}'$ is defined as $\mathbf{p}+\mathbf{p}' = (p_1+p'_1, p_2+p'_2)$. We say that $\mathbf{p}$ *(weakly) dominates* $\mathbf{p}'$, denoted as $\mathbf{p} \preceq \mathbf{p}'$, iff $p_1 \leq p'_1$ and $p_2 \leq p'_2$. For an *approximation factor* (or, more precisely, a pair of approximation factors) $\boldsymbol{\varepsilon} = (\varepsilon_1, \varepsilon_2)$, we say that $\mathbf{p}$ *$\boldsymbol{\varepsilon}$-dominates* $\mathbf{p}'$, denoted as $\mathbf{p} \preceq_{\boldsymbol{\varepsilon}} \mathbf{p}'$, iff $p_1 \leq (1+\varepsilon_1)\cdot p'_1$ and $p_2 \leq (1+\varepsilon_2)\cdot p'_2$.

A *(bi-objective) graph* is a tuple $G = \langle S, E, \mathbf{c}\rangle$, where S is a finite set of *states* and $E \subseteq S \times S$ is a finite set of (directed) *edges*. $succ(s) = \{s' \in S : \langle s, s'\rangle \in E\}$ denotes the *successors* of state s. The *cost function* $\mathbf{c} : E \to \mathbb{R}_{>0} \times \mathbb{R}_{>0}$ maps an edge to its *cost*, which is a pair of positive numbers. A *path* π from state s_1 to state s_ℓ is a sequence of states $[s_1, s_2 \dots s_\ell]$ with $\langle s_j, s_{j+1}\rangle \in E$ for all $j = 1, 2 \dots \ell - 1$. $s_1 = s_{\text{start}}$ unless mentioned otherwise. Slightly abusing the notation, we define the cost of π as $\mathbf{c}(\pi) = \sum_{j=1}^{\ell-1} \mathbf{c}(\langle s_j, s_{j+1}\rangle)$. We say that path π dominates another path π' (resp. π $\boldsymbol{\varepsilon}$-dominates π') iff $\mathbf{c}(\pi) \preceq \mathbf{c}(\pi')$ (resp. $\mathbf{c}(\pi) \preceq_{\boldsymbol{\varepsilon}} \mathbf{c}(\pi')$).

A *WCSP instance* is a tuple $P = \langle G, s_{\text{start}}, s_{\text{goal}}, W\rangle$, where G is a graph, $s_{\text{start}} \in S$ is the *start state*, $s_{\text{goal}} \in S$ is the *goal state*, and $W \in \mathbb{R}_{>0}$ is the *weight limit*. The two components c_1 and c_2 of the cost function $\mathbf{c}$ correspond to the cost and weight in the context of the WCSP problem, respectively. A path π is a *solution* of P iff it is from s_{start} to s_{goal} and satisfies $c_2(\pi) \leq W$. We say that P is *solvable* iff it has a solution. An *optimal solution* of P is a solution with the minimum c_1-value, denoted as c_1^*, of all solutions. Given a non-negative ε, a solution π is *$(1+\varepsilon)$-suboptimal* iff $c_1(\pi) \leq (1+\varepsilon)\cdot c_1^*$. A *bounded-suboptimal WCSP algorithm* takes a WCSP instance P and a parameter $\varepsilon \geq 0$ as input and computes a $(1+\varepsilon)$-suboptimal solution.

A path π from s_{start} to s_{goal} is *Pareto-optimal* iff there does not exist another path π' from s_{start} to s_{goal} with $\mathbf{c}(\pi') \preceq \mathbf{c}(\pi)$ and $\mathbf{c}(\pi') \neq \mathbf{c}(\pi)$. The *Pareto front* Π^* from s_{start} to s_{goal} is the set of all Pareto-optimal paths from s_{start} to s_{goal}. For a non-negative pair $\boldsymbol{\varepsilon}$, a set of paths $\Pi_{\boldsymbol{\varepsilon}}$ from s_{start} to s_{goal} is an *$\boldsymbol{\varepsilon}$-approximate Pareto front* from s_{start} to s_{goal} iff any path from s_{start} to s_{goal} is $\boldsymbol{\varepsilon}$-dominated by at least one path in $\Pi_{\boldsymbol{\varepsilon}}$. Note that different $\boldsymbol{\varepsilon}$-approximate Pareto fronts can exist for the same s_{start}, s_{goal}, and $\boldsymbol{\varepsilon}$.

A BOSP instance is a tuple $\langle G, s_{\text{start}}, s_{\text{goal}}\rangle$, where G is a graph, $s_{\text{start}} \in S$ is the start state, and $s_{\text{goal}} \in S$ is the goal state. An *approximate BOSP algorithm* takes a BOSP instance and an approximation factor $\boldsymbol{\varepsilon} \succeq (0,0)$ as input and computes an $\boldsymbol{\varepsilon}$-approximate Pareto front from s_{start} to s_{goal}.

The following observation shows the connection between a bounded-suboptimal WCSP algorithm and an approximate BOSP algorithm.

Observation 1. *For a solvable WCSP instance $P = \langle G, s_{start}, s_{goal}, W\rangle$ and $\varepsilon \geq 0$, any $(\varepsilon, 0)$-approximate Pareto front $\Pi_{\boldsymbol{\varepsilon}}$ from s_{start} to s_{goal} contains a $(1+\varepsilon)$-suboptimal solution of P.*

Proof. Let π^* denote an optimal solution of P. By definition, there exists a path $\pi \in \Pi_{\boldsymbol{\varepsilon}}$ with $\mathbf{c}(\pi) \preceq_{(\varepsilon,0)} \mathbf{c}(\pi^*)$ (i.e., $c_1(\pi) \leq (1+\varepsilon)\cdot c_1(\pi^*)$ and $c_2(\pi) \leq c_2(\pi^*) \leq W$). Thus, π is a $(1+\varepsilon)$-suboptimal solution of P. □

See Figure 1 for a visualization of an ε-approximate Pareto front and a $(1+\varepsilon)$-suboptimal solution of a WCSP instance.

