Abstract—Shortest path problem has been successfully applied in numerous domains. Unfortunately, its complexity increases drastically when several objective criteria must be considered. Apart from the relatively slow classic search algorithms, attempts to accelerate multicriteria shortest path search are mostly represented by goal-directed one-to-one search methods and pruning heuristics. The one-to-many version of the problem is rarely addressed, though it arises in various scenarios, such as multi-stop planning and dynamic rerouting. This paper introduces a novel algorithm combination designed for fast one-to-many multicriteria shortest path search. A preprocessing algorithm excludes irrelevant vertices by building a smaller cover graph. A modified version of the multicriteria label-setting algorithm operates on the cover graph and employs a dimensionality reduction technique for swifter domination checks. While the method itself maintains solution optimality, it is able to additionally incorporate existing heuristics for further speedups. Additionally, its operation is not limited to bicriteria cases and requires no modifications to incorporate a higher number of criteria. The proposed algorithm was tested on multiple criteria combinations of varying correlation and compared to existing algorithms in terms of scalability. Apart from the speedup provided, graph preprocessing also reduces memory requirements of queries by up to 13 times.

Index Terms—Multicriteria path search, pareto optimization, shortest path problem.

I. INTRODUCTION

SHORTEST path problem is one of the most well-known and studied tasks in graph theory. It poses a challenge of finding a feasible path between a pair of vertices that has a minimum possible sum of its component edge weights. However, the traditional form of the problem is only concerned with minimization of a single criterion parameter, which limits its applicability to real-world tasks. A prominent example of insufficient characterization provided by a single criterion is route planning in transportation. Edges in road networks can be assigned multiple parameters, such as traversal time, distance, fuel requirement, and each of the parameters can potentially influence the optimality of a path.

Therefore, multicriteria or multiobjective shortest path search emerged as a natural extension of the aforementioned task. In contrast to the single-criteria problem, multicriteria shortest path search is aimed at producing a path minimizing multiple objective parameters at the same time. Instead of a single path with minimum cost, its solution is represented by a Pareto set of optimal solutions. This set contains paths that are not dominated (worse according to all criteria) by any of the other possible paths. Due to the criteria often being in conflict, there are usually multiple Pareto solutions and the set can grow quite large in size. Multiple algorithms solving the shortest path problem were successfully adapted to the multicriteria version (i.e. Dijkstra’s algorithm [1], A* algorithm [2]). However, the size of a Pareto set grows exponentially in the size of the graph [3], thus making the problem NP-hard. Moreover, the solution size is also exponential in the number of criteria considered. As a result, standard versions of the algorithms tend to have unacceptably high runtimes on larger graphs such as country-size road maps.

To alleviate this issue, a number of acceleration techniques were developed. Unfortunately, a major part of the techniques in consideration either can only be applied to one-to-one search (i.e. guiding heuristics for A*) or does not preserve the solution optimality. On the other hand, most research papers address mainly the bicriteria setting. And although the overwhelming majority of the methods can be applied to an arbitrary number of criteria, an increase in the parameter dimensionality poses further limitations.

While one-to-one search is perfectly sufficient for the majority of the real-world scenarios, we are sometimes interested in paths to multiple goals. For instance, longer routes for electric vehicles (EVs) have to be planned with the locations of charging stations in mind due to their relatively limited numbers. The shorter travel range of EVs changes the task into multi-stop routing with charges. To solve it, we have to plan routes to and between the multitudes of charging stations and select the optimal combination. On the other hand, multicriteria routing is highly dependent on driver priorities, which can change in the process. Running a many-to-one query to the destination and storing the results would allow us to perform dynamic rerouting with no additional computations. In practice, these scenarios involve hundreds of goals, making the repetitive one-to-one query approach inefficient. On the other hand, the existing algorithms for one-to-many queries are too slow to be applied to any sort of a real-world scenario.

Thus, the objective of this paper is to introduce a practical algorithmic approach for fast one-to-many path search in the context of multicriteria route planning. The proposed method combines a swift preprocessing phase with dimensionality reduction aimed at dominance check simplification. The graph preprocessing algorithm constructs a compressed $k$-Path-Cover [4] of the initial graph. The compressed graph...
TABLE I

| NOTATIONS | \(\gamma = (\gamma_1, \ldots, \gamma_t) \in \mathbb{R}^t\) | vector of real values |
| \(\gamma^t = (\gamma_2, \ldots, \gamma_t) \in \mathbb{R}^{t-1}\) | truncated version of vector \(\gamma\) |
| \(V\) | weak dominance |
| \(E\) | set of vertices |
| \(I \subseteq V\) | set of edges |
| \(G = (V, E, I)\) | set of goals |
| \(u, v, w\) | graph |
| \(e\) | arbitrary vertices |
| \(k\) | arbitrary edge |
| \(\Pi_k\) | set of all \(k\)-sized simple paths in \(G\) |
| \(\pi\) | an arbitrary simple path in \(G\) |
| \(v^k\) | vertices of kPC graph |
| \(E^k\) | edges of kPC graph |
| \(G^k = (V^k, E^k, I)\) | kPC graph |
| \(l = (v, (\gamma_1(v), \ldots, \gamma_t(v)))\) | label of vertex \(v\) |
| \(<_L\) | lexicographic precedence |
| \(perm(v)\) | set of permanent labels of \(v\) |
| \(temp(v)\) | set of temporary labels of \(v\) |

has far fewer vertices than the initial one, but preserves the paths necessary for optimal solutions by containing parallel edges. Multicriteria label-setting algorithm [1], one of the few solution algorithms for multicriteria shortest path search, is modified to operate on this graph, further augmented with dimensionally reduced dominance checks [5] that tackle the increased number of edges. This combination of three techniques results in a highly flexible algorithm able to be tailored to a variety of specific scenarios. It has far smaller time requirements and memory footprint than the existing alternatives, having the only downside of requiring a preprocessing stage that needs to be performed only once. Furthermore, the algorithm can be easily modified to produce even faster one-to-one queries or heuristic solutions using existing techniques.

The layout of this article is as follows. Section II provides a brief overview of the related research. Section III contains the theoretical problem statement. The individual parts of the proposed algorithm and the order of their combined operation are described in Section IV. Section V is dedicated to the empirical evaluation of the different aspects of the proposed algorithm. A conclusion to the article is provided in Section VI. Table I provides an overview of notations used throughout the paper.

II. RELATED WORK

The earliest algorithms solving the multicriteria shortest path search are based on Dijkstra’s algorithm [6]. One of the first algorithms was proposed by Martins [1] in 1984. The multicriteria label-setting algorithm (MLS) proposed by Martins maintains a Pareto set of optimal labels, expanding the lexicographically smallest one during every iteration. Lexicographic ordering allows the algorithm to only expand non-dominated labels, so a processed label is guaranteed to belong to the optimal Pareto set of a node. In contrast, the label-correcting approach [7] expands labels in the first-in-first-out (FIFO) fashion, compensating for the expansion of non-optimal labels with the time saved on ordering absence.

Goal-directed one-to-one techniques were also extended successfully to manage multiple objective parameters. Namely, MOA* [2] is a modification of A* algorithm [8] designed specifically for multicriteria shortest path search. Its improved version called NAMOA* [9] was proposed later.

