Multi-Objective Safe-Interval Path Planning with Dynamic Obstacles

Zhongqiang Ren[1], Sivakumar Rathinam[2], Maxim Likhachev[1] and Howie Choset[1]

***Abstract*—Path planning among dynamic obstacles is a fundamental problem in Robotics with numerous applications. In this work, we investigate a problem called Multi-Objective Path Planning with Dynamic Obstacles (MOPPwDO), which requires finding collision-free Pareto-optimal paths amid obstacles moving along known trajectories while simultaneously optimizing multiple conflicting objectives, such as arrival time, communication robustness and obstacle clearance. Most of the existing multi-objective A*-like planners consider no dynamic obstacles, and naively applying them to address MOPPwDO can lead to large computation times. On the other hand, efficient algorithms such as Safe-Interval Path Planing (SIPP) can handle dynamic obstacles but for a single objective. In this work, we develop an algorithm called MO-SIPP by leveraging both the notion of safe intervals from SIPP to efficiently represent the search space in the presence of dynamic obstacles, and search techniques from multi-objective A* algorithms. We show that MO-SIPP is guaranteed to find the entire Pareto-optimal front, and verify MO-SIPP with extensive numerical tests with two and three objectives. The results show that the MO-SIPP runs up to an order of magnitude faster than the conventional alternates.**

***Index Terms*—Motion and Path Planning**

I. Introduction

PLANNING collision-free trajectories for a robot amid dynamic obstacles is of fundamental importance in Robotics with numerous applications [24], [26]. In this work, we focus on a graph-based formulation of the problem in which the goal is to find a collision-free trajectory from a given start node to a destination node in the presence of dynamic obstacles whose motions are known. This problem has been addressed for a single objective (e.g. minimizing the arrival time of the robot at its destination) by several algorithms, such as the popular safe-interval path planning (SIPP) [13] method and its variants [4], [12], [27].

One can envision applications, ranging from urban search and rescue [5], [7], autonomous driving [1] to logistics [26], where multiple (conflicting) objectives such as arrival time, energy consumption and communication robustness, need to be optimized simultaneously. Aggregating these objectives into a single, weighted objective is difficult because the choice of weights is hard to obtain [19] and may not be known a-priori. This challenge motivated research in multi-objective path planning (MOPP) problems [8], [22], which generalizes the conventional (single-objective) path planning problem by associating each edge in the graph with a cost vector where each component of the vector corresponds to an objective to be minimized.

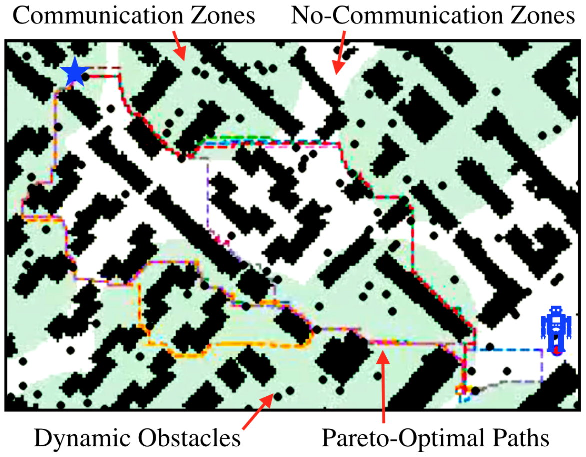


Fig. 1: A motivating example of MOPPwDO in the context of urban search and rescue. A robot needs to plan collision-free paths among dynamic obstacles in a city-like map while optimizing the arrival time and communication robustness along the path. For paths with smaller arrival times, the robot may have to traverse regions with no communication (the white area). Computing a Pareto-optimal set of solutions can reveal the underlying trade-off between different objectives and can potentially help decision makers to better understand the mission and make informed decisions.

In the presence of multiple conflicting objectives, in general, there is no single solution path that optimizes all the objectives at the same time. Therefore, the goal of MOPP is to find a Pareto-optimal set and its corresponding cost vectors referred to as the Pareto-optimal front. A solution is called Pareto-optimal if no objective can be improved without deteriorating at least one of the other objectives. Finding the Pareto-optimal front for MOPP even with two objectives can be computationally hard [6], [21]. There are several multi-objective A* (MOA*) planners [9], [22] in the literature to address the challenges in MOPP. However, we are not aware of existing MOA* planners that can find the Pareto-optimal front for a MOPP with Dynamic Obstacles (MOPPwDO). This paper aims to fill this gap.

One way to solve MOPPwDO is to (i) add a time dimension $\{0, 1, \ldots, T\}$ to the given graph G and construct a time-augmented graph $G^t = G \times \{0, 1, \ldots, T\}$ where dynamic obstacles are represented as blocked nodes in G^t, and (ii) run existing MOA* planners on G^t to find the Pareto-optimal

Zhongqiang Ren, Maxim Likhachev and Howie Choset are with Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213, USA. (email: zhongqir@andrew.cmu.edu; maxim@cs.cmu.edu; choset@andrew.cmu.edu).

Sivakumar Rathinam is with Texas A&M University, College Station, TX 77843-3123. (email: srathinam@tamu.edu).

front. The inclusion of the time dimension complicates the problem as it significantly increases the size of the graph to be searched. To bypass this challenge, we leverage the concept of *safe intervals* from SIPP which compresses time steps into contiguous safe and unsafe intervals to efficiently represent the search space. We then develop an algorithm called Multi-Objective Safe-Interval Path Planning (MO-SIPP) by leveraging the notion of *dominance* from the multi-objective optimization literature [2] and search techniques from MOA* algorithms [9], [23]. We show that MO-SIPP is guaranteed to find the entire Pareto-optimal front.

Motivated by urban search and rescue [5], [7], we generate test instances using city-like maps (Fig. 1) from an online data set and evaluate MO-SIPP with extensive tests where arrival time, communication robustness and obstacle clearance are considered. Our numerical results show that MO-SIPP runs up to an order of magnitude faster than the baseline.[1] To facilitate practitioners' usage and further development, we have made our MO-SIPP implementation available online[2].