In this paper, we focus on heuristic-search-based WCSP algorithms. We assume that a *heuristic function* $\mathbf{h} : S \to \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}$, which provides a lower bound on the cost from any given state s to s_{goal}, is always available. Additionally, we assume that the heuristic function is *consistent*, that is, $\mathbf{h}(s_{\text{goal}}) = \mathbf{0}$ and $\mathbf{h}(s) \preceq \mathbf{c}(e) + \mathbf{h}(s')$ for

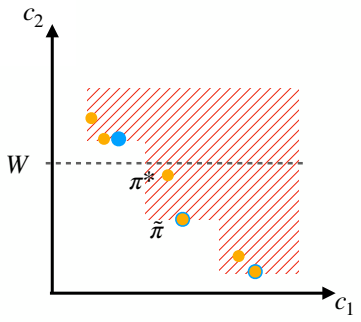


Figure 1: Example of the Pareto front (the costs of whose paths are shown by the orange dots) and an $(\varepsilon, 0)$-approximate Pareto front (the costs of whose paths are shown by the blue dots, two of which overlap with orange dots) for a WCSP instance. The shaded region shows the costs that are $(\varepsilon, 0)$-dominated by at least one blue dot. Note that all orange dots are within the shaded region. Solutions π^* and $\tilde{\pi}$ are an optimal solution and a $(1+\varepsilon)$-suboptimal solution of the WCSP instance, respectively, with $\mathbf{c}(\tilde{\pi}) \preceq_{(\varepsilon,0)} \mathbf{c}(\pi^*)$.

all $e = \langle s, s' \rangle \in E$. It is a common practice in the existing WCSP and BOSP literature (Ahmadi et al. 2021, 2022b; Hernández et al. 2023; Zhang et al. 2022a; Salzman et al. 2023) to use Dijkstra's algorithm (starting from s_{goal}) to compute the minimum cost $c_i^*(s)$ from any state s to s_{goal} for the i-th objective, $i = 1, 2$, (while ignoring the other objective) and $\mathbf{h}(s) := (c_1^*(s), c_2^*(s))$ as the heuristic function. We call this heuristic function the *perfect-distance heuristic*.

Algorithmic Background

In this section, we review existing WCSP and BOSP algorithms, with a focus on BOA* (Hernández et al. 2023), WC-A* (Ahmadi et al. 2022b), and A*pex (Zhang et al. 2022a). All three algorithms fit the same *best-first bi-objective search* framework: A (search) node n contains a state $s(n)$, a $\mathbf{g}$-value $\mathbf{g}(n)$, and an $\mathbf{f}$-value defined as $\mathbf{f}(n) = \mathbf{g}(n) + \mathbf{h}(s(n))$. Node n corresponds to a path from s_{start} to $s(n)$. The search algorithm maintains a priority queue $Open$, which contains the generated but not yet expanded nodes, and a set of solutions. $Open$ is initialized with a node that contains the start state s_{start} and the $\mathbf{g}$-value $\mathbf{0}$.

In each iteration, the search algorithm *extracts* a node n from $Open$ with the *lexicographically smallest* $\mathbf{f}$-value (i.e., extracts a node with the smallest f_1-value and breaks ties in favor of a smaller f_2-value). It then performs a *dominance check* to determine whether n or any of its descendants have the potential to be added to the set of solutions. If not, the search algorithm discards n. Otherwise, it *expands* n: If $s(n) = s_{\text{goal}}$, then the search algorithm adds the path corresponding to n, which is a solution, to the set of solutions. If $s(n) \neq s_{\text{goal}}$, then the search algorithm *generates* a child node for each of the states in $succ(s(n))$. The search algorithm then performs a dominance check for the generated node and adds it to $Open$ if it passes the check. When $Open$ becomes empty, the search algorithm terminates and returns the solution set.

Best-first bi-objective search algorithms differ mainly in which information is contained in the nodes and how the dominance checks work. The dominance checks of both BOA* and A*pex check if the $\mathbf{f}$-value of a node is weakly dominated by the $\mathbf{f}$-value of any expanded node with the same state or s_{goal}.

BOA* and WC-A*

BOA* (Hernández et al. 2023) computes a Pareto front for the given start and goal states. In BOA*, each node n corresponds to a path from s_{start} to $s(n)$ whose cost is $\mathbf{g}(n)$. Hernández et al. (2023) show that, due to the consistent heuristic function that BOA* uses, the f_1-values of the extracted nodes are monotonically non-decreasing. Thus, the dominance checks do not need to consider the f_1-values. Consequently, BOA* stores only the minimum g_2-value $g_2^{\min}(s)$ of all expanded nodes containing the same state s. The dominance check for a node n containing state s can then be done by checking if $g_2(n) < g_2^{\min}(s)$ and $f_2(n) < g_2^{\min}(s_{\text{goal}})$. This dominance check can be performed in constant time (in contrast to earlier methods which required time linear in the number of nodes that contain s).

WC-A* is an optimal WCSP algorithm built on BOA*. It only maintains at most one *incumbent solution*. In addition to discarding nodes via dominance checks, WC-A* also discards nodes (1) whose f_2-values are larger than the weight limit W or (2) whose f_1-values are not smaller than the c_1-value of the incumbent solution. Since WC-A* extracts nodes with monotonically non-decreasing f_1-values, it terminates (and returns the incumbent solution) once the minimum f_1-value in $Open$ is not smaller than the one of the incumbent solution. During the computation of the heuristics with Dijkstra's algorithm, the minimum-c_1 and minimum-c_2 paths from any state s to s_{goal} can also be obtained. We call these paths the *complementary paths* of s. When generating a node n, WC-A* tries to update the incumbent solution with better solutions that extend the corresponding path of n with the complementary paths of $s(n)$.

WC-A* can be converted to a bounded-suboptimal WCSP algorithm by terminating the algorithm when the minimum f_1-value in $Open$ is no longer smaller than the f_1-value of the incumbent solution divided by $(1 + \varepsilon)$. We include this variant of WC-A*, called WC-A*-ε, in our empirical study.

Ahmadi et al. (2022b) propose WC-BA*, a bi-directional variant of WC-A* that runs two WC-A* searches (one starting from s_{start} and the other one starting from s_{goal}) concurrently. We omit WC-BA* from our empirical study because Ahmadi et al. (2022a) later report that WC-BA* does not dominate WC-A* in terms of runtime and, in fact, has larger average runtime in several scenarios.

A*pex

A*pex computes an ε-approximate Pareto front for a given BOSP instance and a given ε-value. In A*pex, a node is a so-called apex-path pair $\mathcal{AP} = \langle \mathbf{A}, \pi \rangle$ that consists of a cost pair $\mathbf{A}$, called the *apex*, and a path π, called the *representative path*. We define the $\mathbf{g}$-value of $\mathcal{AP}$ as $\mathbf{g}(\mathcal{AP}) := \mathbf{A}$ and the state of $\mathcal{AP}$ as the last state of the representative path π.

The $\mathbf{f}$-value of $\mathcal{AP}$ is $\mathbf{f}(\mathcal{AP}) := \mathbf{g}(\mathcal{AP}) + \mathbf{h}(s(\mathcal{AP}))$. Conceptually, an apex-path pair corresponds to a set of paths that end at the same state, and its apex is the component-wise minimum of the costs of these paths. $\mathcal{AP}$ is said to be $\boldsymbol{\varepsilon}$-bounded iff $\mathbf{c}(\pi) + \mathbf{h}(s(\mathcal{AP})) \preceq_{\boldsymbol{\varepsilon}} f(\mathcal{AP})$.

