Spatial and Temporal Splitting Heuristics for Multi-Robot Motion Planning

Teng Guo Shuai D. Han Jingjin Yu

***Abstract*— In this work, we systematically examine the application of spatio-temporal splitting heuristics to the Multi-Robot Motion Planning (MRMP) problem in a graph-theoretic setting: a problem known to be NP-hard to optimally solve. Following the divide-and-conquer principle, we design multiple spatial and temporal splitting schemes that can be applied to any existing MRMP algorithm, including integer programming solvers and Enhanced Conflict Based Search, in an orthogonal manner. The combination of a good baseline MRMP algorithm with a proper splitting heuristic proves highly effective, allowing the resolution of problems $10+$ times than what is possible previously, as corroborated by extensive numerical evaluations. Notably, spatial partition of problem fusing with the temporal splitting heuristic and the enhanced conflict based search (ECBS) algorithm increases the scalability of ECBS on large and challenging DAO maps by 5–15 folds with negligible impact on solution optimality.**

I. Introduction

We study the labeled Multi-Robot Motion Planning problem (MRMP) under a graph-theoretic setting, also known as Multi-Agent Path Finding (MAPF). The basic objective of MRMP is to find a set of collision-free paths to route multiple robots from a start configuration to a goal configuration. In practice, solution optimality is also of key importance; yet optimally solving MRMP is generally NP-hard [1]–[3]. As one can readily imagine, given the ubiquity of the problem setting, effective algorithms find many important large-scale applications, e.g., warehouse automation [4], [5]. Other application scenarios include formation [6], [7], agriculture [8], object transportation [9], swarm robotics [10], to list a few. Due to the wide range of impactful applications, even though MRMP had been studied since the 1980s in the robotics domain [11]–[14], it remains an active research topic. Many effective algorithms, for example [15]–[17], have been proposed recently that balance fairly well between computational efficiency and solution optimality.

Nevertheless, there persists the practical need to continuously improve the efficiency and scalability of MRMP solutions, since a few percentage of computation time or path quality difference on path planning and motion execution could significantly affect the efficiency and throughput of these multi-robot systems. Such needs motivate us to carefully examine three key factors in MRMP that impact the performance of related algorithms: *the number of robots* n, *the size and complexity of the environment* S, and *the planning horizon* T. The overall complexity of a given problem can be measured as a function of these three factors, i.e., $f(n, S, T)$. A simple but reasonable approximation is the product $f(n, S, T) \propto nST$. The measure is rough as the factors are interrelated. For example, as n/S approaches its upper limit for a given S (i.e., the robot density hitting extremes), the complexity can grow exponentially in n. Most existing methods for optimally solving MRMP work with one or more of these factors. The most popular *decoupling* approach [12], [18]–[20] essentially treats each robot individually, handling interactions on an ad-hoc basis. Spatial and temporal domains have also been exploited, though rather sparsely. In [15], a rudimentary divide-and-conquer approach is applied to split the planning horizon to 2^m slices, decoupling a problem over the time domain. Combined spatial-temporal approach has also been exploited, e.g., in [21], where a space-time window is used to reduce the computational effort.

In this work, we made a first attempt to systematically exploit the application of divide-and-conquer over spatial and temporal domains, at the global level. Careful spatial and/or temporal division can be applied to most existing MRMP algorithms in an orthogonal manner, often bringing significant performance boosts. Specifically, our **main contributions** are: *(i)* We exploit multiple schemes for decoupling an MRMP instance over the temporal domain. Combined with effective solvers such as ILP [15] or ECBS [22], problems with many folds more robots can be readily solved, often with minimal impact on the solution optimality. *(ii)* We devise schemes for decoupling an MRMP instance over the spatial domain that is also compatible with temporal decoupling schemes. Spatial division heuristics allow much larger MRMP instances to be solved without significant impacts on the solution optimality.

Related Work. Whereas the feasibility question has been answered for MRMP [11], due to the hardness [1]–[3], many attempts have been made at optimally solving MRMP [12], [18]–[20], [23]–[26]. Among these, combinatorial-search based solvers have been demonstrated to be effective. One of the earliest work is Local Repair A* (LRA*) [23], which employs a basic form of decoupled search assisted with local repairs. The decoupling idea was also explored in [12]. Subsequently, a windowed approach [24] was shown to provide additional efficiency gain through restricting the spatio-temporal search domain. Specific heuristics were later developed, including independence detection [18], sub-dimensional expansion [19], conflict-based search [20], increasing-cost-tree search [25], to list a few. It is also possible to solve the problem through reducing MRMP to other problems, e.g., SAT [27], answer set programming [28], integer linear programming (ILP) [15]. Though these converted problems are also hard, they in fact facilitate the optimal resolution of the original MRMP problem due to the availability of specialized solver. As optimal solvers can be time con-

G. Teng, S. D. Han, and J. Yu are with the Department of Computer Science, Rutgers, the State University of New Jersey, Piscataway, NJ, USA. E-Mails: {teng.guo, shuai.han, jingjin.yu} @ rutgers.edu. This work is supported in part by NSF award IIS-1734419 and IIS-1845888.

suming to run, sub-optimal solutions to MRMP have also be extensively studied. Solvers like push and swap [29], push and rotate [30], windowed hierarchical cooperative A*, developed as part of [21], all return feasible solutions quickly. Balancing efficiency and optimality is one of the most attractive topics; some algorithms emphasize scalability without sacrificing much optimality, e.g., enhanced conflict based search (ECBS) [22], DDM [31].