The rest of the article discusses the related work in Sec. II, formulates the problem in Sec. III, and reviews SIPP in Sec. IV. We then present MO-SIPP in Sec. V, and analyze its properties in Sec. VI. Numerical results are then presented in Sec. VII, with conclusion and future work in Sec. VIII.

II. Related Work

A. Multi-Objective Path Planning

Existing approaches for multi-objective path planning (MOPP) problems compute an exact or approximated set of Pareto-optimal paths for the robot between its start and goal locations with respect to multiple objectives. One common approach to solve a MOPP is to weight the multiple objectives and transform it to a single-objective problem [2], [3]. The transformed problem can then be solved using any single-objective algorithm. This approach, however, requires an in-depth knowledge of the application to design the weighting procedure; it may also requires one to repeatedly solve the transformed single-objective problem for different sets of weights in order to capture the Pareto-optimal front which is quite challenging [10].

Additionally, MOPP has been solved directly via graph search techniques [9], [22], [23] and evolutionary algorithms [25], to name a few, where a Pareto-optimal set of solutions is computed exactly or approximately. These graph-based approaches provide guarantees about finding all Pareto-optimal solutions but may run slow for hard cases, especially when the number of Pareto-optimal solutions is large. The MO-SIPP in this work belongs to this category of methods that compute a set of Pareto-optimal solutions with quality guarantees.

Finally, MO-SIPP differs from our prior work MOPBD* [18]. Although both algorithms consider a dynamic environment, MOPBD* considers the scenario where graph edge costs can change while MO-SIPP focuses on avoiding dynamic obstacles.

[1]As discussed before, adding the time dimension to the given graph and using MOA* on the augmented graph.

[2]https://github.com/wonderren/public_mosipp

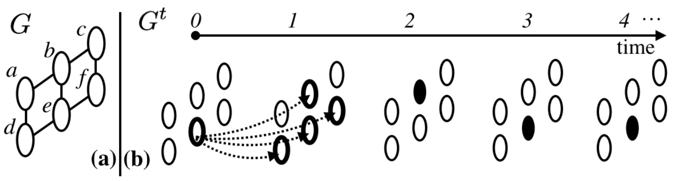


Fig. 2: An illustration of G, G^t and dynamic obstacles. G has 6 nodes as shown in (a). The time-augmented graph G^t is visualized with time steps between 0 and 4. A dynamic obstacle enters the environment at node b at time 2, moves to node e at time 3 and stays there afterwards.

B. Safe-Interval Path Planning

Safe-interval path planning (SIPP) [14] was originally developed to compute a collision-free trajectory from a start to a goal location while minimizing the arrival time, in an environment with dynamic obstacles moving along known trajectories. SIPP introduces the notion of safe intervals to efficiently represent the search space and identifies that by arriving at each safe interval at the earliest possible time, the computed solution is optimal. SIPP has been extended in several variants in the literature, such as sub-optimal SIPP [13], anytime SIPP [12], generalized SIPP [4], any-angle SIPP [27], etc. However, all these algorithms optimize a single objective.

III. Problem Formulation

Let $G = (V, E)$ denote a graph with vertex set V representing the possible locations of the robot in the workspace, and edge set E representing the transition between locations. Let $G^t = (V^t, E^t) = G \times \{0, 1, \ldots, T\}$ denote a *time-augmented graph* corresponding to G, where each vertex $v \in V^t$ is defined as $v = (u, t), u \in V, t \in \{0, 1, \ldots, T\}$ and T is the time horizon, which is typically a large positive integer. Edges in G^t are represented as $E^t = V^t \times V^t$ where vertices $(u_1, t_1), (u_2, t_2)$ are connected in G^t if $(u_1, u_2) \in E$ and $t_2 = t_1 + 1$.[3] In addition, $(u, t), (u, t+1), u \in V$ is connected in G^t to represent the wait in place action of the robot.

A dynamic obstacle along known trajectory is represented as a set of *blocked nodes* in G^t. An illustration of G, G^t and dynamic obstacles is shown in Fig. 2. Each edge $e \in E$ is associated with a non-negative cost vector $\vec{c}(e) \in (\mathbb{R}^+)^M$ with M being a positive integer and $\mathbb{R}^+$ being the set of non-negative real numbers. The wait in place action takes a constant non-negative cost vector $\vec{c}_{wait}$ at any nodes in G.

Let $\pi(v_0, v_\ell)$ denote a path connecting a pair of vertices $v_0, v_\ell \in G$ via a sequence of vertices $(v_0, v_1, \ldots, v_\ell)$ in G, where the subscript t of $v_t \in \pi(v_0, v_\ell)$ indicates the time step and v_t, and v_{t+1} are connected by an edge $(v_t, v_{t+1}) \in E$. Additionally, path $\pi(v_0, v_\ell)$ is *collision-free* if for each $v_t \in \pi(v_0, v_\ell)$, the corresponding (v_t, t) is not a blocked node in G^t. Let $\vec{g}(\pi(v_0, v_\ell))$ denote the cost vector corresponding to the path $\pi(v_0, v_\ell)$, which is the sum of the cost vectors of all edges

[3]To represent an edge $(u_1, u_2) \in E$ whose transition time takes multiple time steps, note that the edge can be "broke up" into a sequence of nodes $\{u_1, u_{k_1}, u_{k_2}, \ldots, u_{k_\ell}, u_2\}$ so that the transition between any two subsequent nodes takes unit time, and those intermediate nodes $\{u_{k_1}, u_{k_2}, \ldots, u_{k_\ell}\}$ can be added to the graph G.

present in the path: $\vec{g}(\pi(v_0, v_\ell)) = \Sigma_{t=0,1,\dots,\ell-1}\vec{c}(v_t, v_{t+1})$. To compare any two paths, we compare the cost vectors associated with them using the dominance relation [2]:

Definition 1 (Dominance): Given two vectors a and b of length M, a dominates b (denoted as $a \succeq b$) if and only if $a(m) \leq b(m)$, $\forall m \in \{1, 2, \dots, M\}$, and $a(m) < b(m)$, $\exists m \in \{1, 2, \dots, M\}$.
If a does not dominate b, we say a is non-dominated by b. Any two paths $\pi_1(v_0, v_\ell), \pi_2(v_0, v_\ell)$ are non-dominated (with respect to each other) if the corresponding cost vectors are non-dominated by each other.

