Constant Factor Optimal Multi-Robot Path Planning in Well-Connected Environments

Jingjin Yu

Abstract—Fast algorithms for optimal multi-robot path planning are sought after in both research and real-world applications. Known methods, however, generally do not simultaneously guarantee good solution optimality and fast run time for difficult instances. In this work, we develop a low-polynomial running time algorithm, called SPLITGROUP, that solves the multi-robot path planning problem on grids and grid-like environments, and produces constant factor time- and distance-optimal solutions, in expectation. In particular, SPLITGROUP computes solutions with sub-linear makespan. SPLITGROUP is capable of handling cases when the density of robot is extremely high - in a graph-theoretic setting, the algorithm supports cases where all vertices of the underlying graph are occupied by robots. SPLITGROUP attains its desirable properties through a careful combination of divide-and-conquer technique and network flow based methods for routing the robots.

I. INTRODUCTION

Fast methods for multi-robot path planning have found many real-world applications including shipping container handling (Fig. 1(a)), order fulfillment (Fig. 1(b)), horticulture, among others, drastically improving the associated process efficiency. While commercial applications have been able to scale quite well, e.g., a single Amazon fulfillment center can operate over a thousand Kiva mobile robots, it remains unclear what level of optimality is achieved by the underlying scheduling algorithms in these applications. The same optimality-efficiency gap exists in the multi-robot research domain: known algorithms for multi-robot path planning do not simultaneously guarantee good solution optimality and fast running time. This is not entirely surprising as it is well known that optimal multi-robot path planning problems are generally NP-hard.

In this work, we narrow this optimality-efficiency gap in multi-robot path planning, focusing on a class of grid-like well-connected environments. Well-connected environments (to be formally defined) include the container shipping port scenario and the Amazon fulfillment center scenario. A key property of these environments is that sub-linear time-optimal solution is possible, which is not true for general environments. Using a careful combination of divide-and-conquer and network flow techniques, we show that expected constant factor time- and distance-optimal solutions can be computed in low-polynomial running time in such settings. We call the resulting algorithm SPLITGROUP. In other words, SPLITGROUP can efficiently compute $O(1)$ optimal solutions. The method readily generalizes to higher dimensions as well.

Jingjin Yu is with the Department of Computer Science, Rutgers University at New Brunswick. E-mails: jingjin.yu@cs.rutgers.edu.
computationally intractable under the graph-theoretic setting \cite{50, 51, 52, 53}, but the complexity has dropped from PSPACE-hard to NP-complete in many cases. Decoupling-based heuristics prove to be useful \cite{54, 55, 56}, allowing the effective minimization of certain accrued cost \cite{9, 57, 58, 59, 60, 61, 62, 63}. Beyond decoupling, other ideas have also been explored, including casting the problem as other known NP-hard problems \cite{64, 65, 66} for which high-performance solvers are available.

**Contributions.** The main contribution brought forth by this work is a low-polynomial time deterministic algorithm, called SPLITGROUP, for solving the optimal multi-robot path planning problem in grids and grid-like well-connected environments. Under the prescribed settings, SPLITGROUP computes a solution with sub-linear makespan. Moreover, the solution is only a constant multiple of the true optimal solution in terms of both makespan and total distance, in expectation. SPLITGROUP applies to settings with extreme robot density. To the best of our knowledge, SPLITGROUP is the first such algorithm that achieves the combination of desirable properties.

**Organization.** The rest of the paper is organized as follows. In Sec. II, the discrete multi-robot path planning problem is formally defined, followed by analysis on connectivity for achieving good solution optimality. This leads us to the choice of grid-like environments. We describe the details of the main algorithm, SPLITGROUP, in Sec. III. In Sec. IV, complexity and optimality properties of SPLITGROUP are established. In Sec. V, we show that SPLITGROUP generalizes to higher dimensions and (grid-like) well-connected environments including certain continuous ones.

II. PRELIMINARIES

In this section, we state the multi-robot path planning problem and two important associated optimality objectives, in a graph-theoretic setting. Next, we establish that working with arbitrary graphs may lead to rather sub-optimal solutions (i.e., super-linear with respect to the number of robots). This necessitates the restriction of the graphs if desirable optimality results are to be achieved.

A. Graph-Theoretic Optimal Multi-Robot Path Planning

Let $G = (V, E)$ be a simple, undirected, and connected graph. On this graph $G$, a set $R$ of labeled robots may move synchronously in a collision-free manner. At (integer) time steps starting from $t = 0$, each robot resides on a unique vertex of $G$, inducing a configuration $X$ of the robots. Effectively, $X$ is an injective map $X: R \rightarrow V$ specifying which robot occupies which vertex (see Fig. 2). From time step $t$ to time step $t+1$, a robot may move from its current vertex to an adjacent one under two (collision avoidance) conditions: (i) the new configuration at $t+1$ remains an injective map, i.e., each robot occupies a single vertex, and (ii) no two robots travel along the same edge in opposite directions.