Relating to our work, divide and conquer techniques had been applied to tackle optimally solving MRMP. Similar to approaches explored in this work, sub-goals and sub-plans are stitched together to construct a global plan for multi-agent planning in [32], which reduces the branching factor, leading to reductions in computation time. The *k-way-split* ILP [15] divides a problem into equal sized sub-problems by finding intermediate goals in the middle of individual paths. It effectively reduces the computation time of the ILP solver. The time domain split heuristic in this work builds on these earlier ideas and renders them more general.

Organization. In Section II, we formally define the multi-robot motion planning problem and provide preliminaries for ILP and ECBS. In Section III and Section IV we describe the time split heuristic and space heuristic respectively. In Section V, we provide evaluation results of these heuristics combined with MRMP solvers. We conclude in Section VI.

II. Preliminaries

The *Multi-Robot Motion Planning* problem (MRMP) is defined on an undirected graph $G = (V, E)$. We assume that G is a grid graph by default. That is, given integers w and h as the graph's *width* and *height*, the vertex set can be represented as $V \subseteq \{(i,j) \mid 1 \leq i \leq w, 1 \leq j \leq h, i \in \mathbb{Z}, j \in \mathbb{Z}\}$. The graph is 4-way connected, i.e., for a vertex $v = (i,j)$, the set of its neighboring vertices are defined as $N(v) = \{(i+1,j),(i-1,j),(i,j+1),(i,j-1)\} \bigcap V$. The problem involves n robots $r_1, \ldots, r_n$, where each robot r_i has a unique start state $s_i \in V$ and a unique goal state $g_i \in V$. We denote the joint start configuration as $X_S = \{s_1, \ldots, s_n\}$ and the goal configuration as $X_G = \{g_1, \ldots, g_n\}$. The objective of MRMP is to find a set of feasible path for all robots. Here, a *path* for robot r_i is defined as a sequence of $T+1$ vertices $P_i = (p_i^0, \ldots, p_i^T)$ that satisfies: *(i)* $p_i^0 = s_i$; *(ii)* $p_i^T = g_i$; *(iii)* $\forall 1 \leq t \leq T, p_i^{t-1} \in N(p_i^t)$. Apart from the feasibility of each individual path, for P to be collision-free, $\forall 1 \leq t \leq T, 1 \leq i < j \leq n$, P_i, P_j must satisfy *(i)* $p_i^t \neq p_j^t$ (no collisions on vertices); *(ii)* $(p_i^{t-1}, p_i^t) \neq (p_j^t, p_j^{t-1})$ (no head-to-head collisions on edges).

In this work, we consider two optimization objectives. The first objective is to minimize the *makespan*, which is the time for all robots to reach the goal vertices. Following our problem definition, the makespan objective is interpreted as $\min T$. The second objective is to minimize the *sum-of-costs*, a cumulative cost function that sums over all robots of the number of time steps required to reach the goals. For each robot, denoting t_i such that $\forall t_i \leq t \leq T, p_i^t = g_i$, the sum-of-costs objective is calculated as $\min \sum_{1 \leq i \leq n} t_i$. We point out that this later objective is also often a good proxy to the total travel distance objective.

The problems studied in this work are as follows.

Problem 1. Min-Makespan MRMP. *Given* (G, X_S, X_G), *find a conflict-free path set* P *that routes the robots from* X_S *to* X_G *and minimizes makespan* T.

Problem 2. Min-Sum-of-Costs MRMP. *Given* (G, X_S, X_G), *find a conflict-free path set* P *that routes the robots from* X_S *to* X_G *and minimizes sum of costs* $\sum_{1 \leq i \leq n} t_i$.

Instead of developing full algorithms, this work focuses on heuristics for dividing an MRMP instance into sub-problems. These are solved using existing algorithms, in parallel when possible. The two classes of heuristics divide the original problem in time or space domain. After the split, we make sure that the solution for one sub-problem does not affect computing solutions for the others, thus maintaining the completeness guarantee of the existing MRMP algorithms.

III. Splitting Over the Time Domain

Our time-division heuristic has its roots in a split heuristic from [15]. We first provide a brief introduction of that heuristic, and continue to describe our significant generalizations.

Given an MRMP instance, the original k-way (k as a power of 2) split heuristic [15] divides a problem into k equal sized sub-problems. Denoting the original start configuration as X_S and goal configuration as X_G, in the first iteration, the heuristic finds an *intermediate configuration* X_{IM}, and generates one sub-problem that routes robots from X_S to X_{IM}, and another sub-problem that routes robots from X_{IM} to X_G. Here, for each robot, its intermediate configuration in X_{IM} is a vertex that is roughly the same distance to the robot's start and goal vertices. Such a process is then recursively applied to the two sub-problems for another $k/2-1$ times each. After all sub-problems are created and solved individually, a solution to the original problem is found by concatenating solutions for the sub-problems. An example of such a time division is provided in Fig. 1.