Let v_o and v_d denote the initial and destination nodes in G respectively. The set of all non-dominated paths between v_o and v_d is called the *Pareto-optimal* set. A maximal subset of the Pareto-optimal set, where any two paths in this subset do not have the same cost vector is called a *cost-unique* Pareto-optimal set. This paper considers the Multi-Objective Path Planning with Dynamic Obstacles (MOPPwDO) problem that aims to compute a cost-unique Pareto-optimal set.

IV. A Brief Review of SIPP

MOPPwDO reduces to the SIPP problem [14] when the number of objectives (M) is equal to 1 and the cost of any edge $e \in E$ is the time required to traverse e. A naive baseline approach to solve the SIPP problem is to apply A* to search over G^t. However, this A* approach is inefficient as the time dimension is searched in a step-by-step manner (i.e., the time step is increased by one unit after each expansion). To overcome this challenge, SIPP [14] compresses time steps into intervals. Let tuple $s = (v, [t_a, t_b])$ denote a search *state* at node $v \in G$, where t_a, t_b are the beginning and ending time steps of a safe (time) interval respectively. A *safe interval* is a maximal contiguous time range in which a node is not blocked by any dynamic obstacles. State $s = (v, [t_a, t_b])$ indicates that node v is not blocked by any dynamic obstacle and hence the name safe interval for $[t_a, t_b]$. Note that, any two safe intervals at the same node never intersect. Two states are the same, if both states share the same node, the beginning and ending time steps. Otherwise, two states are different. For example, two states with the same node but different safe intervals are different states.

SIPP conducts A*-like heuristic search. For each state s, let $g(s)$ represent the earliest arrival time at state s at any time of the search and let $h(s)$ denote the heuristic value of s, which underestimates the cost-to-go (i.e., travel time to v_d from s). In addition, the f-value of a state is $f(s) := g(s) + h(s)$. Let OPEN denote the open list containing candidate states to be expanded, where candidate states are prioritized by their f-values.

SIPP starts by inserting the initial state s_o into OPEN, where s_o is a tuple of v_o and the safe interval with a beginning time set to zero. In each search iteration, a state with the minimum f-value in OPEN is popped and expanded. To expand a state $s = (v, [t_a, t_b])$, SIPP considers all reachable successor states from s and finds the earliest possible arrival time at each of those states via "wait and move" actions (i.e., wait for a minimum amount of time to arrive at the successor state as early as possible). A key observation in SIPP is that, arriving at a state s at the earliest possible time can generate the maximum number of successors from state s.

We provide examples of states and state expansion. In Fig. 2, at node b, there are two possible states $(b, [0, 1])$ and $(b, [3, T])$ while at node e, there is only one possible state $(e, [0, 2])$. The agent's initial state is $(a, [0, T])$. To expand the initial state, one successor is generated at node d, which is the state $(d, [0, T])$ with the earliest arrival time 1, and two successors are generated at node b, which are $(b, [0, 1])$ and $(b, [3, T])$ with the earliest arrival time 1 and 3 respectively.

During the search, SIPP records the earliest arrival time found thus far as $g(s)$ at each state s. When a new path is found to reach state s (from v_o) with an earlier arrival time, $g(s)$ is updated. Here, a path from v_o to some state $s = (v, [t_a, t_b])$ means a path from v_o to v with an arrival time at v within the safe interval $[t_a, t_b]$. When a state at the destination node (i.e., the node contained in the state is v_d) is expanded, a path with the minimum arrival time is found and SIPP terminates.

V. MO-SIPP

A. Concepts and Notations

As in SIPP, let a search state $s = (v, [t_a, t_b])$ be a tuple of a node and a safe interval. In SIPP, at each state s, a g-value is maintained to keep track of the minimum-arrival-time partial solution path from s_o to s. In a multi-objective problem, however, there can be multiple partial solution paths with non-dominated cost vectors from v_o to s, and all of them need to be recorded at s in order to compute a set of cost-unique Pareto-optimal solutions. The algorithm also needs to discriminate between those non-dominated partial solutions that arrive at the same state s with different cost vectors and arrival times.

Based on this observation, let $l = (s, \vec{g}, t_r)$ denote a *label*[4] at state s, which identifies a partial solution path from v_o to s with an arrival time t_r and a cost vector $\vec{g}$. For presentation purposes, let $\vec{g}(l)$, $t_r(l)$ and $s(l)$ represent the cost vector, arrival time and state associated with label l respectively. Also, let $v(l)$, $t_a(l)$ and $t_b(l)$ denote the node (in G), beginning time and ending time of the safe interval of state $s(l)$ respectively. Let *frontier set* $\alpha(s)$ denote a set of labels at state s, each of which identifies a non-dominated partial solution path from v_o to s. Let parent(l) denote the parent label of l, which enables easy reconstruction of a path for any label after the search. In addition, let $\mathcal{S}$ denote a set of (solution) labels, each of which identifies a Pareto-optimal solution path from v_o to v_d.