Accelerating techniques for multicriteria shortest path search can be separated into two categories: optimality preserving or not. There are few optimality preserving techniques for the given problem, and even fewer can be used effectively for one-to-many scenarios such as multi-stop planning and dynamic rerouting. One of the existing methods is based on the observation that label-setting queries are parallelizable [10]. The label-setting property of the algorithm remains if a subset of globally Pareto optimal labels are expanded in parallel. During every iteration, the approach identifies a subset of such labels and scans these in parallel. While this approach can be effectively implemented for bicriteria cases, its generalization to higher dimensions is non-trivial since no efficient way to identify and maintain globally optimal labels has been proposed yet.

Recently, multiple other modifications were tested on one-to-one cases. While earlier propositions make use of stopping conditions and bidirectional searches [11], more recent approaches make use of other non-trivial techniques. Reference [12] proposes a heavily modified version of MLS the authors named Multiobjective Dijkstra Algorithm (MDA). While this method expands labels in lexicographic order, it only maintains one label per vertex at any given time, creating a new label for a vertex whenever its current one is processed and expanded. While the algorithm essentially conducts one-to-all path search, it was modified for experiments on one-to-one instances by precomputing criteria limits. A label-correcting algorithm called LCDPF [13] is another recent proposition. Operating in two phases, it first computes the supported solutions of the Pareto set and solution boundaries through multiple runs of classic Dijkstra algorithm. The second phase computes the rest of the Pareto set using modified label-correcting search.

Due to the high complexity of the task, suboptimal solutions received more attention in recent years. Some approaches employ standard label-setting or label-correcting techniques combined with relaxation heuristics. These include \(\epsilon\)-dominance, cost-based pruning, ellipse pruning, etc. [14], and operate by discarding solutions that are considered irrelevant based on their similarity or geographical features. Other state-of-the-art methods, however, are based on metaheuristic techniques: simulated annealing [15], ant colony optimization [16], and evolutionary algorithms [17].

III. ONE-TO-MANY MULTICRITERIA SHORTEST PATH SEARCH PROBLEM

Multicriteria optimization is concerned with optimization of multiple objective criteria simultaneously. These objectives are often conflicting, which excludes the possibility of finding a single optimal solution. Instead, one can build a set of Pareto optimal solutions. A solution is called Pareto optimal if it cannot be improved in any of the criteria without degrading
at least one of the other criteria. In other words, each of the
Pareto optimal solutions provides a possible trade-off between
the criteria. A Pareto set is thus a set of all Pareto optimal
solutions. To formalize this notion, multicriteria optimization
defines the dominance property. The relation of weak dom-
inance of a vector \( \gamma' = (\gamma'_1, \ldots, \gamma'_q) \in \mathbb{R}^q \) by a vector
\( \gamma = (\gamma_1, \ldots, \gamma_q) \in \mathbb{R}^q \) is defined as follows:
\[
\gamma \succeq \gamma' \iff \forall i \in \{1, \ldots, q\} \left\{ \begin{array}{ll}
\gamma_i \leq \gamma'_i & \text{if criterion } i \text{ is minimized} \\
\gamma_i \geq \gamma'_i & \text{if criterion } i \text{ is maximized}.
\end{array} \right.
\]
(1)

Accordingly, \( \gamma \) dominates \( \gamma' \) (\( \gamma \succ \gamma' \)) iff \( \gamma \succeq \gamma' \) and \( \gamma' \nsucceq \gamma \).

Let \( G = (V, E, I) \) be a finite labeled directed graph with \( |V| \) vertices and \( |E| \) edges. Every edge \( e = (u_i, v_j) \in E \) starting at a vertex \( v_j \in V \) and ending at \( v_j \in V \) has an associated vector of criteria-costs. A path in \( G \) is any sequence of nodes \( \pi = v_1, v_2, \ldots, v_q \) such that for all \( i < n, (v_i, v_{i+1}) \in E \). A path can also be represented by its compound edges \( \pi = e_1, e_2, \ldots, e_{n-1} \), where it holds \( \forall i \in \{1, \ldots, n - 1\} : e_i = (v_i, v_{i+1}) \in E \). The cost vector of \( \pi \) is defined by a function incorporating cost vectors of edges included in \( \pi \). In this research, we use the sum-type criterion function \( \gamma^\pi = (\sum e_i \gamma'_1, \ldots, \sum e_i \gamma'_q), e \in \pi \) for all criteria, where the criterion value of a path is calculated as the sum of the corresponding criterion values of all its edges. However, this is not necessarily the case in other scenarios, since the function is defined by the specific setting and the criteria in consideration. For a given source vertex \( s \) and a subset \( I \subseteq V \) of goal points, the task is to find the Pareto sets of optimal routes to all the goal vertices.

IV. MULTICRITERIA SHORTEST PATH SEARCH ALGORITHM

As it was stated earlier, the overall approach is a combi-
nation of three techniques: a graph preprocessing method,
a query algorithm, and a dimensionality reduction technique
exploited in queries. This section describes each of these
independently and provides a layout of the combined approach
afterwards.

A. Graph Preprocessing

In modern path planning algorithms, graph preprocessing
is a commonly used technique. An important benefit of
a preprocessing technique is the ability to perform it only once
on a single graph instance and save the results for subsequent
usage, which is especially beneficial when operating on large
instances such as road networks. As long as the cost function
and the structure of the graph are unchanged, its preprocessed
version remains actual. This gives developers the possibility to
perform a preprocessing operation with high resource demands
in advance and save its results for subsequent usage. The
article addresses the “one-to-many” version of the problem as
opposed to “one-to-all” due to the fact that the preprocessing
phase constructs a cover with a small subset of original vertices
for the query to run on. This assumption is sensible in real-
world scenarios. For example, a driver planning a route to the
nearest fuel station or pharmacy is not interested in routes to
book stores, which eliminates the necessity to calculate and
maintain Pareto sets for their corresponding vertices.

1) Vertex Cover Construction: The approach in discussion
uses a compressive preprocessing technique called \( k \)-(All-Path
Covers (kPC) [4]. For a given directed graph \( G = (V, E, I) \)
and a size constant \( k \in \mathbb{N}_+ \), the set \( \Pi^k \) is the set of all possible
simple paths of size \( k \) in \( G \). Then, a \( k \)-Path cover of \( G \) is a
subset \( \Pi^k \subseteq V \) such that for every simple path of size \( k \) in \( G \), at least one of its vertices must be in
the cover. For a given graph and a size constant, there is a \( k \)-Path
cover of minimum size. However, the complexity of finding
the minimum \( k \)-Path cover was proven to be APX-hard [18],
which makes its generation intractable in practice. Thus,
\( k \)-Path covers of suboptimal sizes are more appropriate for
the method to be of use in real-world problem solving.