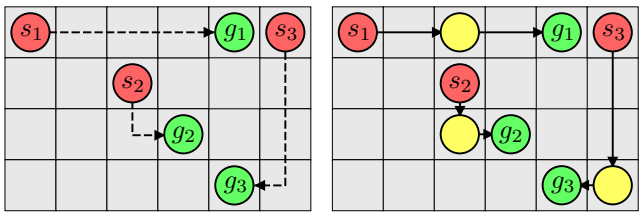

Fig. 1. An example of the time-split heuristic. (a) The original problem in a 6×4 grid with 3 robots. The start and goal configurations are visualized using red and green disks. The dashed lines show one set of possible solution paths. (b) With the time-split heuristic, the problem can be solved in two phases. The robots are first moved to an intermediate configuration (yellow disks), and then to the goals.

A. Arbitrary Splitting over the Time Domain

The original k-way split limits k to be powers of 2 and force each sub-problem to have roughly equal underestimated makespan (i.e. the minimum possible makespan when ignoring robot collisions). We remove these limitations, allowing

splitting the original problem into arbitrary number of sub-problems with different underestimated makespan.

Given an MRMP instance (G, X_S, X_G) and $k \in \mathbb{Z}$, our time-split heuristic (Algorithm 1) first computes a shortest path P_i for every robot i. Each time the instance is split, conflict-free intermediate states are located. For each intermediate state x_{IM}^{ij}, the intermediate state for robot i in the j-th sub-problem, we consider all robots in a descending order of the shortest path length. Then for each robot i, we find a series of candidate intermediate goal states X_{IM}^{i}.

The intermediate state x_{IM}^{ij} is picked from the vertices that are about $d_{ij} = j \cdot \text{dist}(x_S^i, x_G^i)/k$ from the start vertex and $|P_i| - d_{ij}$ from the goal vertex while avoiding conflicts, where $\text{dist}(u, v)$ is denoted as the shortest distance between two vertices u and v. If no conflict-free intermediate state is found, vertices whose distance form start vertex are $d_{ij} \pm 1$ are considered. When all intermediate goal states are decided, a polynomial time algorithm (i.e. [33], [34]) $CheckSolvable()$ can check if the resulting sub-problems are solvable. This procedure is repeated until a feasible intermediate goal state is found. The final feasible intermediate states denoted as $X_{IM} := \{x_{IM}^{ij}; 1 \leq j \leq k-1, 1 \leq i \leq n\}$ will be returned. In this way, the initial instance is split into k sub-problems, $P_1(G, X_S, X_{IM}^1), ..., P_k(G, X_{IM}^{k-1}, X_G)$. Any MRMP solvers may be applied to solve the resulting sub-problems. Since there is no interaction between the individual sub-problems, once we obtain the solution for each sub-instance the final path can be obtained by concatenating them together. The final makespan is obtained by adding all the makespan of each sub-problem together, which is $T = \sum_j T_j$. In practice, the simple heuristic dramatically improves algorithm performance without heavy negative impact on path optimality in terms of makespan; we observe a consistent speedup in computational experiments.

As a further generalization, our time-split also allows splitting the problem into instances with arbitrary ratio. For arbitrary ratio λ_j with $\sum_j \lambda_j = 1$, we just let $d_{ij} = (\sum_{\ell=1}^{\ell=j} \lambda_\ell) dist(x_S^i, x_G^i)$. For *min-makespan* MRMP, if we decide to split original problem into k sub-problems, we observe that even splits are generally better than uneven splits. Empirically, the computational time of MRMP solvers is largely determined by the time span. Since the computational time of time-split MRMP is decided by the maximum running time to solve each sub-problem, even splits lead to the smallest expected maximum time span of sub-problems, and consequently make the parallelization more efficient.

B. Special Considerations for Min-Sum-of-Cost Objective

Apart from requiring k to be a power of 2, the original k-way split heuristic performs poorly in solving *min-sum-of-costs* MRMP. Since the heuristic generates intermediate configurations by equally splitting the shortest paths, robots often cannot reach the goal configurations until the last sub-problem. Thus, a robot does not reach the goal vertex as fast as possible, even though it might be very close to the goal in the beginning. As an example, suppose that a 2-way split is carried out with each sub-problem having a time horizon

Algorithm 1: *k-time-split* for *min-makespan* MRMP

Input: Start and goal configurations X_S, X_G, graph G

Call A* to find an individual path P_i for each robot i;
Sort paths P_i according to path length in descent order;
for $j = 1$ *to* $k-1$ **do**
  while *true* **do**
    $H_{used} \leftarrow \emptyset$;
    for $i = 1$ *to* n **do**
      $S_1 \leftarrow \emptyset$, $S_2 \leftarrow \emptyset$, $d_{ij} \leftarrow j \cdot dist(x_S^i, x_G^i)/k$;
      $s_{min} = s_{max} = d_{ij}$, $g_{min} = g_{max} = |P_i| - d_{ij}$;
      while *true* **do**
        Find the vertices whose distance is in $[s_{min}, s_{max}]$ from x_S^i, add them to S_1;
        Find the vertices whose distance is in $[g_{min}, g_{max}]$ from x_G^i, add them to S_2;
        $V \leftarrow (S_1 \cap S_2) - H_{used}$;
        if $V \neq \emptyset$ **then**
          Choose random $x_{IM}^{ij} \in V$, add it to H_{used};
          break;
        else
          $s_{min} \leftarrow s_{min} - 1, s_{max} \leftarrow s_{max} + 1$;
          $g_{min} \leftarrow g_{min} - 1, g_{max} \leftarrow g_{max} + 1$;
    if $CheckSolvable(X_{IM}^j)$ **then** break;
return X_{IM};

of $T/2$. If a robot r_i does not move in the solution to the second sub-problem (i.e., $T/2 \leq t \leq T$), it contributes 0 to the total distance. However, if r_i moves even a single step in the solution to the first sub-problem (i.e., $0 \leq t \leq T/2$), then r_i will contribute at least $T/2$ to the total sum of costs. Thus, the final sum of costs obtained would be highly sub-optimal as pure an artifact of the heuristic.