Scalar values g, h, f used in SIPP are now replaced with the corresponding cost vectors $\vec{g}, \vec{h}, \vec{f}$ in MO-SIPP. Specifically, $\vec{g}$ is associated with labels and it describes the cost-to-come from v_o. Notation $\vec{h}$ stands for an admissible heuristic[5] and is defined over states. Intuitively, $\vec{h}(s)$ provides a component-wise underestimate of the cost vector of all non-dominated

[4]To identify a partial solution path, different names such as nodes [23], states [15], [18] and labels [11], [20], have been used in the multi-objective path planning literature. This work uses "labels" to identify partial solution paths.

[5]The heuristic is not required to be consistent in this work.

paths from state s to v_d. Finally, $\vec{f}$ is defined over labels and $\vec{f}(l) := \vec{g}(l) + \vec{h}(s(l))$, which underestimates the cost vector of all paths from v_o to v_d via label l. At any time of the search, let OPEN denote a list of labels, where labels are prioritized in lexicographic order based on their $\vec{f}$-vectors.

B. Algorithm Description

As shown in Algorithm 1, the pseudo-code can be roughly divided into three parts: initialization (lines 1-5), checking and filtering (lines 7-21) and expansion (lines 22-28).

To initialize, MO-SIPP begins by creating an initial label l_o and inserts it into OPEN. The solution set $\mathcal{S}$ and all frontier sets $\alpha(s)$ are initialized to empty sets. Then, label l_o is added to the frontier set of the initial state $\alpha(s_o)$.

At the beginning of each search iteration (line 6), a lexicographically minimum label l is popped from OPEN for checking and filtering before being expanded:

- (Solution check) If $\vec{f}(l)$ is dominated by the $\vec{f}$-vector of any solution that is already found (identified by a label in $\mathcal{S}$), l is discarded (lines 8-9).
- (Frontier check) If l is label-dominated (see Sec. V-C) by any other labels in the current frontier set of $s(l)$, then l cannot lead to a cost-unique Pareto-optimal solution, and is thus discarded (lines 10-11).

After those checks, if $v(l) = v_d$, a new solution is found, and l is then used to *filter* existing solutions by removing any label $l' \in \mathcal{S}$ if $\vec{f}(l) \leq \vec{f}(l')$ (lines 13-15). This ensures that the solution set $\mathcal{S}$ always contains labels that represent non-dominated solution paths. l is then added to $\mathcal{S}$ and the current iteration ends (lines 16-17). If $v(l) \neq v_d$, l is used to filter the frontier of state $s(l)$ so that the frontier set $\alpha(s(l))$ contains labels that represent non-dominated partial solution paths from v_o to $s(l)$ (lines 18-20). Then, l is added to $\alpha(s(l))$.

After all the checking and filtering, l is expanded in a similar way as SIPP does, with the only difference that MO-SIPP expands and generates new labels (rather than states as in SIPP). Specifically, to expand a label l, MO-SIPP considers all possible reachable states from state $s(l)$, and then for each reachable state s', the earliest arrival time t'_r and the cost vector $\vec{g}'$ for reaching s' from $s(l)$ are computed. A successor label $l' = (s', \vec{g}', t'_r)$ is then generated. Then, for each successor label l', the aforementioned solution check and frontier check are applied to l' (line 24) to ensure l' is not dominated. Finally, the parent of l' is set to be l, and l' is added to OPEN for future expansion. The entire search process terminates when OPEN depletes, and all cost-unique Pareto-optimal solution paths are found (Sec. VI).

C. Label comparison

The comparison step in MO-SIPP differs from the one in SIPP. In SIPP, when a new path from v_o to a state s is found, the g-value of this new path and the previously stored g-value at s is compared and the smaller value is kept. In MO-SIPP, multiple non-dominated paths, which are represented by labels, need to be tracked at state s. To properly compare two labels, a new type of dominance between labels is defined as follows.

Algorithm 1 Pseudocode for MO-SIPP

```
1: $l_o \leftarrow (s_o, \vec{0}, 0)$
2: add $l_o$ into OPEN
3: $\mathcal{S} \leftarrow \emptyset$
4: $\alpha(s) \leftarrow \emptyset, \forall s$
5: add $l_o$ into $\alpha(s_o)$
6: while OPEN not empty do                          ▷ Main search loop.
7:     $l \leftarrow$ pop from OPEN
8:     if $\vec{f}(l') \leq \vec{f}(l)$, $\exists l' \in \mathcal{S}$ then
9:         continue                                 ▷ Pruned by existing solutions.
10:    if $l' \succeq_l l$, $\exists l' \in \alpha(s(l))$ then
11:        continue                                 ▷ Pruned by label dominance.
12:    if $v(l) = v_d$ then
13:        for all $l' \in \mathcal{S}$ do           ▷ Filter existing solutions.
14:            if $\vec{f}(l) \leq \vec{f}(l')$ then
15:                Remove $l'$ from $\mathcal{S}$
16:        add $l$ into $\mathcal{S}$
17:        continue                                 ▷ Find a new solution.
18:    for all $l' \in \alpha(s(l))$ do              ▷ To filter other labels.
19:        if $l \succeq_l l'$ then
20:            Remove $l'$ from $\alpha(s(l))$
21:        add $l$ to $\alpha(s(l))$
22:    $L_{succ} \leftarrow$ GetSuccessors($l$)
23:    for all $l' \in L_{succ}$ do                   ▷ Label expansion.
24:        if $l'' \succeq_l l'$, $\exists l'' \in \alpha(s(l'))$ or $\vec{f}(l'') \leq \vec{f}(l')$, $\exists l'' \in \mathcal{S}$ then
25:            continue
26:        $f(l') \leftarrow g(l') + h(s(l'))$
27:        parent($l'$) $\leftarrow l$
28:        add $l'$ into OPEN
29: return $\mathcal{S}$
```

Definition 2 (Label-dominance): Given two labels $l = (s, \vec{g}, t_r)$ and $l' = (s', \vec{g}', t'_r)$ with $s = s'$ (i.e., nodes and safe intervals in both s and s' are the same), if the following two conditions both hold: (i) $t_r \leq t'_r$, (ii) $\vec{g} + (t'_r - t_r)\vec{c}_{wait} \leq \vec{g}'$ (in (ii), $\leq$ means component-wise no larger than), then we say l *label-dominates* l' with notation $l \succeq_l l'$. (Here, $\succeq_l$ can be intuitively interpreted as "better than".)