Since the naive enumerative approach for \( k \)-Path cover
generation would have unacceptably high runtimes on larger
graph instances, Funke et al. [4] proposed another approach
based on iterative pruning. Initially, all the nodes in the graph
are assumed to belong to the cover, i.e., \( V^k = V \). The nodes
are examined one by one in a chosen order. For every node
\( v \in V^k \), the algorithm tests if it must stay in the cover to
maintain the path property. In order for it to hold after the
removal of \( v \), there must exist no path of size \( k \) containing
\( v \) such that it does not include any other vertex from \( V^k \).
If such a path is found, then \( v \) remains in the cover. Essentially,
to test this condition one must explore all incoming and
outgoing paths of \( v \) reaching other nodes of \( V^k \). If there is
a simple combination of an incoming and an outgoing path
with cumulative size at least \( k \), \( v \) is kept in \( V^k \).

A pseudocode of this method outlined in Algorithm 1 shows
two stages. The algorithm is prohibited from removing the
subset of goal points \( I \) from the cover (lines 3-4). During
the first stage (lines 5-7), all simple paths outgoing from
\( v \) and ending at another cover node are enumerated. This
can be performed using a two-step technique based on
depth-first search (DFS). First, DFS starts from the examined vertex and
traverses only outgoing edges without visiting other cover
vertices. If a path with size \( k \) is found this way, the second
step can be omitted, since the necessity of \( v \) in \( V^k \) is already
proven. Otherwise, the second step (lines 8-15) attempts to
extend every enumerated path to \( k \) by connecting it to an
incoming path. This step is also based on the DFS method,
though the edges must be traversed backwards. It is important
to note that the algorithm only explores simple paths of length
at most \( k \), since \( V^k \) is assumed to be a feasible cover before
examination of \( v \). The correctness of the algorithm can be
proved trivially through mathematical induction.

2) Edge Cover Construction: In order for a shortest path
search algorithm to be able to operate on the cover, the cover
vertices must be directly connected by cover or overlay edges
(lines 16-26). This is achieved in a fashion similar to the vertex
cover construction. For every vertex \( v \in V^k \) in the completed
cover, DFS search is started from it. Whenever the search
meets another cover vertex \( w \in V^k \), a new edge from \( v \) to \( w \)
is added to the overlay graph with its cost vector being the
Algorithm 1 kPC Construction

| Line | Description |
|------|-------------|
| 1    | Data: graph $G = (V, E, I)$, path size $k$ |
| 2    | Result: kPC graph $G^k = (V^k, E^k, I)$ |
| 3    | /* node cover construction */ |
| 4    | $V^k = V$ |
| 5    | for $\forall v \in V^k$ do |
| 6    | if $v \in I$ then |
| 7    | continue |
| 8    | $P_o$ = the set of all outgoing paths from $v$ not containing other nodes from $V^k - \{v\}$ |
| 9    | if $\exists \pi_o \in P_o$ such that $|\pi_o| \geq k$ then |
| 10   | continue |
| 11   | remove = true |
| 12   | for $\forall \pi_o \in P_o$ do |
| 13   | $\pi_i$ = an incoming path to $v$ not containing nodes from $(V^k \cup \pi_o) - \{v\}$ such that $|\pi_i| + |\pi_o| - 1 = k$ |
| 14   | if $\exists \pi_i$ then |
| 15   | remove = false |
| 16   | break |
| 17   | if remove = true then |
| 18   | remove $v$ from $V^k$ |
| 19   | /* edge cover construction */ |
| 20   | for $\forall v \in V^k$ do |
| 21   | perform DFS search starting from $v$ |
| 22   | if a node $w \in V^k$ is met during search then |
| 23   | save the current path to new edge $e$ |
| 24   | for every other cover edge $e'$ from $v$ to $w$ do |
| 25   | if $e \geq e'$ then |
| 26   | remove $e'$ from $E^k$ |
| 27   | else if $e' \geq e$ then |
| 28   | discard $e$ |
| 29   | break |
| 30   | if $e$ is not discarded, add it to $E^k$ |

Standard domination pruning tests domination of an edge by every other edge in $E'$, which gives $|E'|^2$ domination checks for a pair of vertices. Since quadratic runtime would be very costly on large graphs with high $k$ size values, another approach can be used. For every considered criterion $i \in \{1, \ldots, q\}$, one can find the edge $e'_i \in E'$ having the optimal value $c_i$ among all edges in $E'$. Afterwards, all other edges in $E'$ can be checked against these optimal edges, which would only yield $q|E'|$ comparisons. Although dominance pruning can be performed after overlay edge construction is finished, a far more preferable option is to perform it online, whenever a new cover edge is formed (lines 20-25).

Another pruning technique mentioned in [19] is triangle pruning. Let there be three cover vertices $u, v, w$ such that there are cover edges between the pairs $(u, v), (u, w), (v, w)$. Triangle pruning removes from the edge set of $(u, w)$ edges that are dominated by at least one combined path $(u, v, w)$. This pruning method can also be accelerated similarly to domination pruning. Out of all of the produced edge combinations of $(u, v)$ and $(v, w)$, one can extract the combinations with the optimal criteria values and evaluate the $(u, w)$ set against these. Unlike dominance pruning, which can be performed on the fly, triangle pruning must be performed after all of the cover edges are formed due to the fact that it operates on multiple cover edge sets at a time.

B. Multicriteria Label-Setting Algorithm

Multicriteria label-setting algorithm (MLS) [1] is one of the most prominent algorithms to solve the multicriteria shortest paths search problem. As it was stated earlier, it can be considered an extension of Dijkstra’s algorithm to multicriteria shortest path search problem. Since the vast majority of the original cover edges produced by DFS are exceedingly long, these may have unacceptably high memory requirements. While online dominance pruning may require a higher number of operations, it allows the construction algorithm to operate on a much smaller amount of memory.

As one can observe, the resulting overlay graph is not simple due to there being multiple possible paths between a pair of non-adjacent vertices. Moreover, a significant part of the routes produced by this method is redundant in context of shortest path search, since DFS enumerates and saves all the possible paths, even the unnecessarily long and winding ones. Therefore, multiple edge pruning techniques are described in [19], the essential one being domination pruning. In short, for a set of overlay edges $E'$ between a pair of cover vertices, an edge $e' \in E'$ is pruned from $E'$ if there is another edge $e \in E'$ such that $e \geq e'$, i.e. $e$ (weakly) dominates $e'$.

Algorithm 2 MLS Construction

| Line | Description |
|------|-------------|
| 1    | Data: graph $G = (V, E, I)$, path size $k$ |
| 2    | Result: MLS graph $G^k = (V^k, E^k, I)$ |
| 3    | /* node cover construction */ |
| 4    | $V^k = V$ |
| 5    | for $\forall v \in V^k$ do |
| 6    | if $v \in I$ then |
| 7    | continue |
| 8    | $P_o$ = the set of all outgoing paths from $v$ not containing other nodes from $V^k - \{v\}$ |
| 9    | if $\exists \pi_o \in P_o$ such that $|\pi_o| \geq k$ then |
| 10   | continue |
| 11   | remove = true |
| 12   | for $\forall \pi_o \in P_o$ do |
| 13   | $\pi_i$ = an incoming path to $v$ not containing nodes from $(V^k \cup \pi_o) - \{v\}$ such that $|\pi_i| + |\pi_o| - 1 = k$ |
| 14   | if $\exists \pi_i$ then |
| 15   | remove = false |
| 16   | break |
| 17   | if remove = true then |
| 18   | remove $v$ from $V^k$ |
| 19   | /* edge cover construction */ |
| 20   | for $\forall v \in V^k$ do |
| 21   | perform DFS search starting from $v$ |
| 22   | if a node $w \in V^k$ is met during search then |
| 23   | save the current path to new edge $e$ |
| 24   | for every other cover edge $e'$ from $v$ to $w$ do |
| 25   | if $e \geq e'$ then |
| 26   | remove $e'$ from $E^k$ |
| 27   | else if $e' \geq e$ then |
| 28   | discard $e$ |
| 29   | break |
| 30   | if $e$ is not discarded, add it to $E^k$ |

cost function value of the found path. Further path expansion through $u$ is unnecessary. As a result, the overlay edges only connect pairs of directly adjacent cover vertices, i.e. cover vertices connected by a path in the original graph which does not contain any other cover vertex.