A multi-robot path planning problem (MPP) is fully defined by a 3-tuple $(G, X_I, X_G)$ in which $G$ is a graph, and $X_I$ and $X_G$ are two configurations. In this work, we look at the extreme case of $|X_I| = |X_G| = |V|$. That is, all vertices of $G$ are occupied. We are interested in two optimal MPP formulations. In what follows, makespan is the time span covering the start to the end of a task and all edges of $G$ are assumed to have a length of 1 so that a robot traveling at unit speed can cross it in a single time step.

**Problem 1 (Minimum Makespan (TMPP)).** Given $G, X_I$, and $X_G$, compute a sequence of moves that takes $X_I$ to $X_G$ while minimizing the makespan.

**Problem 2 (Minimum Total Distance (DMPP)).** Given $G, X_I$, and $X_G$, compute a sequence of moves that takes $X_I$ to $X_G$ while minimizing the total distance traveled.

These two problems are known to be NP-hard and cannot always be solved simultaneously \cite{51, 52}. In this paper, we assume that $X_I$ and $X_G$ are randomly distributed.

B. Effects of Environment Connectivity

The well-known pebble motion problems, which are highly similar to MPP, may require $\Omega(|V|^3)$ individual moves to solve \cite{67}. Since each pebble (robot) may only move once per step, at most $|V|$ individual moves can happen in a time step. This implies that pebble motion problems, even with synchronous moves, can have an optimal makespan of $\Omega(|V|^2)$, which is super linear (i.e., $\omega(|V|)$). The same is true for TMPP under certain graph topologies. We first prove a simple but useful lemma for a class of graphs we call figure-8 graphs. In such a graph, there are $|V| = 7n + 6$ vertices for some integer $n \geq 0$. The graph is formed by three disjoint paths of lengths $n, 3n + 2$, and $3n + 2$, meeting at two common end vertices. Figure-8 graphs with $n = 1$ are illustrated in Fig. 3.

![Fig. 3. A three-step plan for exchanging the locations of robots 1 and 2 on a figure-8 graph with 7n + 6 vertices (n = 1 in this case).](image-url)
An interesting property of figure-8 graphs is that an arbitrary MPP instance on such a graph is feasible.

**Lemma 1.** An arbitrary MPP instance $(G, X_I, X_G)$ is feasible when $G$ is a figure-8 graph.

**Proof.** Using the three-step plan provided in Fig. 3, we may exchange the locations of robots 1 and 2 without collision. This three-step plan is scale invariant and applies to any $k$. With the three-step plan, the locations of any two adjacent robots (e.g., robots 4 and 5 in the top left figure of Fig. 3) can be exchanged. To do so, we may first rotate the two adjacent robots of interest to the locations of robots 1 and 2, do the exchange using the three-step plan, and then reverse the initial rotation. Let us denote such a sequence of moves as a 2-switch. Because the exchange of any two robots on the figure-8 graph can be decomposed into a sequence of 2-switches, such exchanges are always possible. As an example, the exchange of robots 4 and 9 can be carried out using a 2-switch sequence $\langle(3,4),(2,4),(1,4),(4,9),(1,9),(2,9),(3,9)\rangle$, of which each individual pair is an adjacent one after the previous 2-switch is completed. Because solving the MPP instance $(G, X_I, X_G)$ can be decomposed into a sequence of two-robot exchanges, arbitrary MPP instances are solvable on figure-8 graphs. □

The introduction of figure-8 graphs allows us to formally establish that sub-linear optimal solutions are not possible on an arbitrary connected graph.

**Theorem 2.** There exists an infinite family of TMPP instances with $\omega(|V|)$ minimum makespans.

**Proof.** We will establish the claim on the family of figure-8 graphs. By Lemma 1, there exists a sequence of moves that takes arbitrary configuration $X_I$ to arbitrary configuration $X_G$. For a figure-8 graph with $|V|$ vertices, there are $|V|!$ possible configurations. Starting from an arbitrary configuration $X_I$, let us build a tree of adjacent configurations with $X_I$ as the root and estimate its height $h_T$. In each move, only one of the three cycles on the figure-8 graph may be used to move the robots and each cycle may be moved in clockwise or counterclockwise direction. Therefore, the tree has a branching factor of at most 6. Assume the best case in which the tree is balanced and has no duplicated (configuration) nodes, we can bound $h_T$ as $6^{h_T+1} \geq |V|!$. That is, the tree must have at least $|V|!$ different configuration nodes derived from the root $X_I$, because all $|V|!$ configurations are reachable from $X_I$. With Stirling’s approximation applied to $|V|$, which yields

$$|V|! \geq \sqrt{2\pi|V|}\left(\frac{|V|}{e}\right)^{|V|},$$

we have

$$h_T = \Omega(|V| \log |V|).$$

This shows that solving some instances $(G, X_I, X_G)$ requires $\Omega(|V| \log |V|)$ time steps, establishing that TMPP on figure-8 graphs can require a minimum makespan of $\omega(|V|)$. □

**Corollary 3.** TMPP does not admit solutions with linear or sub-linear makespans on an arbitrary graph.

Corollary 3 suggests that seeking general algorithms for providing linear or sub-linear makespans that applies to all environments will be a fruitless attempt. With this in mind, we first focus our attention on a restricted but very practical class of discrete environments: grid graphs.