In this label-dominance relationship, if $l \succeq_l l'$, condition (i) guarantees that label l will have at least the same set of successor labels as l' (refer to Lemma 1 in Sec. VI). Condition (ii) ensures that, if there exists a path π' from v_o to v_d via l', then the portion of the path from l' to v_d can be cut-and-paste to l, and the resulting path π'', which connects v_o and v_d via l, has a cost that is component-wise no larger than the cost of π'. Therefore, l' can be discarded.

Remark. Similar comparison rules to the label-dominance in this work have been developed in GSIPP [4] for a single-objective case, where a single non-arrival-time objective is minimized. The label-dominance here can be regarded as a generalization of the rule in GSIPP [4] from single-objective to multi-objective.

D. Relationship to SIPP

For readers that are familiar with SIPP, this section explains how MO-SIPP is related to SIPP. When MO-SIPP is applied to solve the aforementioned SIPP problem, MO-SIPP works in the same way as SIPP in the following sense: **First**, the label-dominance rule becomes a comparison between the arrival times of any two labels, which is the same as in SIPP. **Second**, the frontier set at each state contains only a single label, whose g-value is the minimum arrival time at s. Maintaining a frontier set at each state is thus equivalent to maintaining a g-value at each state. **Third**, the solution set $\mathcal{S}$ will either be empty or contain a single optimal solution at any time during the search, and the solution cost is the minimum arrival time at v_d. **Fourth**, when the first solution label l is found, all other candidate labels in OPEN must have a cost that is no less than $f(l)$ and are thus filtered in lines 8-9, which leads to the termination of Algorithm 1. As a result, MO-SIPP solves the SIPP problem by finding a collision-free path with the minimum arrival time.

VI. Analysis

A. Pareto-Optimality

Lemma 1: Arriving at a state s at the earliest possible time can generate the maximum number of successors.

Proof 1: Note that the set of reachable states from a label depends only on the arrival time of the label and is not dependent on the cost vector of the label. Therefore, given two labels $l = (s, \vec{g}, t_r)$ and $l' = (s, \vec{g}', t'_r)$ at the same state $s = (v, [t_a, t_b])$ with $t_r \leq t'_r$, any reachable state from l' (within time interval $[t'_r, t_b]$) is also reachable from l (within time interval $[t_r, t_b]$), regardless of the cost vectors $\vec{g}$ or $\vec{g}'$. Hence proved.

Lemma 2: At any time of the search, if a label at state s is pruned, the path represented by the label cannot be part of a cost-unique Pareto-optimal path.

Proof 2: In MO-SIPP, there are four cases where a generated label l' is pruned: (i) l' is pruned in the solution check by comparing $\vec{f}$-vectors (lines 8-9, line 23); (ii) l' is pruned in the frontier check by the label-dominance (lines 10-11, line 23); (iii) l' is filtered by a new label l that enters $\mathcal{S}$ when comparing their $\vec{f}$-vectors (lines 13-15). (iv) l' is filtered by label-dominance when a new label l enters $\alpha(s(l'))$ (lines 18-20). For either of those four cases, l' cannot lead to a cost-unique Pareto-optimal solution.

Therefore, in MO-SIPP, label expansion always generates successors with the earliest arrival time which guarantees the maximum number of successors (Lemma 1). Those generated successor labels are pruned if they cannot lead to a cost-unique Pareto-optimal solution (Lemma 2). MO-SIPP terminates when OPEN is empty, which means all labels are expanded or pruned, which guarantees that all cost-unique Pareto-optimal paths are found. The corresponding cost vectors of these cost-unique Pareto-optimal paths form the entire Pareto-optimal front. This property can be summarized with the following theorem:

Theorem 1: MO-SIPP algorithm is able to compute all cost-unique Pareto-optimal paths connecting v_o and v_d.

B. Completeness

In the formulation of MOPPwDO, the maximum possible time step is bounded by a finite time horizon T. Therefore, there are a finite number of states to be searched and MO-SIPP terminates in finite time if the given problem instance is infeasible (i.e., there does not exist a collision-free path connecting v_o and v_d within the time horizon T). When there exists a feasible solution, since the number of search states and the number of paths between any pair of nodes are finite, MO-SIPP finds a solution in finite time.

VII. Numerical Results

A. Test Settings and Implementation

We select four maps of different sizes from a online data set[6] and generate a four-connected grid-like graph G from each of the maps. This data set also includes 25 test instances for each map, where each instance includes hundreds of start-goal pairs. We use the first start-goal pair as the v_o and v_d for the robot. We then use A* to plan shortest paths between other start-goal pairs while considering only the static obstacles, and the dynamics obstacles move back-and-forth between their respective starts and goals along these shortest paths. The dynamic obstacles are allowed to overlap with each other during their movement. The number of dynamic obstacles (denoted as #Obst) is a parameter that varies in different tests.

Motivated by urban search and rescue [5], [7], we consider up to three objectives: arrival time, communication robustness and obstacle clearance, each of which is represented as a component of the cost vector of edges in G. To test MO-SIPP, we approximate the continuous measure of communication robustness and obstacle clearance as follows. We randomly generate circles in each of the map (Fig. 3) to represent the possible communication ranges after an urban disaster, where the communication infrastructure may be partially damaged [5], [7]. Each move within the communication range incurs a unit cost, while each move outside the communication range incurs a cost of ten (to penalize). For obstacle clearance, we inflate all the static obstacles by a certain range, which varies in maps of different sizes (see Fig. 6). Each move outside the inflated zone incurs a unit cost, while each move within the inflated zone incurs a cost of ten (to penalize). The cost vector for the wait action $\vec{c}_{wait}$ is set to a vector of all ones (at any node in the graph).