As one can observe, the resulting overlay graph is not simple due to there being multiple possible paths between a pair of non-adjacent vertices. Moreover, a significant part of the routes produced by this method is redundant in context of shortest path search, since DFS enumerates and saves all the possible paths, even the unnecessarily long and winding ones. Therefore, multiple edge pruning techniques are described in [19], the essential one being domination pruning. In short, for a set of overlay edges $E'$ between a pair of cover vertices, an edge $e' \in E'$ is pruned from $E'$ if there is another edge $e \in E'$ such that $e \geq e'$, i.e. $e$ (weakly) dominates $e'$.
preceding \( \gamma' \) in lexicographic ordering is defined as
\[
\gamma <_t \gamma' \quad \text{iff:}
\]
\[
\begin{align*}
&\exists n \in \{1, \ldots, q\} : \{\forall i \in \{1, \ldots, n-1\} : \gamma_i = \gamma'_i\} \quad \text{and} \\
&\begin{cases}
\gamma_n < \gamma'_n & \text{if } n \text{ is minimized} \\
\gamma_n > \gamma'_n & \text{if } n \text{ is maximized}
\end{cases} \\
\text{OR} \\
\forall i \in \{1, \ldots, q-1\} : \gamma_i = \gamma'_i \quad \text{and} \\
\begin{cases}
\gamma_q \leq \gamma'_q & \text{if } q \text{ is minimized} \\
\gamma_q \geq \gamma'_q & \text{if } q \text{ is maximized}
\end{cases}
\end{align*}
\]

MLS starts by generating a blank label for the source node and adding it to an empty lexicographically ordered priority queue and the temporary label set of the source node. Temporary and permanent label sets of all other vertices are empty. After that, the main algorithm loop starts to operate. At each iteration, the lexicographically smallest label \( l = (v, (\gamma_1(v), \ldots, \gamma_q(v))) \) is extracted from the priority queue. It is also moved from the temporary set of its corresponding vertex to the permanent one. Label expansion proceeds as follows: for every adjacent vertex \( w, (v, w) \in E \), a new label \( l_{\text{new}} = (w, (\gamma_1(w), \ldots, \gamma_q(w))) \) is created. It is then compared to the labels in \( \text{perm}(w) \) and \( \text{temp}(w) \). If \( l_{\text{new}} \) is dominated by at least one of these labels, it is discarded. On the other hand, if \( l_{\text{new}} \) dominates a label \( l' \in \text{temp}(w) \), \( l' \) is removed from \( \text{temp}(w) \) and from the priority queue. It is worth mentioning that \( l_{\text{new}} \) cannot dominate any label from \( \text{perm}(w) \). If the dominance tests are completed and \( l_{\text{new}} \) is not dominated, it is added to \( \text{temp}(w) \) and to the priority queue. The algorithm terminates when the priority queue is emptied, and permanent sets of vertices contain their exact Pareto sets.

In order for the algorithm to be able to reproduce the path corresponding to an individual entry of a Pareto set, every label must also include a pointer to its parent label, which is omitted in this description. The desired path can then be extracted by traversing labels through the parent pointers from the preferred entry of the goal vertex up to the source. The operation closely resembles the Dijkstra’s path extraction procedure.

C. Dimensionality Reduction

A significant runtime part of MLS as well as other multicriteria shortest path search algorithms is spent on dominance checks. These time requirements grow with the number of criteria to be considered. Although most multicriteria methods can be limited to bicriteria cases without loss of generality, in practice addition of a single criterion may have a dramatic effect on the overall performance of an algorithm. For this reason, a dimensionality reduction technique applicable to multicriteria label-setting algorithms was developed [5].

Given a vector \( \gamma = (\gamma_1, \ldots, \gamma_n) \), its truncated vector \( \gamma' \) is \( \gamma \) without the first component, i.e. \( \gamma' = (\gamma_2, \ldots, \gamma_n) \). For a set of vectors \( S \), its corresponding set of truncated vectors or \( t \)-set is the set of non-dominated truncated vectors \( S' = \{\forall \gamma' \in S : \not\exists \gamma'' : \gamma'' \geq \gamma'; \gamma, \gamma' \in S\} \). Dimensionally reduced weak dominance check is based on \( t \)-discarding operation: given a set of vectors \( S \) and a vector \( \gamma \), \( \gamma \) is \( t \)-discarded by \( S \) if \( \forall \gamma' \in S : \gamma'_1 \leq \gamma_1 \) and there is a vector \( \phi \in S \) such that \( \phi'_1 \geq \gamma'_1 \).

According to the theoretical proof in [5], \( t \)-discarding can partially replace regular dominance checks provided the labels are expanded in the lexicographic order. In MLS, this is ensured by the priority queue. Given this, regular dominance tests of a new label \( l \) of a vertex \( v \) against the set of permanent labels \( \text{perm}(v) \) can be replaced by checking if \( \text{perm}(v) \) \( t \)-discards \( l \). In the case of weak dominance, the first condition holds automatically. If all the vectors in \( \text{perm}(v) \) are lexicographically smaller than the vector of \( l \), the maximum value of the first criterion in \( \text{perm}(v) \) is necessarily not greater than that of \( l \). To check if the second condition holds, it is necessary to check the truncated criteria vector of \( l \) against all vectors in \( \text{perm}(v) \).

D. Combined Algorithm

After all key parts have been described, it is now possible to lay out the overall approach. Given a graph \( G = (V, E, I) \) and a subset of interest points \( I \subseteq V \), the algorithm first builds a kPC of the graph and interconnects it with non-dominated cover edges. As it was stated earlier, the cover graph \( G^k = (V^k, E^k, I) \) remains valid until the initial graph is changed, so the cover can be read from memory if it was built beforehand. When the preprocessing phase is finished, the algorithm can perform queries on the prepared cover. Algorithm 2 presents the query pseudocode.

In order for the query phase to operate correctly on a cover graph, several adjustments must be made to standard MLS. The overlay graph is not necessarily simple, hence one transition between two neighboring cover nodes can result in different cost vectors (corresponding to different paths between the pair in the original graph). Additionally, the source vertex may not always belong to the cover.