Whenever A*pex inserts an apex-path pair to *Open*, A*pex attempts to *merge* it with another apex-path pair in *Open* that contains the same state on condition that the resulting apex-path pair is $\boldsymbol{\varepsilon}$-bounded. The implementation of A*pex by Zhang et al. (2022a) uses, for each state, a list to maintain the apex-path pairs in *Open* for that state and hence can iterate over these apex-path pairs efficiently. The new apex after merging two apex-path pairs is the component-wise minimum of the apexes of the two apex-path pairs, and the new representative path is either one of the two representative paths of the two apex-path pairs. See Figure 2(a) for a visualization of the two possible outcomes.

WC-A*pex

In this section, we describe WC-A*pex, our bounded suboptimal WCSP algorithm that finds a $(1+\varepsilon)$-suboptimal solution for a given ε. We first describe WC-A*pex and then provide theoretical results and speed-up techniques.

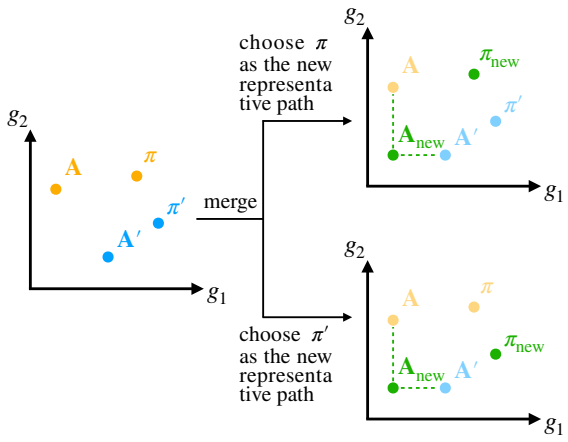
Observation 1 shows that a $(1+\varepsilon)$-suboptimal solution of a WCSP instance can be found in a corresponding $(\varepsilon, 0)$-approximate Pareto front. This motivates us to propose WC-A*pex, which can be viewed as A*pex with $\boldsymbol{\varepsilon} = (\varepsilon, 0)$ and additional node pruning. We use bold $\boldsymbol{\varepsilon}$ and regular ε to distinguish between the approximation factor of A*pex and the given suboptimality factor for the WCSP problem. Similar to A*pex, a node of WC-A*pex is an apex-path pair $\mathcal{AP} = \langle \mathbf{A}, \pi \rangle$. Since the second component of $\boldsymbol{\varepsilon}$ is 0, when merging two apex-path pairs $\langle \mathbf{A}, \pi \rangle$ and $\langle \mathbf{A}', \pi' \rangle$ with $c_2(\pi) < c_2(\pi')$, WC-A*pex cannot choose π' as the new representative path, in which case the resulting apex path pair is not $\boldsymbol{\varepsilon}$-bounded. Therefore, WC-A*pex chooses the path with a smaller c_2-value and breaks ties in favor of a smaller c_1-value. See Figure 2(b) for a visualization of merging two apex-path pairs for WC-A*pex.

Algorithm 1 shows the pseudocode of WC-A*pex. It starts with an apex-path pair $\langle \mathbf{0}, [s_{\text{start}}] \rangle$ in *Open* (Line 1). At each iteration, WC-A*pex extracts an apex-path pair from *Open* with the lexicographically smallest $\mathbf{f}$-value (Line 5). Similar to BOA*, WC-A*pex maintains a $g_2^{\min}$-value for each state s that contains the smallest g_2-value of all expanded nodes with state s and updates it on Line 10. Both after extracting (that is, after Line 5) and before generating (that is, before Line 16) an apex-path pair $\mathcal{AP}$ with state s, WC-A*pex discards the apex-path pair if (1) $g_2(\mathcal{AP}) \geq g_2^{\min}(s(\mathcal{AP}))$ or (2) $f_2(\mathcal{AP}) > W$. Case (1) holds iff there exists an expanded node containing state s whose $\mathbf{g}$-value weakly dominates $\mathbf{g}(\mathcal{AP})$, which implies that any solution found via $\mathcal{AP}$ is also $(\varepsilon, 0)$-dominated by a solution found via the expanded node and $\mathcal{AP}$ thus can be safely pruned. Case (2) holds only if the representative path of $\mathcal{AP}$ cannot be extended to a solution (since the c_2-value of any solution cannot be larger than W) and $\mathcal{AP}$ thus can be safely pruned.

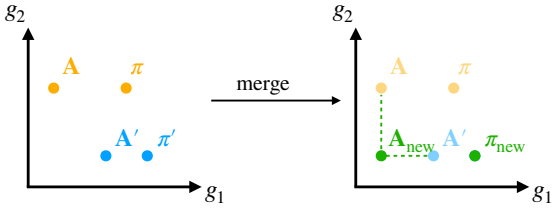
When WC-A*pex expands an apex-path pair $\mathcal{AP}$ with state s, it generates a child apex-path pair for each successor s' of state s. The apex of the child apex-path pair is the sum of the apex of $\mathcal{AP}$ and $\mathbf{c}(\langle s, s' \rangle)$ (Line 12), and the representative path of the child apex-path pair is the representative path of $\mathcal{AP}$ appended with state s' (Line 13). Before adding the child apex-path pair $\mathcal{AP}'$ to *Open*, WC-A*pex attempts to merge $\mathcal{AP}'$ with an apex-path pair in $Open[s(\mathcal{AP}')]$ on condition that the resulting apex-path pair is $(\varepsilon, 0)$-bounded (Lines 19-25), where ε is the input suboptimality factor and $Open[s(\mathcal{AP}')]$ is the set of apex-path pairs in *Open* for state $s(\mathcal{AP}')$.

WC-A*pex terminates when it finds a solution (Line 9) or when *Open* becomes empty (Line 17). In the latter case, the given WCSP instance has no solution.

We use an example WCSP instance to demonstrate how WC-A*pex works. Figure 3(a) shows its graph. The weight limit W and ε are 7 and 0.2, respectively. Figure 3(b) shows the costs of all paths from s_{start} to s_{goal}. Path $[s_{\text{start}}, s_1, s_3, s_4, s_{\text{goal}}]$, whose cost is $(7, 7)$, is the optimal solution of this WCSP instance. Moreover, since the second-best solution $[s_{\text{start}}, s_1, s_2, s_3, s_5, s_{\text{goal}}]$ has a large c_1-value of 13, $[s_{\text{start}}, s_1, s_3, s_4, s_{\text{goal}}]$ is also the only 1.2-suboptimal solution of this WCSP instance. We use the perfect-distance heuristic. Slightly abusing the notation, we use tuple $\langle s(\mathcal{AP}_i), \mathbf{f}(\mathcal{AP}_i), \mathbf{c}(\pi_i) \rangle$ to denote an apex-path pair $\mathcal{AP}_i = \langle \mathbf{A}_i, \pi_i \rangle$.