We implement MO-SIPP and NAMOA* [9] in C++ without multi-threading or compiler optimization. Here, NAMOA* serves as a baseline and is used to search the time-augmented graph G^t and is hereafter referred to as NAMOA*-st (-st stands for space-time). The heuristic vectors for both algorithms are computed by running an exhaustive backwards Dijkstra search for each type of the cost over the graph G ignoring all dynamic obstacles.[7] The resulting heuristic vectors are guaranteed to be admissible. All experiments are carried out on a laptop with a CPU i7-11800H 2.30GHz and 16GB RAM.

[6]https://movingai.com/benchmarks/mapf/index.html

[7]Using Dijkstra search to obtain heuristic vectors are common for MOA*-like algorithms [17], [23] due to its small runtime in comparison with the MOA* search. The numerical results about the runtime in this work do not include the time to compute the heuristic vectors.

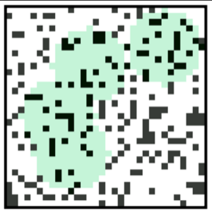
Maps and Test Settings	 Random 32x32 #Obst=100, R=6	Den312d 65x81 #Obst=200, R=10	Berlin 256x256 #Obst=300, R=32	Boston 256x256 #Obst=300, R=32
Average/Median/Maximum Runtime (in seconds)				
MO-SIPP (ours)	**0.037** / 0.036 / **0.108**	**0.07** / 0.05 / **0.25**	**1.54** / 0.92 / **6.00**	**1.56** / **0.81** / **8.06**
NAMOA*-st	0.043 / **0.030** / 0.184	0.10 / 0.05 / 0.59	3.18 / **0.84** / 14.66	6.74 / 1.84 / 37.79
Average/Median/Maximum Number of Expansions				
MO-SIPP (ours)	**347** / **357** / **1136**	**506** / **328** / **2139**	**7837** / **4938** / **30061**	**8708** / **4538** / **40085**
NAMOA*-st	1272 / 864 / 5843	2628 / 1091 / 16748	40700 / 19619 / 164579	102233 / 29484 / 451515
Average/Median/Maximum Number of Solutions				
	2.5 / 2.0 / 7.0	1.6 / 1.0 / 4.0	5.4 / 4.0 / 13.0	3.7 / 2.0 / 10.0
Average/Median/Maximum State-Vertex Ratios				
	3.6 / 3.6 / 4.5	4.2 / 4.2 / 6.8	6.6 / 6.2 / 13.1	5.8 / 5.6 / 11.2

Fig. 3: MO-SIPP and NAMOA*-st with two objectives: arrival time and communication robustness. Communication ranges are generated as circles at random locations (the green areas). The radius of the circle (R) and the number of dynamic obstacles (#Obst) are shown below each map. MO-SIPP (ours) runs faster than the baseline NAMOA*-st for up to four times. In larger maps, the advantage of MO-SIPP is more obvious. The state-vertex ratio is discussed in the text.

B. Experiment 1: Two Objectives

We begin our tests with two objectives (arrival time and communication robustness). As shown in Fig. 3, in terms of runtime, MO-SIPP runs up to four times faster than NAMOA*-st, which demonstrates the efficiency of MO-SIPP. Additionally, when the size of the map increases, the advantage of MO-SIPP over the baseline becomes more obvious. We also report the number of Pareto-optimal solutions and the number of expansions for reference.[8]Additionally, we introduce a metric "state-vertex ratio" to describe the impact of dynamic obstacles on the complexity of the problem, which is defined as the ratio of the number of states being explored by MO-SIPP over the number of vertices being explored by MO-SIPP. A state (or vertex) is explored by MO-SIPP if at least one label is generated at that state (or vertex).

C. Experiment 2: Varying Numbers of Obstacles

We then test MO-SIPP with the same two objectives as the previous section and change the number of obstacles. We select two maps, random 32x32 and Boston 256x256. In this experiment, we let the obstacles disappear after their arrival at the destinations to make the instances feasible. Otherwise, when #Obst is large, for some of the instances, there is no feasible trajectory for the robot to move from v_o to v_d.

As shown in Fig. 4, MO-SIPP runs faster than NAMOA*-st in all settings. In the Boston 256x256 (large) map, MO-SIPP runs up to an order of magnitude faster on average than NAMOA*-st for any considered #Obst. For the random 32x32 (small) map, the online data set has at most 409 start-goal pairs, and we test for up to 400 obstacles in this map. As shown in Fig. 4, the advantage of MO-SIPP over NAMOA*-st is less obvious in this smaller map. Additionally, after #Obst increases above a certain threshold (e.g. 200), the runtime of both algorithms start to decrease. To find the reason, we simulate some of the instances (Fig. 5). The large number of

[8]Each expansion in MO-SIPP is in general more time-consuming than NAMOA*-st since MO-SIPP needs to lookup safe-intervals at adjacent nodes to find reachable states, while the expansion in NAMOA*-st only needs to consider the next time step and adjacent nodes.

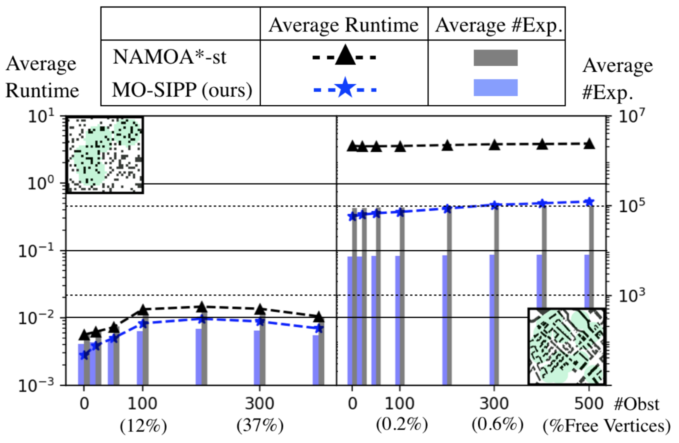


Fig. 4: MO-SIPP and NAMOA*-st in random 32x32 (left) and Boston 256x256 (right) with varying #Obst. The horizontal axis shows #Obst, and the percentage of free vertices in the graph that are occupied by the dynamic obstacles. MO-SIPP runs faster than NAMOA*-st on average in all settings, and is up to an order of magnitude faster than NAMOA*-st in the larger map (right).