**III. Routing Robots on Rectangular Grids with a Sub-Linear Makespan**

We begin the analysis with rectangular grids. Assuming unit edge lengths, such a grid is fully specified by two integers $m_\ell$ and $m_s$, representing the number of vertices on the long and short sides of the rectangular grid, respectively. Without loss of generality, assume that $m_\ell \geq m_s$ (see Fig. 4 for a $8 \times 4$ grid). We further assume that $m_\ell \geq 3$ and $m_s \geq 2$ since a $2 \times 2$ grid does not admit non-trivial solutions. These assumptions on grid dimensions are implicitly assumed in this paper whenever “$m_\ell \times m_s$ grid” is used. The main result to be established in this section is the following.

**Theorem 4.** Let $(G, X_I, X_G)$ be an arbitrary TMPP instance in which $G$ is an $m_\ell \times m_s$ grid. The instance admits a solution with a makespan of $O(m_\ell)$.

Note that the $O(m_\ell)$ bound is sub-linear with respect to the number of robots, which is $\Omega(m_\ell m_s)$ and $\Omega(m_s^2)$ for square grids. We name the algorithm, to be constructed, as SPLITANDGROUP and first briefly sketch how the divide-and-conquer algorithm works at a high level. In this section we focus on the makespan optimality aspects of SPLITANDGROUP. We delay the the establishment of polynomial-time complexity and additional properties of the algorithm to Section IV.

To simplify the explanation, assume that $m_\ell = m_s = 2^k$ for some integer $k$. In the first iteration of SPLITANDGROUP, it splits the grid into two smaller rectangular grids, $G_1$ and $G_2$, of size $2^k \times 2^{k-1}$ each. Then, robots are moved so that at the end of the iteration, if a robot belongs to $G_1$ (resp., $G_2$) in $X_G$, it should be on some arbitrary vertex of $G_1$ (resp., $G_2$). This is the grouping operation. An example of a single SPLITANDGROUP iteration is shown in Fig. 4. We will show that such an iteration can be completed in $O(m_\ell) = O(2^k)$ time steps (makespan). In the second iteration, the same process is carried out on both $G_1$ and $G_2$ in parallel, which again requires $O(2^k)$ time steps. In the third iteration, we start with four $2^{k-1} \times 2^{k-1}$ grids and the iteration can be completed in $O(2^{k-1})$ time steps. After $2k$ iterations, the problem is solved with a desired makespan of

$$2O(2^k) + 2O(2^{k-1}) + \ldots + 2O(1) = O(2^k) = O(m_\ell).$$

We now proceed to describe the SPLITANDGROUP algorithm in detail. To achieve the stated $O(m_\ell)$ makespan, SPLITANDGROUP needs to enable as much concurrent robot movements as possible. This is rather challenging because of
FLIP can be decomposed into a constant number of time steps. Our worst case assumption that there are as many robots as the number of vertices. This is where the grid graph assumption becomes critical: it enables the concurrent “flipping” or “bubbling” of robots. Let $G = (V, E)$ be an $m \times n$ grid graph whose vertices are fully occupied by robots. Let $E' \subseteq E$ be a set of vertex disjoint edges of $G$. Suppose for each edge $e = (v_1, v_2) \in E$, we wish to exchange the two robots on $v_1$ and $v_2$ without collision. Let us call this operation FLIP($E'$).

**Lemma 5.** Let $G = (V, E)$ be an $m \times n$ grid. Let $E' \subseteq E$ be a set of vertex disjoint edges. Then the FLIP($E'$) operation can be completed in a constant number of time steps.

**Proof.** A $3 \times 2$ rectangular grid may be viewed as a figure-8 graph with $|V| = 6$ vertices. Applying Lemma 1 to the $3 \times 2$ grid tells us that the location of any two robots on such a graph can be exchanged without collision. Furthermore, all such exchanges can be pre-computed and performed in $O(1)$ (i.e., a constant number) of time steps.

To perform FLIP($E'$) on a $m \times n$ grid $G$, we partition the grid into multiple disjoint $3 \times 2$ blocks. Using up to 4 different such partitions, it is always possible to cover all edges of $G$. Therefore, the FLIP($E'$) operation can be broken down into parallel two-robot exchanges on the $3 \times 2$ blocks of these partitions. Because of the parallel nature of the two-robot exchanges, the overall FLIP($E'$) operation can be completed $O(1)$ time steps. As an example, Fig. 5 illustrates how a FLIP($E'$) operation can be carried out on a $7 \times 5$ grid. In the figure, the solid edges represent the edge set $E'$. After each partition starting from the top left one, two-robot exchanges can be performed which allow the removal of the edges covered by the partition, as shown in the subsequent picture. Note that, if either $m \times n$ or $n \times m$ is even, only two partitions are required.