(a) A*pex (adapted from Figure 2 by Zhang et al. (2022a))



(b) WC-A*pex

Figure 2: Examples of merging apex-path pairs $\langle \mathbf{A}, \pi \rangle$ (orange) and $\langle \mathbf{A}', \pi' \rangle$ (blue) into apex-path pair $\langle \mathbf{A}_{\text{new}}, \pi_{\text{new}} \rangle$ (green) for A*pex and WC-A*pex, respectively.

Algorithm 1: WC-A*pex

Input : $P = \langle G, s_{\text{start}}, s_{\text{goal}}, W \rangle$
ε
$\mathbf{h}$

$Open \leftarrow \{\langle \mathbf{0}, [s_{\text{start}}] \rangle\}$
for each $s \in S$ **do**
  $g_2^{\min}(s) \leftarrow \infty$
while $Open \neq \emptyset$ **do**
  $\mathcal{AP} = \langle \mathbf{A}, \pi \rangle \leftarrow Open$.extract_min()
  if $g_2(\mathcal{AP}) \geq g_2^{min}(s(\mathcal{AP})) \vee f_2(\mathcal{AP}) > W$ **then**
    continue
  if $s(\mathcal{AP}) = s_{goal}$ **then**
    return π
  $g_2^{\min}(s(\mathcal{AP})) \leftarrow g_2(\mathcal{AP})$
  for $s' \in succ(s(\mathcal{AP}))$ **do**
    $\mathbf{A}' \leftarrow \mathbf{A} + \mathbf{c}(\langle s(\mathcal{AP}), s' \rangle)$
    $\pi' \leftarrow \pi$.append(s')
    if $A'_2 \geq g_2^{min}(s') \vee A'_2 + h_2(s') > W$ **then**
      continue
    insert_to_Open$(\langle \mathbf{A}', \pi' \rangle)$
return *None*

Function *insert_to_Open*$(\mathcal{AP}' = \langle \mathbf{A}', \pi' \rangle)$**:**
  for $\mathcal{AP} = \langle \mathbf{A}, \pi \rangle \in Open[s(\mathcal{AP}')]$ **do**
    $\mathbf{A}_{\text{new}} \leftarrow (\min(A_1, A'_1), \min(A_2, A'_2))$
    $\pi_{\text{new}} \leftarrow$ the one of π and π' with the smaller c_2-value, breaking ties in favor of a smaller c_1-value
    if $\langle \mathbf{A}_{new}, \pi_{new} \rangle$ *is* $(\varepsilon, 0)$*-bounded* **then**
      remove $\mathcal{AP}$ from $Open$
      add $\langle \mathbf{A}_{\text{new}}, \pi_{\text{new}} \rangle$ to $Open$
      return
  add $\mathcal{AP}'$ to $Open$
  return

- In Iteration 1, WC-A*pex expands apex-path pair $\mathcal{AP}_1 = \langle s_{\text{start}}, (5,5), (0,0) \rangle$ and generates two child apex-path pairs $\mathcal{AP}_2 = \langle s_1, (5,5), (1,2) \rangle$ and $\mathcal{AP}_3 = \langle s_2, (6,6), (3,2) \rangle$.
- In Iteration 2, WC-A*pex expands apex-path pair $\mathcal{AP}_2$ and generates two child apex-path pairs $\mathcal{AP}_4 = \langle s_2, (5,7), (2,3) \rangle$ and $\mathcal{AP}_5 = \langle s_3, (7,5), (5,3) \rangle$. $\mathcal{AP}_4$ is merged with $\mathcal{AP}_3$ in $Open$, resulting in apex-path pair $\mathcal{AP}_6 = \langle s_2, (5,6), (3,2) \rangle$. $\mathcal{AP}_6$ is $(\varepsilon, 0)$-bounded because $(3,2) + \mathbf{h}(s_2) = (6,6) \preceq_{(\varepsilon,0)} (5,6)$.
- In Iteration 3, WC-A*pex expands apex-path pair $\mathcal{AP}_6$ and generates child apex-path pair $\mathcal{AP}_7 = \langle s_3, (5,6), (4,4) \rangle$. $\mathcal{AP}_7$ is not merged with $\mathcal{AP}_5$ because, given that the new representative path would have a cost of $(5,3)$ and the new $\mathbf{f}$-value would be $(5,5)$, $(5,3) + \mathbf{h}(s_3) = (7,5)$ does not $(\varepsilon, 0)$-dominate $(5,5)$.
- In Iteration 4, WC-A*pex expands apex-path pair $\mathcal{AP}_7$ and generates two child apex-path pairs $\mathcal{AP}_8 = \langle s_4, (5,8), (5,6) \rangle$ and $\mathcal{AP}_9 = \langle s_5, (13,6), (9,5) \rangle$. $\mathcal{AP}_8$ is pruned because $f_2(\mathcal{AP}_8) > W$.
- In Iteration 5, WC-A*pex expands apex-path pair $\mathcal{AP}_5$ and generates two child apex-path pairs $\mathcal{AP}_{10} = \langle s_4, (7,7), (6,5) \rangle$ and $\mathcal{AP}_{11} = \langle s_5, (15,5), (10,4) \rangle$. $\mathcal{AP}_{11}$ is merged with $\mathcal{AP}_9$ in $Open$, resulting in apex-path pair $\mathcal{AP}_{12} = \langle s_5, (13,5), (10,4) \rangle$. $\mathcal{AP}_{12}$ is $(\varepsilon, 0)$-bounded because $(10,4) + \mathbf{h}(s_5) = (15,5) \preceq_{(\varepsilon,0)} (13,5)$.
- In Iteration 6, WC-A*pex expands apex-path pair $\mathcal{AP}_{10}$ and generates child apex-path pair $\mathcal{AP}_{13} = \langle s_{\text{goal}}, (7,7), (7,7) \rangle$.
- In Iteration 7, WC-A*pex expands apex-path pair $\mathcal{AP}_{13}$ and returns a solution with cost $(7,7)$.

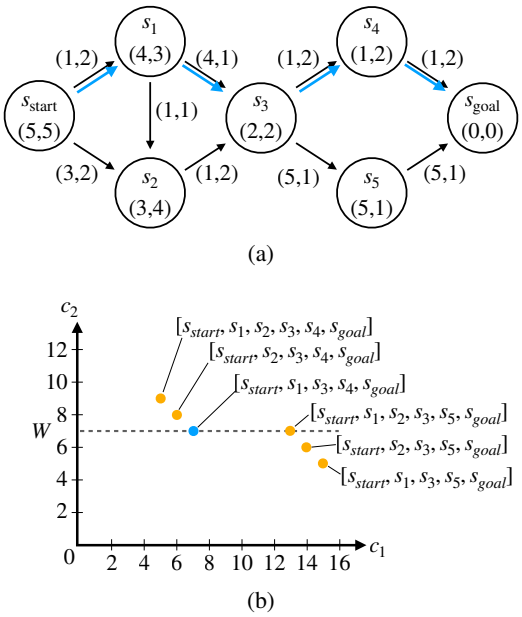


Figure 3: An example WCSP instance. (a) shows the graph of this WCSP instance, where the pair of numbers inside each state is its $\mathbf{h}$-value and the blue arrows show the optimal solution for weight limit $W = 7$. (b) shows the costs of all paths from s_{start} to s_{goal} in the graph.