We modify the *min-makespan* version of time-split, making it applicable to *min-sum-of-costs* MRMP. Take 2-way split as an example, suppose that the makespan lower bound of original MRMP is T, we still break it into two sub-problems with time horizon of $T/2$ each. Instead of choosing intermediate states at the middle of each individual path, for robot r_i the vertex whose distance is $d_i = \min(T/2, |P_i|)$ from start vertex while $|P_i| - d_i$ from goal vertex would be chosen, where $|P_i|$ is the path length of robot r_i found by A* ignoring conflicts with other robots. By setting $T/2$ as the threshold time-span, robots can reach their goal as fast as possible and the resulting sum of costs lower bound of the two sub-problems are additive.

Lemma III.1. *If the original problem is feasible, the time-split heuristic always generates feasible sub-problems.*

Proof. Assume original problem $P(G, X_S, X_G)$ is solvable, then there must be an optimal solution $\mathbf{\Pi} = (\Pi_0, \ldots, \Pi_T)$, with corresponding makespan T. Consider a configuration $\Pi_j = (\Pi_{1j}, \Pi_{2j} \ldots \Pi_{nj})$ where Π_{ij} is denoted as the path vertex of robot i at time step j, the sub-problems $P(G, X_S, \Pi_j)$ and $P(G, \Pi_j, X_G)$ are solvable. That is, a feasible problem indicates that feasible intermediate configurations always exist. Algorithm 1 iterates over all of the possible configurations in the second outer loop and terminates in finite steps when

a feasible configuration is found. □

Proposition III.1. *The time-split heuristic maintains the completeness of the existing* MRMP *algorithms.*

Remark. Theoretically, time-split is complete on any graph. In the worst case, finding a feasible intermediate configuration takes $O(|V|^n)$ time. However, in our experimental evaluation, when robot density is not extremely high, nearly every intermediate configuration found leads to solvable instances. Therefore, checking whether a sub-problem is solvable is unnecessary when robot density is not extremely high.

Remark. Time split heuristic is applicable in combination with any MRMP solvers; ILP and ECBS are chosen as representatives here. The performance of ILP solver is heavily affected by the ILP problem size, i.e. number of variables. Therefore, with smaller sub-problems to solve, time-split ILP runs faster than non-split ILP. As for ECBS, in the worst case, the sub-problems adopt the whole map and expand all states, which is the same as non-split ECBS. But in practice, the original problem is divided into sub-problems whose starts and goals are closer and it takes less time to find individual paths. Also, because the starts and goals are closer, usually when the instance is not very dense it takes less time to find solution for each sub-problem since there would be less conflicts in each sub-problem to resolve and thus the CT-tree needs to be searched is smaller. Besides, the heuristics allows us to take advantage of multiple cores and the resulting sub-problems can be solved in parallel.

IV. Splitting Over the Spatial Domain

As another natural route to the reduction of sub-problem sizes, a *space-split* heuristic is explored which splits the original problem over the spatial domain. To stitch together the sub-problems, *buffer zones* are introduced between divided regions of the environment (Fig. 2). Essentially, buffer zones are regions with small blocks that allow robots to migrate from one region of the larger environment to another region of the environment between sub-problems. After examining multiple choices, we settled with buffer zones containing multiple small rectangular blocks that belong to different regions in different sub-problems.

As an illustration, for the instance in Fig. 2, two buffer zones B_1 and B_2 are created. Each buffer zone contains two disconnected rectangular areas. The buffer zones separate the rest of the graph into two regions G_1, G_2 where $G = G_1 + B_1 + G_2 + B_2$. Here, we define operator "+" as $G(V, E) = G_1(V_1, E_1) + G_2(V_2, E_2)$ where $V = V_1 \cap V_2$ and $E = \bigcap_{v \in V}\{(v, u)|u \in N(v)\}$. Operator "−" is similarly defined.