Lemma 5 in a nutshell, allows the exchange of two adjacent robots to be performed in $O(1)$ time steps. Moreover, it allows such exchanges to happen in parallel on disjoint edges. With Lemma 5 to prove Theorem 4 we are left to show that on an $m \times n$ grid with $m \geq n$, after the split operation the grouping operation in the first SPLITANDGROUP iteration can be decomposed into $O(m)$ FLIP($\cdot$) operations. Because each FLIP($\cdot$) can be carried out in $O(1)$ time steps, the overall makespan cost of the grouping operation is then $O(m)$. To obtain the desired decomposition, we need to maximize parallelism along the split line used for the split operation. We achieve the desired parallelism by partitioning the grid into trees with limited overlap. Each such tree has a limited diameter and crosses the split line. The grouping operation will then be carried out on these trees. Before getting to the tree-partitioning step, we show that grouping robots on trees can be done efficiently. We start by showing that we can effectively “herd” a group of robots to the end of a path. Note that we do not require a robot in the group to go to a specific goal vertex; we do not distinguish robots within the group. In what follows, by non-path grid, we mean a grid that is not a path.

**Lemma 6.** Let $P$ be a path of length $\ell$ embedded in a non-path grid whose vertices are fully occupied by robots. An arbitrary group of up to $\lfloor \ell/2 \rfloor$ robots on $P$ can be relocated to one end of $P$ in $O(\ell)$ time steps. Furthermore, the relocation may be performed using FLIP($\cdot$) on $P$.

**Proof.** Because we are to do the relocation using parallel two-robot exchanges on disjoint edges based on the FLIP($\cdot$) operation, without loss of generality, we may assume that the path is straight and we are to move the robots to the right end of the path. An example illustrating the scenario is given in Fig. 6. For a robot in the group, let its initial location on the path be of distance $k$ from the right end. We inductively prove the claim that it takes $O(k)$ time steps from the beginning of all moves to “shift” such a robot to its desired goal location. We emphasize that the grouping operation and groups of robots are related but bear different meanings.
trivially holds for $k = 0$. Suppose it holds for $k - 1$ and we need to show that the claim extends to $k$. If $r_k$ does not belong to the group of robots to be moved, then there is nothing to do. Otherwise, there are two cases.

In the first case, robot $r_{k-1}$ does not belong to the group of robots to be moved. Then at $t = 0$, $r_k$ and $r_{k-1}$ may be exchanged in $O(1)$ time steps. Now $r_k$ is of distance $k - 1$ to the right and the inductive hypothesis applies to yield that the rest of the moves for $r_k$ can be completed in $O(k - 1)$ time steps. The total number of steps is then $O(k)$.

In the second case, robot $r_{k-1}$ also belongs to the group of robots to be moved. By the inductive hypothesis, $r_k$ can be moved to its desired goal in $O(k - 1)$ time steps. However, once $r_{k-1}$ is moved to the right, it will allow $r_k$ to follow it with a gap between them of at most 2. Once $r_{k-1}$ reaches its goal, $r_k$, whose goal is on the right of $r_{k-1}$, can reach its goal in $O(1)$ additional time steps. The total number of time steps from the beginning is again $O(k)$.

It is clear that all operations can be performed using $\text{FLIP}()$ on edges of $P$ when embedded in a grid.

Using the herding operation, the locations of two disjoint groups of robots with equal number of robots can also be exchanged efficiently.

**Lemma 7.** Let $P$ be a path of length $\ell$ embedded in a non-path grid whose vertices are fully occupied by robots. Let two groups of equal number of robots reside on two non-intersecting segments of $P$. Then positions of the two groups of robots may be exchanged in $O(\ell)$ time steps without net movements of other robots. The relocation may be performed using $\text{FLIP}()$ on $P$.

**Proof.** We may again assume that $P$ is straight. An implicit assumption is that each group of robots contains at most $\lfloor \ell/2 \rfloor$ robots. Fig. 7 illustrates an example in which two groups of 4 robots each need to switch locations on such a path.

![Fig. 7](image)

To do the grouping, we first apply a herding operation that moves one group of robots to one end of $P$. In Fig. 7 this is done to the group of lightly-shaded robots to move them to the right side (the second row of Fig. 7). Then, another herding operation is performed to move the other group to the other end of $P$ (the third row of Fig. 7). In the third and last step, two parallel “reversed” herding operations are carried out on two disjoint segments of $P$ to move them to their desired goal locations. This is best understood by viewing the process as applying the herding operation to the goal configuration. As an example, in Fig. 7 from the goal configuration (last row), we may readily apply two herding operations to move two groups of robots to the two ends of $P$ as shown in the third row of the figure. Because each herding operation takes $O(\ell)$ time steps, the overall operation takes $O(\ell)$ time steps as well. It is clear that in the end, a robot not in the two groups will not have any net movement on $P$ because the relative orders of these robots (unshaded ones in Fig. 7) never change.