In cases when the source vertex is not a part of the kPC, it is connected to the overlay at the beginning of the query (line 5). This can be done similarly to the edge building procedure performed during the overlay construction. An enumerative DFS search operates from the source vertex on the original graph. Whenever it meets a cover vertex, the path is saved if it is not dominated by any other path. If a goal vertex not included in the overlay is defined, it can be added to the overlay similarly. The only difference is that for incoming overlay edges to be found, the DFS search starting from the goal must traverse edges only backwards.

Due to the overlay graph having parallel edges, the modified search must iteratively expand to neighbor vertices through all of them (lines 14, 15). Therefore, while the overall number of iterations is decreased, the number of domination checks relative to the iteration number is significantly higher than that of MLS on the original graph. However, this circumstance...
Algorithm 2 t-Discarding kPC MLS Search

Data: graph \( G = (V, E, I) \), its kPC cover \( G^k = (V^k, E^k, I) \), source vertex \( s \)

Result: Pareto sets of paths from \( s \) to the goal set \( T \)

1. for \( \forall v \in V^k \) do
   2. \( \text{perm}(v) = \emptyset \)
   3. \( \text{temp}(v) = \emptyset \)
   4. \( \text{tset}(v) = \emptyset \)
5. \( \text{connectSourceToKpc}(G, G^k, s) \)
6. queue = empty lexicographically ordered priority queue
7. queue.push(source label)
8. while queue is not empty do
    9. \( l = \text{queue.pop()} \)
    10. \( v = l . \text{vertex} \)
    11. \( \text{temp}(v) . \text{remove}(l) \)
    12. \( \text{perm}(v) . \text{add}(l) \)
    13. \( \text{tset}(v) . \text{update}(l) \)
    14. for \( \forall w \in V^k \) adjacent to \( v \) do
        15. for \( \forall e \in E^k \) connecting \((v, w)\) do
            16. \( l_{\text{new}} = \text{addSuffix}(l, e) \)
            17. if path of \( l_{\text{new}} \) is infeasible then
                18. continue
            19. \( /* \text{domination check} */ \)
            20. if \( \text{tset}(w) \) t-discards \( l_{\text{new}} \) then
                21. continue
            22. \( \text{is-dominated} = \text{false} \)
            23. for \( \forall l' \in \text{temp}(w) \) do
                24. if \( l' \geq l_{\text{new}} \) then
                    25. \( \text{is-dominated} = \text{true} \)
                    26. break
            27. if \( \text{is-dominated} = \text{true} \) then
                28. continue
            29. for \( \forall l' \in \text{temp}(w) \) do
                30. if \( l'_{\text{new}} \geq l' \) then
                    31. \( \text{remove } l' \text{ from } \text{temp}(w) \) and from queue
                    32. \( \text{temp}(w) . \text{add}(l_{\text{new}}) \)
                    33. \( \text{queue.push}(l_{\text{new}}) \)
34. \( \text{paretoSet} = \emptyset \)
35. for \( \forall i \in T \) do
    36. \( \text{paretoSet}(i) = \text{perm}(i) \)
37. return \( \text{paretoSet} \)

Further increases the relative speedup provided by dimensionality reduction, resulting in a synergy between the two techniques.

\( t \)-sets of the nodes are updated online: whenever a new label is added to the closed set, its truncated parameter vector is checked against the current \( t \)-set (lines 12, 13). If the candidate is not dominated by any of the included vectors, it is added to the \( t \)-set while simultaneously discarding all the vectors dominated by it.

As can be seen, the query returns labels of path cost vectors belonging to the Pareto set. To facilitate the retrieval of a full path from the set, every label must contain a pointer to its parent label and an ID of the edge between them. The full path is then produced by iteratively stepping to the parent label up to the source while inserting the intermediate vertices contained in the cover edges used.

Thus, the resulting algorithm is able to retrieve optimal paths to multiple target vertices while being significantly more effective in terms of time and memory requirements. In addition to the efficiency, the algorithm can be modified for one-to-one queries using standard query limitation techniques. In our one-to-one experiments, we used two standard bound techniques. Running single-criterion Dijkstra queries for each criterion on the cover graph, we compute upper \( \bar{y}^v \) and lower \( \gamma^v \) criteria bounds for every vertex \( v \in V^k \). Then, for every new label \( l_{\text{new}} \) of a vertex \( v \in V^k \), we compute its expected criteria \( l_{\text{new}}^t = l_{\text{new}} + \gamma^v \). Using this expectation, we can safely discard \( l_{\text{new}} \) if \( \bar{y}^v \geq l_{\text{new}}^t \) (upper source bounds dominate the expected path) or \( \text{tset}(t) \) t-discards \( l_{\text{new}}^t \) (an existing path to the target dominates the expected one). Both of these techniques and their proofs are described in greater detail in [12].

V. Evaluation

The algorithm was implemented in C++ and compiled using GCC 8.5.0 with optimization parameter \(-03\). The implementation used only standard C++ libraries, but the structures and procedures such as domination checks were optimized for specific instances. The machine the experiments were run on operated on 2.8 GHz AMD EPYC 7543 with 32 GB RAM allocated.

A. Experiment Setting

First, the algorithm was tested in its one-to-many form on two graphs. The first graph represents the roadmap of Bavaria,\(^1\) Germany. The second one is a graph of the entire Germany.\(^2\) In order to test the method on multiple criterion combinations of different sizes and correlation degrees, some of the criteria for the graph were produced artificially. The time parameter \( t \) computed from the source was treated as the ground criterion used for the subsequent generation of the correlated criteria. While the structure of the graphs was left unaltered, the parameter vectors of edges were set to create seven datasets of varying nature. Table II provides the overview of the criterion combinations that were used for the experiments. While the fully random criteria are not in any way correlated with the ground parameter, the formula \( t + ct \) is expected to produce values somewhat close to those of the ground criterion. To avoid confusion, we name all of these instances using three symbols: B or G stand for Bavaria or Germany graph, 2 and 3 display the number of

\(^1\)https://download.geofabrik.de/europe/germany/bayern.html
\(^2\)https://download.geofabrik.de/europe/germany.html
Fig. 1. Bavaria graph visualization parts produced using kepler.gl. Yellow dots represent vertices, white lines are graph edges.

TABLE II

| Name | p1 | p2 | p3 |
|------|----|----|----|
| 2-C  | t  | r  | r  |
| 2-U  | t  | r  | r  |
| 3-C  | t  | r  | r  |
| 3-U  | t  | r  | r  |

used criteria, and C and U mean these criteria were correlated or uncorrelated respectively. Thus, the dataset B-3-C is built on the Bavaria graph with 3 correlated criteria.

The second group of experiments was conducted with a one-to-one version of the algorithm. To compare the proposed approach to existing alternative algorithms, we used the benchmark graphs from the 9th DIMACS implementation challenge. The datasets are unaltered instances of a real-world scenario considering two parameters: distance and traversal time. The sizes of all graphs are presented in Table III.

For all datasets, the algorithm is set to minimize all of the objective criteria. Simultaneously, a criterion value of a path is calculated as the sum of corresponding values of its included edges for all criteria. During every experiment, a source vertex and a subset of goal vertices were picked at random from the whole dataset. Since some of the instances were not solved using the mentioned hardware due to extreme time or memory requirements (especially for standard MLS), the efficiency comparison is only done on instances solved by all of the tested algorithms. The goal subsets were set to size 1000 on the Bavaria graph and 6000 on the Germany graph.