Depending on their starts and goals, robots are classified into 4 groups: (i) $x_S \in G_1 + B_1$, $x_G \in G_1 + B_1$; (ii). $x_S \in G_2 + B_2$, $x_G \in G_2 + B_2$; (iii). $x_S \in G_1 + B_1$, $x_G \in G_2 + B_2$; (iv). $x_S \in G_2 + B_2$, $x_G \in G_1 + B_1$. Depending on the classification, intermediate goal states are selected for each robot. For example, a robot going from B_1 to B_2 may require it to go into G_1 first. The rules of choosing intermediate states are described in Algorithms 2 and 3. Algorithm 2 shows how

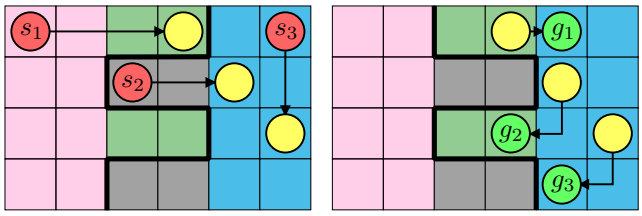

Fig. 2. Space split applied to the example in Fig. 1. Here, G_1, G_2, B_1, B_2 are colored in pink, blue, green, grey, respectively. The two sub-figures show the two sub-problems.

to classify the robots according to their starts and goals. If start and goal are in the same sub-graph, the intermediate state is chosen in that sub-graph. If the goal is in another sub-graph and the robot is in the buffer zone, it should not be sent to the buffer zone. If the goal is in another sub-graph but not in the buffer zone, the robot should be sent to the buffer zone in the first phase. Lines 7-8 ensure that the algorithm can always find a conflict-free intermediate goal even if the buffer zone is not large enough to hold all the robots.

Algorithm 2: Determine intermediate state

Input: Start x_S; goal x_G; subgraphs G_1, G_2; buffer zones B_1, B_2; set H_{used}

1 **if** x_S *and* x_G *are in the same subgraph* G_i **then**
2 $x_{IM} \leftarrow$ allocate(G_i, x_S, x_G);
3 **else if** x_G *is in another buffer zone* **then**
4 $x_{IM} \leftarrow$ allocate($G_i - B_i$, x_S, x_G);
5 **else** $x_{IM} \leftarrow$ allocate(B_i, x_S, x_G) ;
6 **if** $x_{IM} = null$ **then** $x_{IM} \leftarrow$ allocate(G_i, x_S, x_G) ;
7 return X_{IM};

Algorithm 3: Allocate intermediate state

Input: Start and goal x_S, x_G;Subgraph $SG(V, E)$;Set H_{used}

1 $x_{IM} \leftarrow null$, $minV \leftarrow +\infty$;
2 **for** $v \in V$ **do**
3 $f \leftarrow \lambda_1(\max(dist(v, x_S), T_1) + \max(dist(v, x_G), T_2)$
4 $+\lambda_2\rho(v) + dist(v, x_S) + dist(v, x_G)$;
5 **if** $f < minV$ *and* $v \notin H_{used}$ **then**
6 $x_{IM} \leftarrow v$;
7 $minV \leftarrow f$;
8 add x_{IM} to H_{used};
9 return x_{IM};

Algorithm 3 describes how intermediate states are chosen. We define f-value as a linear combination of the maximum makespan of two sub-problems, the local density $\rho(v)$, and the total distance a robot will travel. The local density $\rho(v)$ is defined as the number of occupied neighboring vertices of v. The vertex in a given sub-graph with minimum f value would be set as the intermediate goal. For 2-split we use $T_1 = T_2 = T/2$ as the threshold to make sure that makespan of each sub-problem not exceed $T/2$ so that we can fully take the advantages of multi-core computation.

After all intermediate states are determined, the original problem is dealt with in two phases. In the first phase, robots are sent from starts to intermediate states and we need to solve $P_{11}(X_S^{(11)}, X_{IM}^{(11)}, G_1 + B_1)$, $P_{12}(X_S^{(12)}, X_{IM}^{(12)}, G_2 + B_2)$. In the second phase, robots are sent from intermediate states to their goals and we need to solve $P_{21}(X_{IM}^{(21)}, X_G^{(21)}, G_1 + B_2)$,

$P_{22}(X_{IM}^{(22)}, X_G^{(22)}, G_2 + B_1)$. Again, ILP or any other general MRMP solvers can be readily applied to solve the resulting MRMP sub-problems in parallel.

In general cases, when applying *space split* to divide an original graph into $l \times m$ sub-graphs, we use a fixed buffer zone to complete the division. We find k intermediate states for each robot and the solution procedure breaks into k phases. In the i-th phase, the robots are sent from its $(i-1)$th intermediate state to i-th one. For each phase, there are $l \times m$ sub-instances need to solve and thus in total $l \times m \times k$ sub-instances need to be solved. The number of phases are determined by l, m and the longest paths robots need to travel. Usually, k is roughly $l + m$. A major advantage of the space split is that the reduced environment size simultaneously induces a reduction in sub-problems' makespans, allowing the sub-problems to be easily solved. As such, scalability is significantly boosted. The space-split can be combined with time-split, which brings further improvement to scalability. We denote the combination as *time-space-split*.

V. Experimental Evaluation

In this section, we evaluate how the proposed heuristics affect the *computation time* and the *optimality ratio*. The computation time is the time for an algorithm to generate a solution.The optimality ratio is measured as the solution cost over an underestimated cost, generated by moving all robots to the goals, ignoring collisions. For each test scenario, we push the number of robots up to the solvers' limit to test the effect of the heuristics on solvers' scalability. The start configuration, goal configuration and environment obstacles are uniformly randomly generated. Each result entry in this section is an average over 25 test cases.

We choose Integer Linear Programming (ILP) [15] and Enhanced Conflict-Based Search (ECBS) [22] as the low-level MRMP solvers. These two algorithms are state-of-the-art in terms of solving MRMP on graphs. For ECBS, we set its weight parameter $w = 1.5$ since it is a good balance between optimality and computational efficiency, as indicated in the original publication and from our observation. All experiments are executed on an Intel® Core™ i7-9700 CPU at 3.0GHz. Our heuristics are implemented in Java, while the MRMP solvers are in C++.