Next, we generalize Lemma 7 to a tree embedded in a grid. On a tree graph $T$, we call a subgraph a *path branch* of $T$ if the subgraph is a path with no other attached branches. That is, all vertices of the subgraph have degree one or two in $T$.

**Theorem 8.** Let $T$ be a tree of diameter $d$ embedded in a non-path grid whose vertices are fully occupied by robots. Let $P$ be a length $\ell$ path branch of $T$. Then, a group of robots on $P$ can be exchanged with robots on $T$ outside $P$ in $O(d\ell)$ time steps without net movement of other robots. The relocation may be performed using $\text{FLIP}()$ on $T$.

**Proof.** We temporarily limit the tree $T$ such that, after picking a proper main path that contains $P$ and deleting this main path, there are only paths left. That is, we assume all vertices with degree three or four are on a single path containing $P$. An example of such a tree $T$ and the exchange problem is given in the top row of Fig. 8. In the figure, the main path is the long horizontal path and $P$ is the path on the left of the dotted split line. We call other paths off the main path side branches. Once this version is proved, the general version readily follows. The rest of the paper will only use the less general version of this lemma. For ease of reference, for the two groups of robots, we denote ones on $P$ as $g^1$ and the other group as $g^2$. In the example, $g^1$ has a light shade and $g^2$ has a darker shade.

![Fig. 8](image)
After the step, the robots involved in the first step are no longer relevant. In the example, this is to exchange the robots marked with small arrows in the first row of Fig. 8. After the relocation of these robots is completed, we remove their shades.

In the second step, the relevant robots in $g^2$ on the side branch are moved so that they are just off the main path. We also assign priorities to these robots based on their closeness to $P$ and break ties randomly. For a robot labeled $i$ in a group $g^1$, we denote the robot as $r^1_i$. For our example, this current step yields the third row of Fig. 8 with the priorities marked. Since the moves are done in parallel and each branch is of length at most $d$, only $O(d)$ time steps are needed.

In the third step, robots from $g^2$ will move out of the side branches in the order given, one immediately after the other (when possible). For the example (third row of Fig. 8), $r^2_1$ will move first, $r^2_2$ will follow. Then $r^3_1$, followed by $r^3_2$. Using the same inductive argument from the proof of Lemma 6, we observe that all robots from $g^2$ on the side branch can be moved off the side branches (and reach their goals on the main path) in $O(d)$ time. As the relevant robots from $g^1$ also move across the split line, they will fill in side branches in opposite order to when the robots from $g_2$ are moved out of the branches. In the example, this means that the branch where $r^3_2$ and $r^2_2$ were on will be populated with robots from $g^1$ first, followed by the branch where $r^2_2$ was, and finally the branch where $r^2_2$ was. This ensures that at the end of this step, any robot not in $g^1$ and $g^2$ will have no net movement. The number of time steps for this is again $O(d)$.

In the last step, we simply reverse the second step, which takes another $O(d)$ time steps. Putting everything together, $O(d)$ time steps are sufficient for completing the desired task.

Putting all steps together, only $O(d)$ time steps are required to complete the desired exchange. To see that the same conclusion holds for more general trees with side branches that are not simple paths, we simply need to do the second step and third step more carefully. But, because we are only moving at most $O(d)$ robots, using an amortization argument, it is straightforward to see that the $O(d)$ bound does not change.

We note that many of the operations used to prove Lemma 6, Lemma 7, and Theorem 8 can be combined without changing the outcome. However, doing so will make the proofs less modular. Given the focus of the current paper which is to construct a polynomial time algorithm with constant factor optimality guarantee, we opt for clarity instead of pursuing a smaller asymptotic constant. We proceed to prove Theorem 4.

**Proof of Theorem 4** We prove the theorem by showing in detail how to carry out a single iteration of SPLITANDGROUP. In a split step, we always split along the longer side of the current grid. Since $m_ℓ \geq m_s$, the $m_ℓ \times m_s$ grid is split into two grids of dimensions $\lceil m_ℓ/2 \rceil \times m_s$ and $\lfloor m_ℓ/2 \rfloor \times m_s$, respectively. For convenience, we denote the two split grids as $G_1$ and $G_2$, respectively. Recall that in the grouping operation, we want to exchange robots so that a robot with goal in $G_1$ (resp., $G_2$) resides in $G_1$ (resp., $G_2$) at the end of the operation.

To do this efficiently, we need to maximize the parallelism. This is achieved through the computation of a set of $m_s$ trees with which we can apply Theorem 8. We will use the example from Fig. 9 to facilitate the higher level explanation.

Assume that the grid is oriented so that $m_s$ is the number of columns and $m_ℓ$ is the number of rows (see Fig. 9). The trees that will be built will be based on the columns of one of the split graphs, say $G_2$. A column $i$ of $G_2$ is a path of length $\lceil m_ℓ/2 \rceil - 1$ with $\lceil m_ℓ/2 \rceil$ robots on it. Suppose $k_i$ of these robots have goals outside $G_2$ (the lightly-shaded ones in Fig. 9 (b)), then it is always possible to find $k_i$ robots (the dark-shaded ones in Fig. 9 (b)) on $G_1$ that must go to $G_2$. A tree $T_i$ will be built to allow the exchange of these $2k_i$ robots such that the part of $T_i$ in $G_2$ is simply column $i$. That is, the $m_s$ trees to be built will not have overlaps in $G_2$.