B. kPC Evaluation

First of all, the work of kPC technique was tested. As the experiments described in [4] show, setting the path size value $k$ above 32 generally results in only a minor performance gain while increasing the algorithm runtime dramatically. For this reason, the experiments were conducted with $k = 32$ one-to-many instances and $k = 16$ on one-to-one instances, since they took significantly more time to have a cover built. The only pruning technique used in kPC construction was domination pruning. The nodes were processed in the increasing vertex ID order. Figure 1 presents a visual comparison of a part of the original Bavaria graph and its kPC cover. Table IV shows the average overlay graph sizes. The vertex cover is independent of the criteria maintained due to the fact that cover vertices are only selected based on the graph structure. In contrast, the final number of overlay edges is influenced not only by the cover vertex layout, but also by the proportion of pruned cover edges. This proportion is in turn defined by the considered criteria. As expected for a fixed criterion number, the variety of cover edges is significantly decreased if the criteria are interconnected. On the other hand, the increase in criterion number results in a higher number of non-dominated cover edges, although, in this particular case, the correlation between the criteria has a stronger effect than their amount.

It can be seen from the table that the vertex quantity in the covers for all of the one-to-many datasets is close to 10% of the original vertex number, and the number of cover edges comprises only a fraction of the original ones. As can be seen, overlay construction for the graphs of Bavaria and Germany is relatively fast and has duration independent of the criteria. On the other hand, DIMACS graphs appear to be much more complicated to process, presumably due to their complex and often grid-like structure. Fortunately, these increased time requirements are of no significant importance since the preprocessing only has to be performed again if

TABLE III

| Graph  | Vertex # | Edge # |
|--------|----------|--------|
| Bavaria| 294727   | 587782 |
| Germany| 1521776  | 2947009|
| NY     | 264346   | 733846 |
| BAY    | 321270   | 800172 |
| COL    | 435666   | 1057066|
| FLA    | 1070376  | 2712798|
| NE     | 1524453  | 3897636|

3http://users.diag.uniroma1.it/challenge9/download.shtml
the combinatorial structure of the graph is changed. However, if the cover graph must be built in a shorter time, one can decrease the $k$ value of the algorithm while sacrificing a portion of the provided speedup.

C. Optimal Runtime Evaluation

In order to properly analyze the influence of individual parts of the combined algorithm, we tested each of the 4 possible combinations: MLS – basic multicriteria label-setting, $t$-MLS – MLS with dimensionality reduction through $t$-discarding, kPC-MLS – basic MLS operating on the kPC graph, $t$-kPC-MLS – MLS with dimensionality reduction through $t$-discarding operating on the kPC graph. To make the experiments fair, every one started with a randomly chosen source vertex, from which every technique combination was run one after another.

As expected, classic MLS is the slowest of the method combinations considered in the research. Therefore, it was used as a benchmark for other tested techniques. Unfortunately, the G-3-U instance that is not presented here was too hard to be solved with sufficient consistency. Additionally, the B-3-U dataset would take over 24 hours to be solved by MLS and hit memory limits on $t$-MLS in some of the instances, so the results only consider instances solved by the three modified approaches. A notable circumstance visible from the table are rather high variance values across all datasets. This is caused by a significant presence of outliers, some of which exceed the average runtime values by tens of times. Result examination shows that these outliers are produced by queries starting from far corners of the graphs.

As Tables V and VIII show, despite being extremely efficient under certain conditions, dimensionality reduction through $t$-discarding is not always of use. On instances with heavily correlated criteria, it provides underwhelming results. For instance, while it was able to provide a slight speedup on B-2-C, it slows down kPC-MLS queries on B-3-C. $t$-discarding has an advantage in bicriteria scenarios, using optimized operations where a dominance check against a full set of permanent labels is replaced by a single value comparison. A logical conclusion is that $t$-discarding is of little use on small/simple problem instances. This can be explained by two circumstances. Firstly, the Pareto set sizes for correlated criteria are relatively small, therefore the search algorithm does not spend as much time performing domination checks as it does for uncorrelated criteria. This makes the acceleration provided by $t$-discarding negligible. Secondly, maintenance and regular updates of truncated vector sets require the algorithm to perform additional operations. However, this is only the case for heavily correlated parameters, present in datasets 2-C and 3-C. On uncorrelated datasets $t$-discarding demonstrates notable effectiveness, accelerating the queries by at least 5 times.

On the other hand, kPC provides more stable speedup results, though not always as high as $t$-discarding. The minimum speedup provided by kPC is at least 3 times, making it a useful technique even for smaller datasets. Interestingly, kPC provides a significantly higher speed gain on correlated datasets, which can be explained by the fact that the corresponding graphs have lower numbers of edges. Nevertheless, the experiments show that kPC influences the query performance positively on all datasets. An important quality is that kPC significantly reduces not only time requirements of the searching algorithm, but also the volume of necessary memory. Since the memory is mostly used to store the path labels, a query operating on a cover graph would have several times lower memory requirements. In fact, a cover graph consisting of only 10% of the original vertices would yield on average 10 times less labels. In practice, the expected memory saving of kPC-MLS compared to standard MLS would be somewhat lower due to the necessity of storing the cover graph in addition to the original one.

Combined, $t$-discarding and kPC amplify their individual results. The tendency of $t$-discarding being inefficient on heavily correlated instances becomes more evident, but this result is mitigated by kPC, and the combination of these techniques is still dramatically more efficient than standard MLS. As one can expect, the highest gain is achieved if both techniques are able to provide some acceleration individually. Combined, these form a synergy that results in far better results than the sum of their individual gains. The lowest gained acceleration being achieved on B-2-C, the easiest of the instances, exceeds 5 times over standard MLS, rising all the way to 24 times on B-2-U. Unfortunately, we weren’t able to compare it to MLS on the hardest instances of B-3-U and G-3-U, but there is little doubt that the gained speedup would be even higher. These results confirm the aforementioned synergy: kPC decreases the number of vertices and labels to be processed, while $t$-discarding accelerates domination checks and compensates for their increased number.

D. Memory Requirements Evaluation

Due to the sizes of Pareto sets having exponential dependency on graph sizes and criteria numbers, the amount of memory required for queries on large graphs should also be evaluated and carefully considered. Since the memory consumed by the graph and the kPC cover is constant and negligible compared to that of the Pareto sets, it can be ignored in the measurements. Therefore, we measure the average sizes of Pareto sets for the tested instances.

Table VII provides the estimates of the average and maximum numbers of labels produced during the experiments.