A. Evaluation of the Time Split Heuristic

First, we evaluated k-time-split heuristic on a 32×32 grid with 10% obstacles. Fig. 3 shows the makespan result using ILP. Notation ILP-kt stands for the combination of ILP and k-time-split. We observe that the scalability of ILP is significantly improved with minimal impact on solution optimality. For example, with 4-time-split, problems with 150 robots can be solved around 25 times faster, while the optimality ratio is well under 1.06.

In the second scenario, we demonstrate that by modifying the way intermediate goals are generated, the time split heuristic becomes much better for the *min-sum-of-costs* objective (see Fig. 4 and Fig. 5). Here "mk" stands for makespan and "tt" stands for sum of costs. A first observation

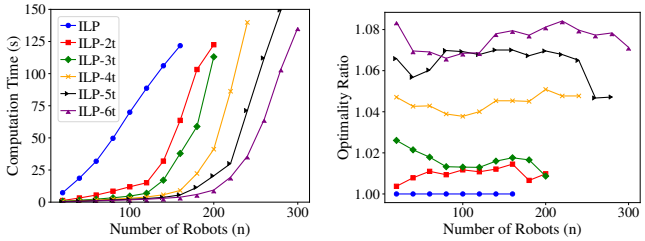

Fig. 3. Result of k-*time-split* on *min-makespan* MRMP.

is that, with the same number of sub-problems, the min-sum-of-costs version of time-split helps ILP to generate solutions much closer to optimal, as compared to the original k-way split heuristic. For example, the optimality ratio dropped from 1.6 to under 1.05 when using the revised heuristic. We also find that, similar to the previous evaluation, the time-split heuristic reduces computation time. There is a small difference on the heuristic's effect on computation time when $k = 2$, since using min-sum-of-costs version of time-split heuristic makes the first sub-problem relatively harder than the second one. This implies that for sum-of-costs time-split, the threshold time span at the middle is not necessarily the optimal choice, which hints further opportunities for improvements.

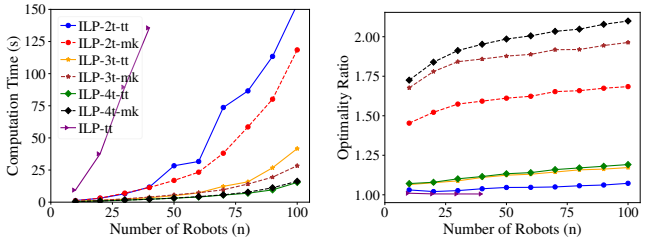

Fig. 4. Comparison of makespan and sum-of-costs time-split heuristics on *min-sum-of-costs* MRMP. The test graph is 32×32 grid with 10% obstacles.

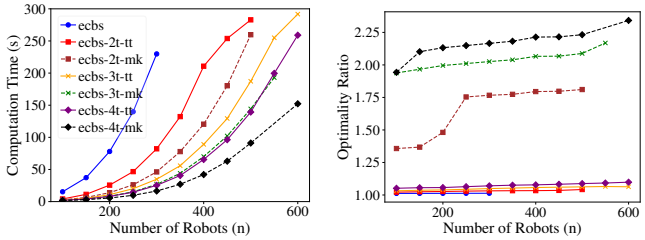

Fig. 5. Comparison of makespan and sum-of-costs time-split heuristics on *min-sum-of-costs* MRMP. The test graph is 128×128 grid with 10% obstacles.

Apart from the ILP solver, the proposed time-split heuristic also applies to other solvers such as ECBS. We evaluate the heuristic with ECBS on both grid graphs and the Dragon Age Origins (DAO) maps [35] (see Fig. 6), optimizing the makespan objective. Here, the test cases are imported from public MRMP benchmark instances that comes along with the maps, instead of randomly generated by ourselves. The results are shown in Fig. 7-9. Here, time-split allows problems with $10\times$ more robots to be solved in the same amount of time while the solution optimality is just above 1.06. This further confirms that the time-split heuristic significantly extends the scalability of existing MRMP solvers while sacrificing little optimality.

Fig. 6. The DAO maps: ost003d, den520d, brc202d.

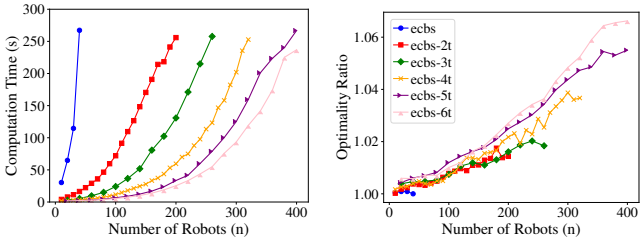

Fig. 7. Performance of time-split ECBS on ost003d.

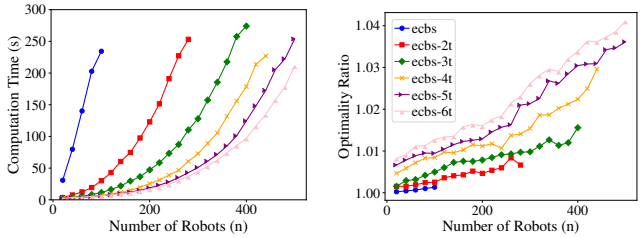

Fig. 8. Performance of time-split ECBS on den520d.