For a column $i$ in $G_2$ with $k_i$ robots to be moved to $G_1$, it is not always possible to find exactly $k_i$ robots on column $i$ of $G_1$. This makes the construction of the trees in $G_1$ is more complex. The construction is done in two steps. In the first step, robots to be moved to $G_2$ are grouped in a distance optimal manner, which induces a preliminary tree structure. Focusing on $G_1$, we know the number of robots that must be moved across the split line in each column (see Fig. 10). For each robot to be moved across the split line, the distance between the robot and all the possible exits of $G_1$ is readily computed. Once these distances are computed, a standard matching procedure can be run to assign each robot an exit point that minimizes the total distance traveled by these robots. The assignment has a powerful property that we will use later. For each robot, either a straight or an $L$ shaped path can be obtained based on the assignment. Merging these paths for robots exiting from the same column then yields a tree for each column (see Fig. 9 (b)). Note that each tree has a single vertical segment.

In the second step, the trees are post-processed to remove crossings between them. Example of such a crossing we refer to is illustrated in Fig. 11 (a) (dotted lines). Formally, we say
two trees $T_1$ and $T_2$ has a crossover if a horizontal path of $T_1$ intersects with a horizontal path of $T_2$, with the additional requirement that one of the involved horizontal path from one tree forms a + with the vertical segment of the other tree. For example, Fig. 11(b) is not considered a crossover.

For each crossover, we update the two trees to remove the crossover, as illustrated in Fig. 11(c). The removal will not change the total distance traveled by the two (or more) affected robots but will change the path for these robots. To see that the process will end, note that one of the two involved paths is shortened. Since there are finite number of such paths and each path can only be shortened a finite number of times, the crossover removal process can get rid of all crossovers. We will show later this can be done in polynomial computation time when we perform algorithm analysis. We note here that the crossover scenario in Fig. 11(c) cannot happen because a removal would shorten the overall length, which contradicts the assumption that these paths have the shortest total distance.

At the end of the crossover removal process, it is possible for the tree structures for different columns to overlap horizontally (see Fig. 12). For two trees that partially overlap with each other (e.g., the left and middle two trees in Fig. 12), one of the trees does not extend lower (row wise) than the row where the overlap occurs. Otherwise, this yields a crossover, which should have already been removed. For two overlapping trees $T_1$ and $T_2$, we say $T_1$ is a follower of $T_2$ if a robot going to $T_2$ must pass through the vertical path of $T_1$. In the example from Fig. 12, $T_1$ is a follower of $T_2$. Similarly, the rightmost (green) tree is a follower of the middle (red) tree.

We state some readily observable properties of overlapping trees: (i) two trees may have at most one overlapping horizontal branch (otherwise, there must be a path crossover), (ii) because of (i), any three trees cannot pair wise overlap at different rows, and (iii) there must be at least one tree that is not a follower, e.g., the left (purple) tree in Fig. 12. We call this tree a leader. From a leader tree, we can recursively collect its followers, and the followers of these followers, and so on so forth. We call such a collection an interacting bundle (e.g., Fig. 12).

With these properties in mind, the grouping operation in a SPLIT-AND-GROUP iteration is carried out as follows. Because robots to be moved from $G_2$ to $G_1$ are on straight vertical paths, there are no interactions among them between different trees. Therefore, we only need to consider interactions of robots on $G_1$. For trees that have no overlap with other trees, Theorem 8 directly applies to complete the robot exchange on these trees in $O(m_1)$ time steps because each tree has diameter at most $2m_1$. In parallel, we can also complete the movement of all robots that should go from $G_1$ to $G_2$ which are not residing on a horizontal tree branch that overlaps with other trees, also in $O(m_1)$ time steps. After these robots are exchanged, we can effectively forget about them.

After the previous step, we are left to deal with robots on overlapping horizontal tree branches that must be moved (e.g., the shaded robots in Fig. 12). It is clear that different interacting bundles do not have any interactions; we only need to focus on a single bundle. This is actually straightforward; we use the example from Fig. 13 to facilitate the proof explanation.

Observe that the problem can be solved for the leader tree (left most tree in Fig. 13). At the same time, for each successive follower tree, the movement of robots can be partially solved for these follower trees. The middle row of Fig. 13 show how this can be done for each tree. Formally, if a horizontal branch is shared by two trees, say $T_2$ and its follower $T_1$, then we obtain a simple exchange problem of moving a few robots through a path on $T_2$. In the figure, these are the first and fourth trees from the left, with the dotted lines marking the path. If a horizontal branch is shared by three or more trees, we get an exchange problem on a tree. In the figure, the middle three trees create such a problem. For this, Theorem 8 applies with minor modifications. All the exchange problems can be carried out in parallel because there are no
further interaction between them. It takes \( O(m_t) \) time steps to complete. We can forget about the involved robots once this step completes and are left with another set of exchange problems, each of which is on a path (e.g., the three problems in the last row of Fig. 13). Lemma 7 applies to yield \( O(m_t) \) required time steps.