**TABLE IV**

| dataset | $k$ = | vertex # | edge # | construction time, s. |
|---------|------|---------|-------|-----------------------|
| B-2-C   | 32   | 23135   | 296759| 83.90                 |
| B-2-U   | 32   | 23213   | 345557| 85.07                 |
| B-3-C   | 32   | 23179   | 318470| 84.23                 |
| B-3-U   | 32   | 23222   | 390914| 83.70                 |
| G-2-C   | 32   | 120959  | 1447934| 521.75               |
| G-2-U   | 32   | 121013  | 1079792| 522.94               |
| G-3-C   | 32   | 119579  | 1553493| 532.40               |
| NY      | 16   | 53152   | 1016196| 61.76                |
| BAY     | 16   | 49385   | 812292| 45.65                 |
| COL     | 16   | 64082   | 953940| 72.09                 |
| FLA     | 16   | 168914  | 2702892| 177.65               |
| NE      | 16   | 241705  | 3835060| 240.36               |
TABLE V
AVERAGE OPTIMAL SOLUTION RUNTIMES AND THEIR STANDARD DEVIATIONS IN SECONDS

| dataset | MLS | t-ML$S$ | kPC-MLS | t-kPC-MLS |
|---------|-----|---------|---------|-----------|
| B-2-C   | 4.24 ± 2.76 | 3.69 ± 2.05 | 0.89 ± 0.45 | 0.83 ± 0.41 |
| B-2-U   | 870.30 ± 870.79 | 162.71 ± 112.96 | 220.73 ± 236.47 | 36.15 ± 26.13 |
| B-3-C   | 69.66 ± 121.81 | 63.25 ± 92.99 | 9.01 ± 18.08 | 9.11 ± 16.31 |
| B-3-U   | - | 8100.95 ± 6472.87 | 51801.95 ± 55113.33 | 2577.58 ± 2296.01 |
| G-2-C   | 47.58 ± 35.81 | 34.91 ± 17.79 | 7.74 ± 4.41 | 7.04 ± 3.61 |
| G-2-U   | 4488.91 ± 2911.09 | 1021.18 ± 495.92 | 1224.16 ± 833.04 | 212.26 ± 100.65 |
| G-3-C   | 3538.09 ± 3067.47 | 2460.66 ± 1782.28 | 521.99 ± 467.18 | 469.48 ± 367.45 |

TABLE VI
RATIOS OF SPEEDUPS PROVIDED BY INDIVIDUAL TECHNIQUES, COMPUTED AS $T_i / T$ , WHERE $T$ IS THE AVERAGE OPERATION TIME OF THE SLOWEST METHOD AND $T_i$ IS THE TIME OF THE CORRESPONDING TECHNIQUE

| dataset | MLS | t-ML$S$ | kPC-MLS | t-kPC-MLS |
|---------|-----|---------|---------|-----------|
| B-2-C   | 0.75 | 4.73 | 5.07 |
| B-2-U   | 1.00 | 3.94 | 24.07 |
| B-3-C   | 0.75 | 7.73 | 7.65 |
| B-3-U   | 1.00 | 0.16 | 3.14 |
| G-2-C   | 0.88 | 6.15 | 6.75 |
| G-2-U   | 0.88 | 3.66 | 21.15 |
| G-3-C   | 0.88 | 6.78 | 7.54 |

TABLE VII
NUMBERS OF ALL PRODUCED PARETO SOLUTIONS ON GRAPHS AND THEIR COVERS, IN AVERAGE AND WORST CASES

| dataset | original | KPC |
|---------|----------|-----|
| B-2-C   | average  | 2826580 | 219869 |
|         | worst    | 6759704 | 598819 |
| B-2-U   | average  | 603974909 | 4802929 |
|         | worst    | 18156754 | 14061220 |
| B-3-C   | average  | 17237523 | 1343315 |
|         | worst    | 108192240 | 9010826 |
| B-3-U   | average  | 72781967 | 55818896 |
|         | worst    | 1652695222 | 126105420 |
| G-2-C   | average  | 28748612 | 2383402 |
|         | worst    | 88572938 | 6935119 |
| G-2-U   | average  | 453135498 | 36182585 |
|         | worst    | 1259175424 | 101012692 |
| G-3-C   | average  | 297312123 | 24399389 |
|         | worst    | 695823074 | 57279143 |

These values measure not only Pareto sets of goal vertices, but of all the vertices processed during queries. The information being presented in the numbers of labels instead of consumed memory provides a more universal estimation that depends only on the problem instance while being unaltered by the implementation. As the table shows, kPC reduces memory requirements of the queries several times. As one can expect, the reduction in label number roughly corresponds to the ratio of vertices that remain in the kPC cover. Given standard variable sizes of C++ and a label consisting of the vertex ID, criteria vector and pointer to its parent, we can estimate two-criteria and three-criteria labels to be of sizes 16 B and 20 B respectively. 100,000 labels would then consume 1.52 MB and 1.91 MB respectively. Taking the worst solved case, which is classic MLS on B-3-U with 1.5 billion labels, the memory consumed would amount to almost 32 GB, and some instances would go beyond that. Additionally, we tried to estimate the memory consumed on the G-3-U dataset, which is not presented here due to being too hard to solve. For this, we’ve run multiple queries of $t$-discarding kPC-MLS on it. The hardest observed instance took over 38 hours to process and produced 1.8 billion labels, amounting to almost 35 GB. On the original graph, this would lead to approximately 22.86 billion labels consuming over 430 GB of memory.

It must be noted that $t$-discarding alters neither the number of processed vertices nor the actual Pareto sets. While it may require additional memory for operation, the volume of truncated vectors is incomparably smaller than that of the Pareto sets due to their decreased size and numbers.

E. Comparison With Literature

As it was stated earlier, most of the research on the multicriteria shortest path search problem is focused on one-to-one query methods. Thus, to be able to estimate a comparative efficiency of the proposed algorithm, we implemented it’s one-to-one version, which we call $t$-discarding kPC-IMA (Improved Martins Algorithm). The only addition necessary was use of the bounds precomputed through single criterion Dijkstra searches for upper and lower bound pruning as is described, for instance, in [12]. Additionally, we implemented the bounded version of classic MLS that is referred to as IMA in [12] and here and bSET in [11] and [13] to serve as a benchmark. We performed a comparison with multiple algorithms: BBDijkstra [11], LCDFP [13], and MDA [12].

All of these algorithms were tested on bicriteria graphs provided in the 9th DIMACS implementation challenge. Specifically, we used the same origin-destination pairs as in [11] and [13], 100 pairs for each graph. Graph sizes are presented in Table III. Since the comparison is performed between four different implementations tested on four different hardware systems, we equalize the conditions by considering only the reported speedup ratios in reference to IMA. This is done to mitigate the inconsistencies caused by the hardware limitations and the implementation details. The measurements do not include kPC construction time with $k = 16$ for $t$-kPC-IMA, since the cover can only be constructed once for a graph and used as many times as desired, making its construction more relevant to dataset preparation than to path queries. On the other hand, limit computation is individual for every query and is therefore included in the results. Unlike the rest of the algorithms, MDA in [12] was tested on different sets of dimensional space, as a benchmark. We performed a comparison with multiple algorithms: BBDijkstra [11], LCDFP [13], and MDA [12].