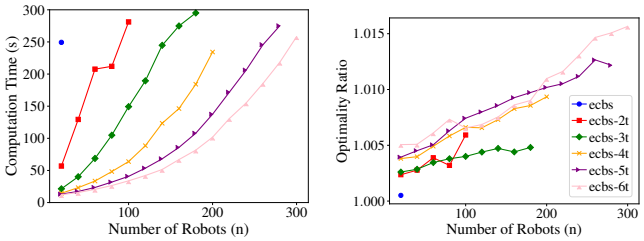

Fig. 9. Performance of time-split ECBS on brc202d.

B. Evaluation of the Space Split Heuristic

For space split, we mainly focus on the makespan objective and evaluated over many types of large grid graphs. In Fig. 10, we test space-split ILP on a 128×128 grid and compare it with time-split. With the same split level k, 2-space and 4-space splits run faster than 2-time and 4-time splits, respectively. 4-time split runs out memory when there are 40 robots while 4-space split can handle 60-70 robots without out of memory error. Due to the fact that 8-space split needs to solve much more sub-problems but the number of CPUs is limited, 8-space split scales better but in some cases not faster than 8-time split.

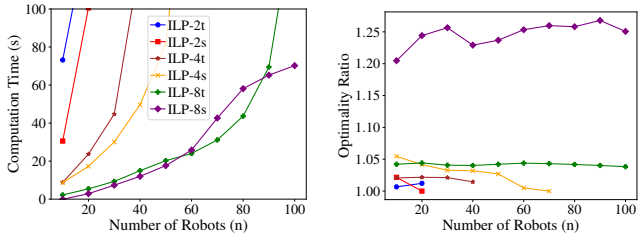

Fig. 10. *Time-split* vs *space-split* ILP on 128×128 grid.

C. Combined time-space split heuristics

As a last evaluation, we combine the two heuristics. Shown by the test result (Fig. 11) on a 128×128 grid with 5% obstacles, the combination of the two heuristics further extends existing algorithms' scalability. Method "xsyt" means that y-time-split is applied after a x-space split. While 16-time split has out of memory error when $n \geq 300$, time-space-split can handle instances with $n > 1000$. The ILP solver can now be used to solve problems in large environments with a large number of robots, while maintaining $1.x$ optimality.

We also attempted different size of buffer zones to see how it would affect the performance of time-space-split (Fig. 12). As it shows, using smaller buffer zones (e.g., 4×2) benefits both of computation time and optimality.

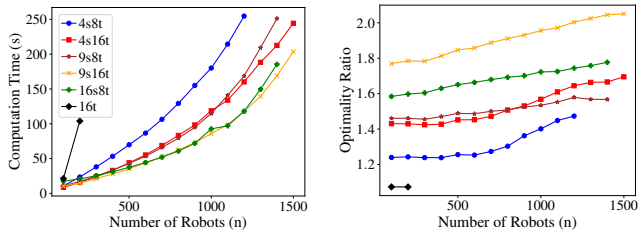

Fig. 11. Performance of time-space-split ILP on a 128×128 graph with 5% obstacles.

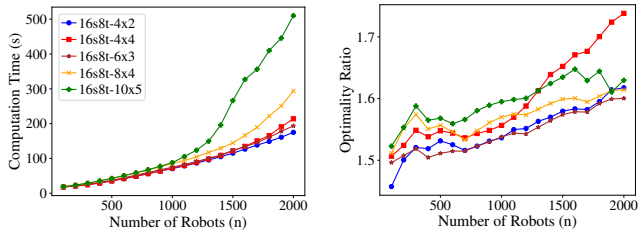

Fig. 12. Performance of space-split ILP using different size of buffer zones on 128×128 grid.

VI. Conclusion

In this work, temporal and spatial division heuristics are developed for improving the performance of MRMP solvers. These heuristics are shown to increase the computational speed while maintaining solution quality. These heuristics can be applied in combination with most MRMP algorithms. We note that (as proved) time split is complete and applicable to any graph; magnitudes of performance gains were consistently observed. On the other hand, space split is not complete. But space split enables ILP to provide solutions of good quality for some challenging MRMP problems that are otherwise not solvable previously.

In future work, we intend to make these heuristics more *data-driven*. That is, we will determine how to perform the temporal and spatial division based on the problem input dynamically. For example, the k in k-time-split can be selected based on $f(n, S, T)$ mentioned in the introduction. Furthermore, we plan to explore how to dynamically choose the buffer zone in the space split heuristic to improve its performance. Dynamic buffer zones are desirable when we work with irregular graphs and graphs with high obstacle density (e.g. $\geq 25\%$). Machine learning techniques may be also be applied.

REFERENCES

[1] J. Yu and S. M. LaValle, "Structure and intractability of optimal multi-robot path planning on graphs," in *Proceedings AAAI National Conference on Artificial Intelligence*, 2013, pp. 1444–1449.

[2] P. Surynek, "An optimization variant of multi-robot path planning is intractable," in *Proceedings AAAI National Conference on Artificial Intelligence*, 2010, pp. 1261–1263.