Stitching everything together then shows that the first iteration of \( \text{SPLIT}\{G\} \) on a \( m_t \times m_s \) grid can be completed in \( O(m_t) \) time steps. In the next iteration, we are working with grids of sizes \( \lceil m_t/2 \rceil \times m_s \), which requires \( O(\max\{\lceil m_t/2 \rceil, m_s\}) \) time steps. Following the simple recursion, which terminates after \( O(\log m_t) \) iterations, we readily obtain that \( O(m_t) \) time steps is enough to solve the entire problem.

IV. COMPLEXITY AND SOLUTION OPTIMALITY

Properties of the \( \text{SPLIT}\{G\} \) Algorithm

In this section, we establish two key properties of \( \text{SPLIT}\{G\} \), namely, its polynomial running-time and asymptotic solution optimality.

A. Time Complexity of \( \text{SPLIT}\{G\} \)

The \( \text{SPLIT}\{G\} \) algorithm is outlined in Algorithm 1. At Lines 1-2, a partition of the current grid \( G \) is made, over which initial path planning is performed to generate the trees for grouping the robots into the proper subgraph. Then, at Line 3 crossovers are resolved. At Line 4 the final paths are scheduled, from which the robot moves can be extracted. This step also yields where each robot will end up at in the end of the iteration, which becomes the initial configuration for the next iteration (if there is one). After the main iteration steps are complete, at Lines 5-10 the algorithm recursively calls itself on smaller problem instances. The special case here is when the problem is small enough (Line 7), in which case the problem is directly solved without further splitting.

We now proceed to bound the running time of \( \text{SPLIT}\{G\} \). It is straightforward to see that the \( \text{SPLIT} \) routine takes \( O(|V|) = O(m_t m_s) \) running time. MATCH\&PLAN\-PATH can be implemented using the standard Hungarian algorithm [69], which runs in \( O(|V|^3) \) time.

For RESOLVE\( \text{CROSSTINGS} \), we may implement it by starting with an arbitrary robot that needs to be moved across the split line and check whether the path it is on has crossovers that need to be resolved. Checking one path with another can be done in constant time; detecting a crossover then takes \( O(|V|) \) running time. We note that, as a crossover is resolved, one of the two paths will end up being shorter (see, e.g., Fig. 11). We then repeat the process with this shorter path until no more crossover exists. Naively, because the path keeps getting shorter, this process will end in at most \( O(|V|) \) steps. Therefore, all together, RESOLVE\( \text{CROSSTINGS} \) can be completed in \( O(|V|^3) \) running time.

The \( \text{SCHEDULE}\text{MOVES} \) routine simply extracts information from the already planned path set \( P \) and can be completed in \( O(|V|) \) running time. The SOLVE routine takes constant time.

Adding everything up, an iteration of \( \text{SPLIT}\{G\} \) can be carried out in \( O(|V|^3) \) time using a naive implementation. Summing over all iterations, the total running time is

\[
O(|V|^3) + 2O\left(\frac{|V|}{2}\right)^3 + 4O\left(\frac{|V|}{4}\right)^3 + \ldots = O(|V|^3),
\]

which is low-polynomial with respect to the input size.

Algorithm 1: \( \text{SPLIT}\{G\} \)

\[
\begin{align*}
\text{Input :} & & \text{G} = (V, E): \text{an } m_t \times m_s \text{ grid graph} & \\
& & X_I: V \rightarrow \mathbb{Z}: \text{initial robot configurations} & \\
& & X_G: V \rightarrow \mathbb{Z}: \text{goal robot configurations} & \\
\text{Output:} & & M = \langle M_1, M_2, \ldots \rangle: \text{a sequence of moves} & \\
\text{\%Run matching and construct initial trees} & & (G_1, G_2) & \leftarrow \text{SPLIT}(G) & \\
\text{\%Remove crossovers} & & P & \leftarrow \text{MATCH\&PLAN\-PATH}(G, X_I, X_G) & \\
\text{\%Schedule the sequence of moves} & & (M, X'_1) & \leftarrow \text{SCHEDULE\text{MOVES}}(P) & \\
\text{\%Recursively solve smaller sub-problems} & & \text{foreach } G_i, i = 1, 2 \text{ do} & \\
& & \text{if } \text{row}(G_i) \leq 3 \text{ and col}(G_i) \leq 3 \text{ then} & \\
& & M & = M + \text{SOLVE}(G_i, X'_1|G_i, X_G|G_i) & \\
& & \text{else} & \\
& & M & = M + \text{SPLIT}\{G_i, X'_1|G_i, X_G|G_i\} & \\
& & \text{end} & \\
\text{return } M & & & & \\
\end{align*}
\]

Fig. 13. An example interacting bundle in detail. The top row is the initial configuration of the robots on the overlapping horizontal tree branches to be moved through the vertical paths. The numbers on the top of the vertical paths mark how many robots should be moved through that path. The middle row and the bottom rows mark how the exchanges can be completed in two steps.
B. Constant Factor Time- and Distance-Optimality

Having established that SplitAndGroup is a polynomial time algorithm that solves MPP with sub-linear makespan, we now show that a solution produced by SplitAndGroup is in fact only a constant factor away from the best possible, in expectation, for both TMPP and DMPP. That is, SplitAndGroup in fact computes an asymptotically optimal solution simultaneously for time- and distance-based objectives.