The results of the comparison are provided in Table VIII. As can be seen, MDA appears to be superior on the smaller...
TABLE VIII
SPEEDUP RATIOS OF ONE-TO-ONE SHORTEST PATH SEARCH METHODS ON DIMACS DATASETS

| Dataset | IMA  | LCDPP | BBDijkstra | MDA  | tKPC-MLS |
|---------|------|-------|------------|------|----------|
| NY      | 1.00 | 3.54  | 5.11       | 6.73 | 2.96     |
| BAY     | 1.00 | 2.88  | 4.91       | 5.47 | 5.51     |
| COL     | 1.00 | 3.71  | 3.59       | 5.82 | 8.67     |
| FLA     | 1.00 | 6.11  | 3.70       | -    | 12.95    |
| NE      | 1.00 | 4.60  | 2.09       | 5.36 | 8.13     |

graphs NY and BAY. This can be attributed to its effective approach to label generation and maintenance, since it foregoes the permanent/temporary label set approaches intrinsic to other algorithms. Nevertheless, t-discarding kPC-IMA scales far better than other algorithms, providing increasing speedup ratios on larger graphs. This result is unsurprising, since both kPC and t-discarding behave very well on larger problems. While kPC prunes the majority of the vertices and reduces memory consumption of IMA, t-discarding in bicriteria scenarios is virtually uninfluenced by the problem size, requiring only one comparison per weak domination check regardless of the number of permanent labels.

VI. CONCLUSION

The paper introduces a novel combination of techniques for accelerated one-to-many multicriteria shortest path search. During the experiments, the approach was compared to a classic multicriteria search algorithm on multiple datasets containing criteria combinations of varied sizes and correlation degrees. Providing a reliable basis for the conclusions on its performance, the tests show it gives at least 3 times speedup on simple problem instances. Moreover, the time gain increases with the complexity of the dataset and can even exceed 25 times while still returning optimal Pareto sets. Exclusion of irrelevant vertices from queries through kPC preprocessing decreases memory requirements on average by 13 times. Due to the core query algorithm being a modified version of MLS, the approach can additionally adopt existing heuristics and preprocessing techniques for further acceleration. To further evaluate the efficiency of the algorithm, we implemented its one-to-one version and compared it to existing one-to-one shortest path search methods. According to the results, the proposed algorithm scales far better than the alternatives due to its ability to minimize the problem through kPC and deal with the growing sizes of Pareto sets via t-discarding.

REFERENCES

[1] E. Q. V. Martins, “On a multicriteria shortest path problem,” Eur. J. Oper. Res., vol. 16, no. 2, pp. 236–245, May 1984.
[2] S. B. Stewart and C. C. White, “Multiobjective A*,” J. ACM, vol. 38, no. 4, pp. 775–814, Oct. 1991.
[3] M. Müller-Hannemann and K. Weihe, “On the cardinality of the Pareto set in bicriteria shortest path problems,” Ann. Oper. Res., vol. 147, no. 1, pp. 269–286, Oct. 2006.
[4] S. Funke, A. Nusser, and S. Storandt, “On k-path covers and their applications,” Proc. VLDB Endowment, vol. 7, no. 10, pp. 893–902, Jun. 2014.
[5] F.-J. Pulido, L. Mandon, and J.-L. Pérez-de-la-Cruz, “Dimensionality reduction in multiobjective shortest path search,” Comput. Oper. Res., vol. 64, pp. 60–70, Dec. 2015.
[6] E. W. Dijkstra, “A note on two problems in connexion with graphs,” Numerische Math., vol. 1, no. 1, pp. 269–271, Dec. 1959.
[7] P. Vincke, “Problèmes multicritères,” in Centre d’Études de Recherche Operationnelle. Cahiers, vol. 16, no. 4. Université Libre de Bruxelles, 1974.
[8] P. Hart, N. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” IEEE Trans. Syst. Sci. Cybern., vol. SSC-4, no. 2, pp. 100–107, Jul. 1968.
[9] L. Mandon and J.-L. Pérez-de-la Cruz, “A new approach to multiobjective A* search,” in Proc. UAI Conf. Artif. Intell., 2005, pp. 1–6.
[10] P. Sanders and L. Mandon, “Parallel label-setting multi-objective shortest path search,” in Proc. IEEE 27th Int. Symp. Parallel Distrib. Process., May 2013, pp. 215–224.
[11] A. Sedeño-noda and M. Colebrook, “A biobjective Dijkstra algorithm,” Eur. J. Oper. Res., vol. 276, no. 1, pp. 106–118, Jul. 2019.
[12] P. M. de las Casas, A. Sedeño-Noda, and R. Borndörfer, “An improved multiobjective shortest path algorithm,” Comput. Oper. Res., vol. 135, Nov. 2021, Art. no. 105424.
[13] Y. Kergosien, A. Giret, E. Néron, and G. Savanet, “An efficient label-correcting algorithm for the multiobjective shortest path problem,” INFORMS J. Comput., vol. 34, no. 1, pp. 76–92, Jan. 2022.
[14] J. Hrncir, P. Zílecky, Q. Song, and M. Jakob, “Practical multicriteria urban bicycle routing,” IEEE Trans. Intell. Transp. Syst., vol. 18, no. 3, pp. 493–504, Mar. 2017.
[15] P. Nunes, A. Moura, J. P. Santos, and A. Compeño, “A simulated annealing algorithm to solve the multi-objective bike routing problem,” in Proc. Int. Symp. Comput. Sci. Intell. Controls (ISCSC), Nov. 2021, pp. 39–45.
[16] Z. Masoumi, J. Van Genderen, and A. S. Iaraki, “An improved ant colony optimization-based algorithm for user-centric multi-objective path planning for ubiquitous environments,” Geocarto Int., vol. 36, no. 2, pp. 137–154, Jan. 2021.
[17] Y. Wang, X. Li, and R. Ruiz, “A fast algorithm for finding the bi-objective shortest path in complicated networks,” in Proc. IEEE 22nd Int. Conf. Comput. Supported Cooperat. Work Design (CSCWD), May 2018, pp. 104–109.
[18] B. Bresler, F. Kardoš, J. Katrenič, and G. Semanišin, “Minimum k-path vertex cover,” Discrete Appl. Math., vol. 159, no. 12, pp. 1189–1195, Jul. 2011.
[19] S. Funke and S. Storandt, “Personalized route planning in road networks,” in Proc. 23rd SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst., Nov. 2015, pp. 1–10.

Temirlan Kurbanov (Graduate Student Member, IEEE) received the master’s degree in artificial intelligence from Czech Technical University in Prague, Czech Republic, in 2020, where he is currently pursuing the Ph.D. degree with the Artificial Intelligence Center, Department of Computer Science, Faculty of Electrical Engineering. His research interests include advanced route planning and multi-agent systems.

Marek Čuchý received the master’s degree in artificial intelligence from Czech Technical University in Prague, Czech Republic, in 2016, where he is currently pursuing the Ph.D. degree with the Artificial Intelligence Center, Department of Computer Science, Faculty of Electrical Engineering. His research interests include travel planning for electric vehicles.

Jiří Vokřínek received the Ph.D. degree in artificial intelligence and bioinformatics from Czech Technical University in Prague, Czech Republic, in 2011. He is currently an Associate Professor and the Head of the Department of Computer Science, Faculty of Electrical Engineering, Czech Technical University in Prague. He is the author of more than 70 scientific publications. His research interests include planning, multiagent systems, and vehicle routing.