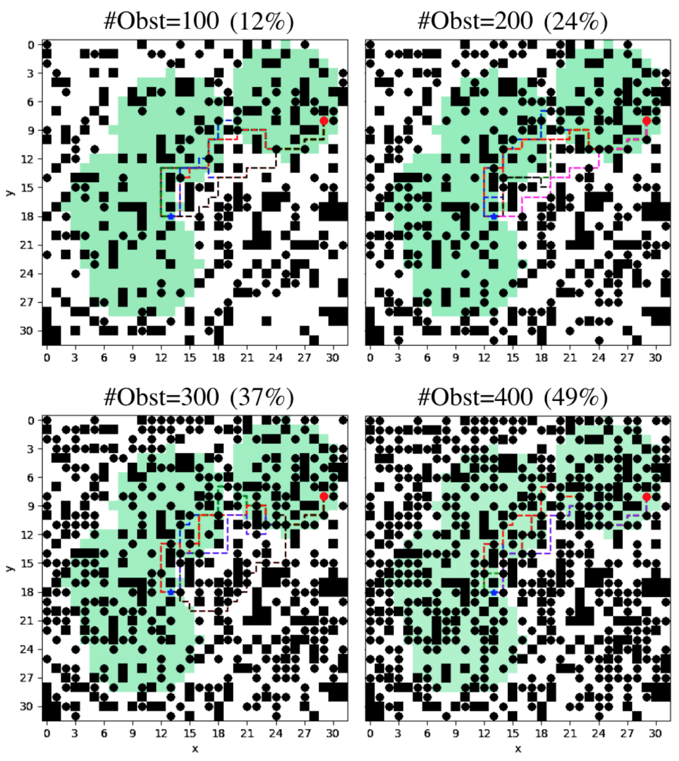


Fig. 5: Visualization of an instance with different #Obst (black dots), and the percentage of free vertices occupied by the dynamic obstacles is also shown for each map. As a complement to Fig. 4, when #Obst increases above a certain threshold, the large number of obstacles reduces the number of successors to be generated by both planners during the search and thus reduces the runtime.

obstacles reduces the number of successors to be generated by both planners during the search, and thus reduces the runtime, which is also verified by the average number of expansions as shown in the bar plots in Fig. 4.

D. *Experiment 3: Three Objectives*

Finally, we test MO-SIPP with three objectives. As shown in Fig. 6, MO-SIPP runs up to five times faster than NAMOA*-st. Additionally, by comparing the number of Pareto-optimal solutions, we find that the Berlin and Boston maps with three objectives are quite challenging as there are many Pareto-optimal solutions to be found. The difficulty of those instances can also be observed from the number of expansions required by both planners. In our tests, we set a runtime limit of ten minutes and NAMOA*-st times out for 6 and 8 instances in the Berlin and Boston maps respectively, which are removed from the data in Fig. 6.

With all these experiments, we can summarize our observations as follows. **First**, MO-SIPP (ours) runs up to an order of magnitude faster than NAMOA*-st (baseline). **Second**, the advantage of MO-SIPP becomes more obvious when the size of the map is large. **Third**, increasing the number of dynamic obstacles in general slows down MO-SIPP. But after a certain threshold, more obstacles makes MO-SIPP run faster. **Fourth**, a larger number of objectives leads to more Pareto-optimal solutions, which slows down MO-SIPP.

VIII. Conclusion and Future Work

This article considers the problem of Multi-Objective Path Planning with Dynamic Obstacles (MOPPwDO). We develop an algorithm called MO-SIPP by leveraging both the notion of safe intervals from SIPP, the dominance principle from multi-objective optimization literature and search techniques from multi-objective A* algorithms. We show that MO-SIPP finds the entire Pareto-optimal front and verify the performance of MO-SIPP with extensive tests. Our numerical results show that MO-SIPP runs up to an order of magnitude faster than the baseline and is particularly advantageous in large maps.

There are several possible directions for future work. MO-SIPP does not consider obstacle clearance with respect to dynamic obstacles or communication range centered on moving agents. One can also extend MO-SIPP to address dynamic obstacles along unknown or uncertain trajectories. To further expedite MO-SIPP, one can consider leveraging other multi-objective planning techniques (e.g. [17]) to handle the case when there are many Pareto-optimal solutions. In addition, MO-SIPP can also be leveraged as a building block to solve multi-objective multi-agent planning problems [15], [16]. Finally, one can consider leveraging the recent notion of rule-books [1] from autonomous driving to smartly select a Pareto-optimal solution that respects the most social conventions for execution.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 2120219 and 2120529. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] Andrea Censi, Konstantin Slutsky, Tichakorn Wongpiromsarn, Dmitry Yershov, Scott Pendleton, James Fu, and Emilio Frazzoli. Liability, ethics, and culture-aware behavior specification using rulebooks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8536–8542. IEEE, 2019.

[2] Matthias Ehrgott. *Multicriteria optimization*, volume 491. Springer Science & Business Media, 2005.

[3] Michael TM Emmerich and André H Deutz. A tutorial on multiobjective optimization: fundamentals and evolutionary methods. *Natural computing*, 17(3):585–609, 2018.