In this example, WC-A*pex finds the optimal solution. Two merges happen during the entire process, namely in Iteration 2 between $\mathcal{AP}_3 = \langle s_2, (6,6), (3,2) \rangle$ and $\mathcal{AP}_4 = \langle s_2, (5,7), (2,3) \rangle$ and in Iteration 5 between $\mathcal{AP}_9 = \langle s_5, (13,6), (9,5) \rangle$ and $\mathcal{AP}_{11} = \langle s_5, (15,5), (10,4) \rangle$. The representative paths of $\mathcal{AP}_3$ and $\mathcal{AP}_4$ are $\pi_3 = [s_{\text{start}}, s_2]$ and $\pi_4 = [s_{\text{start}}, s_1, s_2]$, respectively. Compared to $\mathcal{AP}_5$, which is expanded in Iteration 5 and eventually extended to the returned solution, $\mathcal{AP}_3$ and $\mathcal{AP}_4$ have lexicographically smaller $\mathbf{f}$-values and appear to be more promising. However, the two possible extensions of $\mathcal{AP}_3$ and $\mathcal{AP}_4$ to s_{goal} either violate the weight limit (via $[s_2, s_3, s_4, s_{\text{goal}}]$) or have large c_1-values (via $[s_2, s_3, s_5, s_{\text{goal}}]$), as WC-A*pex finds out in Iterations 3 and 4. Without merging, previous WCSP algorithms, like WC-A*, would represent π_3 and π_4 as two different nodes and spend more search effort. This example demonstrates how merging can speed up the search.

Theoretical Results

In this section, we show that WC-A*pex returns a $(1+\varepsilon)$-suboptimal solution for any solvable WCSP instance.

Lemma 1. *If $g_2(\mathcal{AP}) \geq g_2^{min}(s(\mathcal{AP}))$ on Line 6 or 14, then there exists an expanded apex-path pair $\mathcal{AP}'$ with state $s(\mathcal{AP}') = s(\mathcal{AP})$ and $\mathbf{f}(\mathcal{AP}') \preceq \mathbf{f}(\mathcal{AP})$.*

This lemma is rephrased from Lemma 2 by Zhang et al. (2022a), and the same proof applies.

For the remainder of this section, we use $\pi^* = [s_1^* (= s_{\text{start}}), s_2^* \dots s_\ell^* (= s_{\text{goal}})]$ to denote an optimal solution for the given solvable WCSP instance. We use $\pi_j^* = [s_1^*, s_2^* \dots s_j^*], j = 1, 2 \dots \ell$, to denote a prefix of π^*.

Lemma 2. *At the beginning of any iteration (i.e., before executing Line 5), if WC-A*pex has expanded an apex-path pair $\mathcal{AP}$ with $s(\mathcal{AP}) = s_j^*$ and $\mathbf{g}(\mathcal{AP}) \preceq \mathbf{c}(\pi_j^*)$ for some $j \in \{1, 2 \dots \ell - 1\}$, then there exists an apex-path pair $\mathcal{AP}'$ in Open with $s(\mathcal{AP}') = s_k^*$ and $\mathbf{g}(\mathcal{AP}') \preceq \mathbf{c}(\pi_k^*)$ for some $k > j$.*

Proof. We prove this lemma by induction on j, starting with $j = \ell - 1$ and going backward. Consider an expanded apex-pair $\mathcal{AP}$ with $s(\mathcal{AP}) = s_{\ell-1}^*$ and $\mathbf{g}(\mathcal{AP}) \preceq \mathbf{c}(\pi_{\ell-1}^*)$. When expanding $\mathcal{AP}$, WC-A*pex generates a child apex-path pair $\mathcal{AP}'$ that contains state s_ℓ^* $(= s_{\text{goal}})$. We have $g_2^{\min}(s_{\text{goal}}) = \infty$ because, if WC-A*pex had extracted any apex-path pair from $Open$ that contained s_{goal}, it would have terminated on Line 9 and could not have reached Line 10 to update $g_2^{\min}(s_{\text{goal}})$. We have $\mathbf{g}(\mathcal{AP}') = \mathbf{g}(\mathcal{AP}) + \mathbf{c}(\langle s_{\ell-1}^*, s_\ell^* \rangle) \preceq \mathbf{c}(\pi_{\ell-1}^*) + \mathbf{c}(\langle s_{\ell-1}^*, s_\ell^* \rangle) = \mathbf{c}(\pi^*)$. Since the heuristic $\mathbf{h}$ is consistent, we have $\mathbf{h}(s_\ell^*) = \mathbf{0}$ and hence $f_2(\mathcal{AP}') = g_2(\mathcal{AP}') \leq c_2(\pi^*) \leq W$. Therefore, $\mathcal{AP}'$ is not pruned on Line 15 but inserted to $Open$. If $\mathcal{AP}'$ had been extracted from $Open$, WC-A*pex would have terminated. Therefore, $\mathcal{AP}'$ remains in $Open$.

Now we assume that the lemma holds for $j = i + 1$, $i \in \{1, 2 \dots \ell - 2\}$. Consider an expanded apex-path pair $\mathcal{AP}$ with $s(\mathcal{AP}) = s_i^*$ and $\mathbf{g}(\mathcal{AP}) \preceq \mathbf{c}(\pi_i^*)$. When expanding $\mathcal{AP}$, one of its child apex-path pairs, denoted as $\mathcal{AP}'$, contains state s_{i+1}^*. We have $\mathbf{g}(\mathcal{AP}') = \mathbf{g}(\mathcal{AP}) + \mathbf{c}(\langle s_i^*, s_{i+1}^* \rangle) \preceq \mathbf{c}(\pi_i^*) + \mathbf{c}(\langle s_i^*, s_{i+1}^* \rangle) = \mathbf{c}(\pi_{i+1}^*)$. We distinguish two cases:

1. $\mathcal{AP}'$ is pruned on Line 15. Since $f_2(\mathcal{AP}') = g_2(\mathcal{AP}') + h_2(s_{i+1}^*) \leq c_2(\pi_{i+1}^*) + h_2(s_{i+1}^*) \leq c_2(\pi^*) \leq W$, the reason for pruning $\mathcal{AP}'$ can only be that $g_2^{\min}(s_{i+1}^*) \leq g_2(\mathcal{AP}')$. Then, from Lemma 1, there exists an expanded apex-path pair with state s_{i+1}^* and whose $\mathbf{g}$-value weakly dominates $\mathbf{g}(\mathcal{AP}')$ and hence $\mathbf{c}(\pi_{i+1}^*)$. Because the lemma holds for $j = i + 1$, there exists an apex-path pair $\mathcal{AP}''$ in $Open$ with $s(\mathcal{AP}'') = s_k^*$ and $\mathbf{g}(\mathcal{AP}'') \preceq \mathbf{c}(\pi_k^*)$ for some $k > i + 1$. Thus, the lemma holds for $j = i$.
2. $\mathcal{AP}'$ is not pruned on Line 15 but is inserted to $Open$, perhaps after merging it with another apex-path pair. Thus, an apex-path pair with state s_{i+1}^* whose $\mathbf{g}$-value weakly dominates $\mathbf{c}(\pi_{i+1}^*)$ is inserted to $Open$. The lemma holds for $j = i$ as long as this new apex-path pair remains in $Open$. If this new apex-path pair is extracted and then either pruned or expanded, then there is an expanded apex-path pair with state s_{i+1}^* and whose $\mathbf{g}$-value weakly dominates $\mathbf{c}(\pi_{i+1}^*)$. Because the lemma holds for $j = i + 1$, it thus also holds for $j = i$.

Therefore, the lemma holds for all $j \in \{1, 2 \dots \ell - 1\}$. □

Theorem 1. *WC-A*pex returns a $(1+\varepsilon)$-suboptimal solution for any solvable WCSP problem instance.*

Proof. In the first iteration, there is one apex-path pair $\mathcal{AP}_{\text{init}} = \langle \mathbf{0}, [s_{\text{start}}] \rangle$ in $Open$. We have $s(\mathcal{AP}_{\text{init}}) = s_1^*$ and $\mathbf{g}(\mathcal{AP}_{\text{init}}) \preceq \mathbf{c}(\pi_1^*)$. $\mathcal{AP}_{\text{init}}$ is then expanded in the first iteration, and, from Lemma 2, $Open$ always contains at least one apex-path pair at the beginning of all future iterations. Therefore, WC-A*pex will not reach Line 17 and return $None$. Since there are only a finite number of paths whose c_2-values are not larger than W, there are only a finite number of expanded apex-path pairs, and hence a finite number of iterations before WC-A*pex terminates. Let π_{sol} and $\mathcal{AP}_{\text{sol}}$ be the path returned by WC-A*pex and the apex-path pair that contains π_{sol}, respectively. π_{sol} is a solution because it is from s_{start} to s_{goal} (due to the condition on Line 8) and its c_2-value is not larger than W (otherwise, $\mathcal{AP}_{\text{sol}}$ would have been pruned on Line 7). Because $\mathcal{AP}_{\text{sol}}$ is $(\varepsilon, 0)$-bounded (which is due to the conditions on Line 22 and since the heuristic function is consistent), we have $(1+\varepsilon) \cdot f_1(\mathcal{AP}_{\text{sol}}) \geq c_1(\pi_{\text{sol}})$. We prove that π_{sol} must be $(1+\varepsilon)$-suboptimal by contradiction: Assume that π_{sol} is not $(1+\varepsilon)$-suboptimal, i.e., $c_1(\pi_{\text{sol}}) > (1+\varepsilon) \cdot c_1(\pi^*)$. Put together, we have $(1+\varepsilon) \cdot f_1(\mathcal{AP}_{\text{sol}}) \geq c_1(\pi_{\text{sol}}) > (1+\varepsilon) \cdot c_1(\pi^*)$ and hence $f_1(\mathcal{AP}_{\text{sol}}) > c_1(\pi^*)$. From Lemma 2, there always exists an apex-path pair $\mathcal{AP}' \in Open$ with $s(\mathcal{AP}') = s_k^*$ and $\mathbf{g}(\mathcal{AP}') \preceq \mathbf{c}(\pi_k^*)$ for some k. We have $\mathbf{f}(\mathcal{AP}') = \mathbf{g}(\mathcal{AP}') + \mathbf{h}(s(\mathcal{AP}')) \preceq \mathbf{c}(\pi_k^*) + \mathbf{h}(s_k^*) \preceq \mathbf{c}(\pi^*)$ (and hence $f_1(\mathcal{AP}') \leq c_1(\pi^*) < f_1(\mathcal{AP}_{\text{sol}})$). WC-A*pex must extract $\mathcal{AP}'$ before extracting $\mathcal{AP}_{\text{sol}}$, which is a contradiction. □

Speed-up Techniques

In this section, we describe some speed-up techniques for an efficient implementation of WC-A*pex. Some of these techniques are also used by existing algorithms like WC-A*, and hence we omit the theoretical results for them.

Efficient data structures: Similar to WC-A*, we let WC-A*pex use a bucket queue to implement $Open$. Additionally, for each state s, we use a doubly-linked list to keep track of all apex-path pairs in $Open$ with state s. Therefore, WC-A*pex can efficiently iterate over $Open[s]$ for any state s on Lines 19-25 and efficiently update the doubly-linked list when an apex-path pair is extracted from or inserted to $Open$. Our preliminary results confirmed that these data structures speed up the original implementation of A*pex by an order of magnitude.

Early solution updates: Similar to WC-A*, we let WC-A*pex maintain and update an incumbent solution using complementary paths. An apex-path pair $\mathcal{AP}$ is pruned if