[3] J. Yu, "Intractability of optimal multi-robot path planning on planar graphs," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 33–40, 2016.

[4] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Magazine*, vol. 29, no. 1, pp. 9–19, 2008.

[5] E. Guizzo, "Three engineers, hundreds of robots, one warehouse," *IEEE spectrum*, vol. 45, no. 7, pp. 26–34, 2008.

[6] S. Poduri and G. S. Sukhatme, "Constrained coverage for mobile sensor networks," in *Proceedings IEEE International Conference on Robotics & Automation*, 2004.

[7] B. Smith, M. Egerstedt, and A. Howard, "Automatic generation of persistent formations for multi-agent networks under range constraints," *ACM/Springer Mobile Networks and Applications Journal*, vol. 14, no. 3, pp. 322–335, Jun. 2009.

[8] F. A. A. Cheein and R. Carelli, "Agricultural robotics: Unmanned robotic service units in agricultural tasks," *IEEE industrial electronics magazine*, vol. 7, no. 3, pp. 48–58, 2013.

[9] D. Rus, B. Donald, and J. Jennings, "Moving furniture with teams of autonomous robots," in *Proceedings IEEE/RSJ International Conference on Intelligent Robots & Systems*, 1995, pp. 235–242.

[10] J. A. Preiss, W. Hönig, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2017.

[11] D. Kornhauser, G. Miller, and P. Spirakis, "Coordinating pebble motion on graphs, the diameter of permutation groups, and applications," in *Proceedings IEEE Symposium on Foundations of Computer Science*, 1984, pp. 241–250.

[12] M. A. Erdmann and T. Lozano-Pérez, "On multiple moving objects," in *Proceedings IEEE International Conference on Robotics & Automation*, 1986, pp. 1419–1424.

[13] S. M. LaValle and S. A. Hutchinson, "Optimal motion planning for multiple robots having independent goals," *IEEE Transactions on Robotics & Automation*, vol. 14, no. 6, pp. 912–925, Dec. 1998.

[14] Y. Guo and L. E. Parker, "A distributed and optimal motion planning approach for multiple mobile robots," in *Proceedings IEEE International Conference on Robotics & Automation*, 2002, pp. 2612–2619.

[15] J. Yu and S. M. LaValle, "Optimal multi-robot path planning on graphs: Complete algorithms and effective heuristics," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1163–1177, 2016.

[16] E. Boyarski, A. Felner, R. Stern, G. Sharon, O. Betzalel, D. Tolpin, and E. Shimony, "Icbs: The improved conflict-based search algorithm for multi-agent pathfinding," in *Eighth Annual Symposium on Combinatorial Search*, 2015.

[17] L. Cohen, T. Uras, T. Kumar, H. Xu, N. Ayanian, and S. Koenig, "Improved bounded-suboptimal multi-agent path finding solvers," in *International Joint Conference on Artificial Intelligence*, 2016.

[18] T. S. Standley, "Finding optimal solutions to cooperative pathfinding problems," in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[19] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *2011 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2011, pp. 3260–3267.

[20] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[21] D. Silver, "Cooperative pathfinding." *AIIDE*, vol. 1, pp. 117–122, 2005.

[22] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *Seventh Annual Symposium on Combinatorial Search*, 2014.

[23] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[24] D. Silver, "Cooperative pathfinding," in *The 1st Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2005, pp. 23–28.

[25] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, "The increasing cost tree search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 195, pp. 470–495, 2013.

[26] J. Yu, "Constant factor time optimal multi-robot routing on high-dimensional grids," *2018 Robotics: Science and Systems*, 2018.

[27] P. Surynek, "Towards optimal cooperative path planning in hard setups through satisfiability solving," in *Pacific Rim International Conference on Artificial Intelligence*. Springer, 2012, pp. 564–576.

[28] E. Erdem, D. G. Kisa, U. Öztok, and P. Schueller, "A general formal framework for pathfinding problems with multiple agents." in *AAAI*, 2013.

[29] R. J. Luna and K. E. Bekris, "Push and swap: Fast cooperative path-finding with completeness guarantees," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[30] B. de Wilde, A. W. ter Mors, and C. Witteveen, "Push and rotate: cooperative multi-agent path planning," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 87–94.

[31] S. D. Han and J. Yu, "Ddm: Fast near-optimal multi-robot path planning using diversified-path and optimal sub-problem solution database heuristics," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1350–1357, 2020.

[32] E. Ephrati and J. S. Rosenschein, "Divide and conquer in multi-agent planning," in *AAAI*, vol. 1, no. 375, 1994, p. 80.

[33] D. M. Kornhauser, G. Miller, and P. Spirakis, "Coordinating pebble motion on graphs, the diameter of permutation groups, and applications," Master's thesis, M. I. T., Dept. of Electrical Engineering and Computer Science, 1984.

[34] M. M. Khorshid, R. C. Holte, and N. R. Sturtevant, "A polynomial-time algorithm for non-optimal multi-agent pathfinding," in *Fourth Annual Symposium on Combinatorial Search*, 2011.

[35] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, E. Boyarski, and R. Bartak, "Multi-agent pathfinding: Definitions, variants, and benchmarks," *Symposium on Combinatorial Search (SoCS)*, pp. 151–158, 2019.