[4] Juan P Gonzalez, Andrew Dornbush, and Maxim Likhachev. Using state dominance for path planning in dynamic environments with moving obstacles. In *2012 IEEE International Conference on Robotics and Automation*, pages 4009–4015. IEEE, 2012.

[5] Jason Gregory, Jonathan Fink, Ethan Stump, Jeffrey Twigg, John Rogers, David Baran, Nicholas Fung, and Stuart Young. Application of multi-robot systems to disaster-relief scenarios with limited communication. In *Field and Service Robotics*, pages 639–653. Springer, 2016.

[6] Pierre Hansen. Bicriterion path problems. In *Multiple criteria decision making theory and application*, pages 109–127. Springer, 1980.

[7] Yugang Liu and Goldie Nejat. Robotic urban search and rescue: A survey from the control perspective. *Journal of Intelligent & Robotic Systems*, 72(2):147–165, 2013.

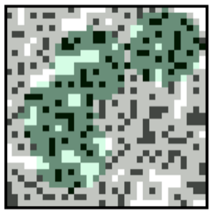
Maps and Test Settings	 Random 32x32 Clearance=1	Den312d 65x81 Clearance=2	Berlin 256x256 Clearance=3	Boston 256x256 Clearance=3
Average/Median/Maximum Runtime (in seconds)				
MO-SIPP (ours)	**0.104 / 0.076 / 0.380**	**0.40 / 0.16 / 3.58**	**19.9 / 6.64 / 128.5**	**18.0 / 1.79 / 102.3**
NAMOA*-st	0.173 / 0.097 / 0.647	0.81 / 0.29 / 9.13	79.3 / 14.2 / 531.8 (*)	122.2 / 8.23 / 547.4 (*)
Average/Median/Maximum Number of Expansions				
MO-SIPP (ours)	**948 / 739 / 3165**	**2554 / 1064 / 23058**	**52910 / 26263 / 238932**	**52830 / 10376 / 185072**
NAMOA*-st	3680 / 2553 / 12362	13162 / 6712 / 117470	276109 / 125247 / 1038554	510085 / 142499 / 2438196
Average/Median/Maximum Number of Solutions				
	8.6 / 7.0 / 34.0	7.4 / 5.0 / 30.0	69.5 / 32.0 / 256	70.8 / 18.0 / 389
Average/Median/Maximum State-Vertex Ratios				
	4.0 / 4.1 / 5.3	4.6 / 5.1 / 7.3	6.5 / 6.2 / 10.7	5.4 / 5.3 / 8.4

Fig. 6: MO-SIPP and NAMOA*-st with three objectives: arrival time, communication robustness and obstacle clearance (see Sec. VII-A for details). The first two objectives are the same as in Fig. 3. Obstacle clearance is implemented by inflating the static obstacles for a number of cells (as shown by the "Clearance" parameter below each map). When the robot moves through the inflated zone, a larger cost is incurred. MO-SIPP runs up to five times faster than NAMOA*-st. Within a runtime limit of 10 minutes, in the Boston and Berlin maps, NAMOA*-st times out for 6 and 8 instances respectively (marked as (*) in the figure). Those instances are removed from both the results of MO-SIPP and NAMOA*.

[8] Ronald Prescott Loui. Optimal paths in graphs with stochastic or multidimensional weights. *Communications of the ACM*, 26(9):670–676, 1983.

[9] Lawrence Mandow and José Luis Pérez De La Cruz. Multiobjective A* search with consistent heuristics. *Journal of the ACM (JACM)*, 57(5):1–25, 2008.

[10] R Timothy Marler and Jasbir S Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.

[11] Ernesto Queiros Vieira Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2):236–245, 1984.

[12] Venkatraman Narayanan, Mike Phillips, and Maxim Likhachev. Anytime safe interval path planning for dynamic environments. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4708–4715. IEEE, 2012.

[13] Mike Phillips and Maxim Likhachev. Planning in domains with cost function dependent actions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 25, 2011.

[14] Mike Phillips and Maxim Likhachev. Sipp: Safe interval path planning for dynamic environments. In *2011 IEEE International Conference on Robotics and Automation*, pages 5628–5635. IEEE, 2011.

[15] Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. Subdimensional expansion for multi-objective multi-agent path finding. *IEEE Robotics and Automation Letters*, 6(4):7153–7160, 2021.

[16] Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. A conflict-based search framework for multiobjective multiagent path finding. *IEEE Transactions on Automation Science and Engineering*, pages 1–13, 2022.

[17] Zhongqiang Ren, Sivakumar Rathinam, Maxim Likhachev, and Howie Choset. Enhanced Multi-Objective A* Using Balanced Binary Search Trees. In *Proceedings of the International Symposium on Combinatorial Search*, 2022.

[18] Zhongqiang Ren, Sivakumar Rathinam, Maxim Likhachev, and Howie Choset. Multi-objective path-based D* lite. *IEEE Robotics and Automation Letters*, 7(2):3318–3325, 2022.

[19] Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013.

[20] Peter Sanders and Lawrence Mandow. Parallel label-setting multi-objective shortest path search. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pages 215–224. IEEE, 2013.

[21] Paolo Serafini. Some considerations about computational complexity for multi objective combinatorial problems. In *Recent advances and historical development of vector optimization*, pages 222–232. Springer, 1987.

[22] Bradley S. Stewart and Chelsea C. White. Multiobjective A*. *J. ACM*, 38(4):775814, October 1991.

[23] Carlos Hernández Ulloa, William Yeoh, Jorge A Baier, Han Zhang, Luis Suazo, and Sven Koenig. A simple and fast bi-objective search algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 143–151, 2020.

[24] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[25] J. Weise, S. Mai, H. Zille, and S. Mostaghim. On the scalable multi-objective multi-agent pathfinding problem. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2020.

[26] Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 29(1):9–9, 2008.

[27] Konstantin Yakovlev and Anton Andreychuk. Towards time-optimal any-angle path planning with dynamic obstacles. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pages 405–414, 2021.