Road Network	ε	WC-A* (-ε)				WC-A*pex				Speed-Up
		Runtime		Expansions		Runtime		Expansions		
		Avg	Max	Avg	Max	Avg.	Max	Avg	Max	
FLA	0	0.085	4.482	1,217K	49,400K					
	0.01	0.081	5.121	1,083K	47,994K	0.023	0.707	142K	3,543K	3.48
	0.05	0.050	3.753	731K	41,380K	0.013	0.454	81K	2,290K	3.88
	0.10	0.035	4.482	458K	41,380K	0.008	0.443	49K	2,128K	4.43
	0.20	0.015	2.321	176K	18,578K	0.004	0.250	19K	1,291K	4.31
NE	0	0.184	8.376	1,877K	59,122K					
	0.01	0.153	5.284	1,577K	44,032K	0.047	2.503	251K	8,302K	3.24
	0.05	0.071	3.014	841K	26,694K	0.018	0.620	109K	2,461K	3.94
	0.10	0.024	0.861	361K	9,610K	0.008	0.192	52K	1,160K	2.91
	0.20	0.005	0.401	76K	6,914K	0.002	0.123	13K	810K	1.93
LKS	0	4.609	129.534	32,448K	721,867K					
	0.01	3.813	97.928	27,285K	551,673K	0.655	17.383	2,128K	40,169K	5.82
	0.05	2.112	60.093	15,441K	327,693K	0.219	4.275	836K	11,486K	9.64
	0.10	0.944	38.345	7,590K	200,622K	0.107	3.046	493K	9,306K	8.83
	0.20	0.077	4.093	1,003K	31,143K	0.016	0.502	91K	2,062K	4.92
E	0	5.485	138.313	36,554K	763,176K					
	0.01	4.614	132.302	31,488K	715,655K	0.845	16.960	2,714K	51,904K	5.46
	0.05	2.629	76.859	18,918K	473,144K	0.216	4.427	876K	14,754K	12.16
	0.10	0.972	40.358	8,248K	279,604K	0.110	3.486	493K	12,604K	8.81
	0.20	0.212	21.542	1,777K	138,240K	0.026	2.079	112K	6,464K	8.13
W	0	4.615	227.762	33,610K	1,096,261K					
	0.01	3.945	180.262	30,536K	920,302K	0.567	33.271	2,248K	86,746K	6.96
	0.05	2.323	70.197	19,962K	553,971K	0.260	7.167	1,233K	21,667K	8.93
	0.10	0.967	57.564	10,048K	467,409K	0.122	3.521	666K	16,178K	7.92
	0.20	0.292	41.619	3,220K	304,128K	0.041	2.768	241K	12,997K	7.05
CAL	0	0.382	21.479	3,897K	138,645K					
	0.01	0.333	20.423	3,476K	131,590K	0.111	4.786	508K	16,659K	3.01
	0.05	0.166	6.520	2,030K	54,052K	0.052	1.560	257K	5,771K	3.21
	0.10	0.093	3.991	1,222K	46,152K	0.030	1.209	160K	5,771K	3.09
	0.20	0.033	3.383	502K	38,419K	0.014	1.199	73K	5,325K	2.36
CTR	0	17.561	243.423	94,661K	1,106,707K					
	0.01	14.711	222.586	82,911K	1,013,655K	1.536	29.957	4,592K	67,054K	9.57
	0.05	8.757	185.336	50,288K	804,131K	0.531	11.827	1,942K	32,594K	16.48
	0.10	4.173	99.930	24,928K	468,607K	0.291	6.359	1,119K	18,877K	14.36
	0.20	0.599	39.787	4,719K	233,685K	0.061	3.008	267K	10,624K	9.84

Table 1: Average and maximum runtimes (in seconds), average and maximum numbers of node expansions, and speed-ups of WC-A*pex over WC-A*-ε in average runtimes for WCSP instances on different road networks.

$(1+\varepsilon)\cdot f_1(\mathcal{AP})$ is not smaller than the c_1-value of the incumbent solution. WC-A*pex then terminates when $Open$ becomes empty and returns the incumbent solution.

Experimental Results

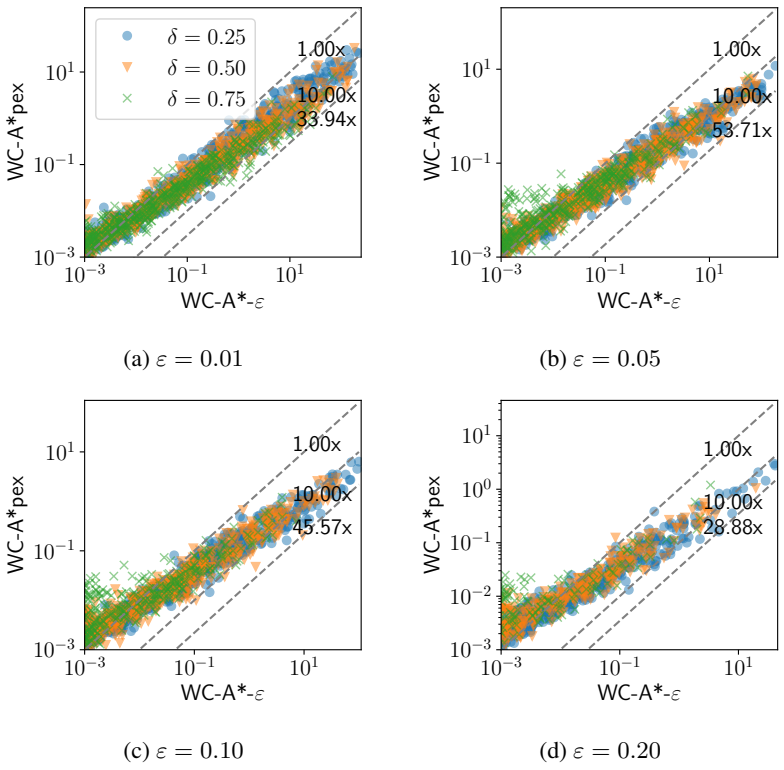
In this section, we evaluate WC-A*pex with WCSP instances on road networks from the 9th DIMACS Implementation Challenge: Shortest Path.[1] We investigate WC-A*pex with different ε-values and compare the runtimes and numbers of node expansions of WC-A*, WC-A*-ε, and WC-A*pex. We implemented WC-A*pex in C++ from scratch[2] and implemented WC-A*-ε based on the C++ implementation of WC-A* provided by the original authors.[3]

We choose seven road networks, namely FLA (1.1M states and 2.7M edges), NE (1.5M states and 3.9M edges), CAL (1.9M states and 4.7M edges), LKS (2.8M states and 6.9M edges), E (3.6M states and 8.8M edges), W (6.3M states and 15.2M edges), and CTR (14.1M states and 34.3M edges) from the DIMACS data set. The c_1- and c_2-values for each edge are its travel time and distance, respectively, both available from the DIMACS data set. Each WCSP instance thus corresponds to computing a path that is bounded-suboptimal with respect to its travel time and with its travel distance being no larger than a given limit. For each road network, we use the same 100 s_{start} and s_{goal} pairs used by Sedeño-Noda and Colebrook (2019) and Ahmadi et al. (2021). Following previous work (Cabrera et al. 2020; Ahmadi et al. 2022b), for each s_{start} and s_{goal} pair, we generate a WCSP instance with the weight limit $W = c_2^{\text{lb}} + \delta\cdot(c_2^{\text{ub}} - c_2^{\text{lb}})$ based on a *tightness factor* δ, where c_2^{lb} and c_2^{ub} are the minimum and maximum c_2-values of all Pareto-optimal paths from s_{start} to s_{goal}, respectively. A smaller δ-value thus cor-

[1] http://www.diag.uniroma1.it/challenge9/download.shtml.

[2] https://github.com/HanZhang39/MultiObjectiveSearch

[3] https://bitbucket.org/s-ahmadi/biobj/src/master/



(a) $\varepsilon = 0.01$ (b) $\varepsilon = 0.05$

(c) $\varepsilon = 0.10$ (d) $\varepsilon = 0.20$

Figure 4: Runtimes of WC-A*pex and WC-A*-ε with different suboptimality factors on all WCSP instances.

responds to a tighter weight limit. For each s_{start} and s_{goal} pair, we use the tightness factors $0.25, 0.5$, and 0.75. Therefore, we have 300 WCSP instances for each road network.

For each WCSP instance, we evaluate WC-A*-ε and WC-A*pex with the ε-values 0.01, 0.05, and 0.1. Table 1 shows the average and maximum runtimes (in seconds) and the numbers of node expansions of WC-A*, WC-A*-ε, and WC-A*pex over all WCSP instances. The results for WC-A* are shown in the rows with $\varepsilon = 0$. We choose not to show the results for different tightness factors (δ-values) separately because, as we will show in Figure 4, they do not affect the results significantly. With $\varepsilon = 0.01$, i.e., a guaranteed suboptimality of at most 1%, the average speed-up of WC-A*pex over WC-A* is more than $11\times$ on the largest road network (CTR). However, the average speed-up of WC-A*-ε with $\varepsilon = 0.01$ over WC-A* is only about 20% because WC-A*-ε still needs to expand a large number of nodes to prove that the incumbent solution is bounded-suboptimal. The runtimes and numbers of node expansions of WC-A*pex are always smaller than the ones of WC-A* and WC-A*-ε with the same ε-value on all road networks, which shows that merging apex-path pairs greatly reduces the runtimes and numbers of node expansions.

Figure 4 shows the individual runtimes (in seconds) of WC-A*pex and WC-A*-ε for all WCSP instances and ε-values. We use different markers for different tightness factors δ used to generate the WCSP instances. The diagonal dashed lines and the numbers along them denote different speed-ups ($1\times$, $10\times$, and the maximum speed-up) of WC-A*pex over WC-A*-ε . For different tightness factors, the trends of the speed-ups of WC-A*pex over WC-A*-ε are similar. Although WC-A*pex is slower than WC-A*-ε on easy WCSP instances (which both algorithm solve mostly within around 0.1 seconds), WC-A*pex achieves significant speed-ups over WC-A*-ε on more difficult instances (represented by the points on the top-right corners).

Conclusions

In this paper, we proposed the bounded-suboptimal WCSP algorithm WC-A*pex. WC-A*pex is built on A*pex, a state-of-the-art approximate BOSP algorithm. Its empirical performance on benchmark road networks highlights two important computational aspects of it. First, huge gains in runtime (namely, up to an order of magnitude) are possible with only a small compromise on the cost of the solution (namely, a 1% suboptimality). Second, the merged representation of paths reduces the number of node expansions and is critical to the success of WC-A*pex over WC-A* and WC-A*-ε.

In future work, we intend to generalize WC-A*pex to multiple costs, multiple weights (Skyler et al. 2022), or both. We also intend to enhance WC-A*pex with recent algorithmic advancements (Zhang et al. 2022b) and make it into an anytime algorithm.

Acknowledgements

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189, 1837779, 1935712, and 2112533. The research was also supported by the United States-Israel Binational Science Foundation (BSF) under grant number 2021643 and by grant No. 3-17385 from the Ministry of Science & Technology, Israel. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or governments.

References

Ahmadi, S.; Tack, G.; Harabor, D.; and Kilby, P. 2021. Bi-objective Search with Bi-Directional A*. In *Symposium on Combinatorial Search (SOCS)*, 142–144.

Ahmadi, S.; Tack, G.; Harabor, D.; and Kilby, P. 2022a. Enhanced Methods for the Weight Constrained Shortest Path Problem: Constrained Path Finding Meets Bi-Objective Search. *arXiv preprint arXiv:2207.14744*.

Ahmadi, S.; Tack, G.; Harabor, D.; and Kilby, P. 2022b. Weight Constrained Path Finding with Bidirectional A*. In *Symposium on Combinatorial Search (SOCS)*, 2–10.

Baum, M.; Dibbelt, J.; Gemsa, A.; Wagner, D.; and Zündorf, T. 2015. Shortest Feasible Paths with Charging Stops for Battery Electric Vehicles. In *SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 1–10.

Cabrera, N.; Medaglia, A. L.; Lozano, L.; and Duque, D. 2020. An Exact Bidirectional Pulse Algorithm for the Constrained Shortest Path. *Networks*, 76(2): 128–146.

Ergun, F.; Sinha, R.; and Zhang, L. 2002. An Improved FPTAS for Restricted Shortest Path. *Information Processing Letters*, 83(5): 287–291.

Handler, G. Y.; and Zang, I. 1980. A Dual Algorithm for the Constrained Shortest Path Problem. *Networks*, 10(4): 293–309.

Hernández, C.; Yeoh, W.; Baier, J. A.; Zhang, H.; Suazo, L.; Koenig, S.; and Salzman, O. 2023. Simple and Efficient Bi-Objective Search Algorithms via Fast Dominance Checks. *Artificial Intelligence*, 314: 103807.

Lorenz, D. H.; and Raz, D. 2001. A Simple Efficient Approximation Scheme for the Restricted Shortest Path Problem. *Operations Research Letters*, 28(5): 213–219.

Pearl, J.; and Kim, J. H. 1982. Studies in Semi-Admissible Heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-4(4): 392–399.

Pohl, I. 1970. Heuristic Search Viewed as Path Finding in a Graph. *Artificial Intelligence*, 193–204.

Rost, M. J. 2019. *Virtual Network Embeddings: Theoretical Foundations and Provably Good Algorithms*. Technische Universitaet Berlin (Germany).

Salzman, O.; Felner, A.; Hernández, C.; Zhang, H.; Chan, S.; and Koenig, S. 2023. Heuristic-Search Approaches for the Multi-Objective Shortest-Path Problem: Progress and Research Opportunities. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 6759–6768.

Sedeño-Noda, A.; and Colebrook, M. 2019. A Biobjective Dijkstra Algorithm. *European Journal of Operational Research*, 276(1): 106–118.

Skyler, S.; Atzmon, D.; Felner, A.; Salzman, O.; Zhang, H.; Koenig, S.; Yeoh, W.; and Hernández, C. 2022. Bounded-Cost Bi-Objective Heuristic Search. In *Symposium on Combinatorial Search (SOCS)*, 239–243.

Storandt, S. 2012. Route Planning for Bicycles—Exact Constrained Shortest Paths Made Practical via Contraction Hierarchy. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 234–242.

Thayer, J. T.; and Ruml, W. 2011. Bounded Suboptimal Search: A Direct Approach Using Inadmissible Estimates. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 674–679.

Thomas, B. W.; Calogiuri, T.; and Hewitt, M. 2019. An Exact Bidirectional A* Approach for Solving Resource-constrained Shortest Path Problems. *Networks*, 73(2): 187–205.

Zhang, H.; Salzman, O.; Kumar, T. K. S.; Felner, A.; Hernández, C.; and Koenig, S. 2022a. A* pex: Efficient Approximate Multi-Objective Search on Graphs. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 394–403.

Zhang, H.; Salzman, O.; Kumar, T. K. S.; Felner, A.; Hernández, C.; and Koenig, S. 2022b. Anytime Approximate Bi-Objective Search. In *Symposium on Combinatorial Search (SOCS)*, 199–207.