Theorem 9. Let \((G, X_I, X_G)\) be an MPP instance in which \(G\) is an \(m_t \times m_s\) grid, and \(X_I\) and \(X_G\) are selected uniformly randomly. Then SplitAndGroup computes constant factor optimal solutions in expectation with respect to the makespan and the total distance objectives.

Proof. Let \(i\) be a specific robot with start \(s_i \in X_I\) and \(g_i \in X_G\) be its start and goal locations on \(G\), respectively. Since \(X_I\) and \(X_G\) are random, the distance between \(s_i\) and \(g_i\) on \(G\) can be readily shown to be \(\Omega(m_t)\). This implies that the minimum possible makespan is \(\Omega(m_t)\). Since SplitAndGroup produces solutions with \(O(m_t)\) makespan, it achieves constant factor approximation regarding optimal makespan.

Similarly, each robot incurs a minimum travel distance of \(\Omega(m_t^2)\); therefore, the minimum total distance for all robots, in expectation, is \(\Omega(m_s m_t^2)\) because there are \(m_s m_t\) robots. On the other hand, because SplitAndGroup produces a solution with an \(O(m_t)\) makespan, each robot can travel a distance of \(O(m_t)\). Summing this over all robots, the solution from SplitAndGroup has a total distance of \(O(m_s m_t^2)\). This matches the lower bound \(\Omega(m_s m_t^2)\).

V. Generalizations

In this section, we show that SplitAndGroup readily generalizes to environments other than 2D rectangular grids, including high dimensional grids and continuous environments.

A. High Dimensions

SplitAndGroup can be readily extended to work for grids of arbitrary dimensions. For dimensions \(d \geq 2\), two updates to SplitAndGroup are needed: (i) the split line should be updated to a split plane of dimension \(d-1\), and (ii) the crossover check now takes \(O(d)\) time instead of \(O(1)\) time. Other than these changes, the rest of SplitAndGroup continue to work without major change. The updated SplitAndGroup algorithm for dimension \(d\) therefore runs in \(O(d|V|^3)\) time.

B. Well-Connected Environments

The selection of \(G\) as a grid plays a critical role in proving the desirable properties of SplitAndGroup. In particular, two features of grid graphs are used. First, grids are composed of small cycles, which allow the 2-switch operation to be carried out locally. This in turn allows multiple 2-switch operations to be carried out in parallel. Second, restricting to two adjacent rows (or columns) of a rectangular grid (e.g., row 4 and row 5 in Fig. 9(a)), multiple 2-switches can be completed between these two rows in constant number of steps. As long as the environment possesses these two features, SplitAndGroup works. We call such environments well-connected.

More precisely, a well-connected environment, \(E\), is one with the following properties. Let \(G\) be an \(m_s \times m_s\) rectangular grid that contains \(E\). Unlike earlier grids, here, \(G\) is not required to have unit edge lengths; a cell of \(G\) is only required to be of rectangular shape with \(O(1)\) side lengths. Let \(r_1\) and \(r_2\) be two arbitrary adjacent rows of \(G\), and let \(c_1 \in r_1, c_2 \in r_2\) be two neighboring cells (see, e.g., Fig. 14). The only requirement over \(E\) is that a robot in \(c_1\) and a robot in \(c_2\) may exchange residing cells locally, without affecting the configuration of other robots. In terms of the example in Fig. 14, the two shaded robots (other robots are not drawn) must be able to exchange locations in constant makespan within a region of constant radius. The requirement then implies that parallel exchanges of robots between \(r_1\) and \(r_2\) can be performed with a constant makespan. The same requirement applies to two adjacent columns of \(G\). Subsequently, given an arbitrary well-connected environment \(E\) and an initial robot configuration \(X_I\), the steps from SplitAndGroup can be readily applied to reach an arbitrary \(X_G\) that is a permutation of \(X_I\). As long as pairwise robot exchanges can be computed efficiently, the overall generalized SplitAndGroup algorithm also runs efficiently while maintaining the optimality guarantees. We note that the definition of well-connectedness can be further generalized to certain continuous settings. Fig. 15 provides a discrete example and a continuous example of well-connected settings, which include both the environment and the robots.

Fig. 14. Illustration of a well-connected non-grid graph environment.

Fig. 15. Examples of well-connected settings, with both environments and robots.

As mentioned in Sec. 9 well-connected environments are frequently found in real-world applications, e.g., automated warehouses at Amazon and road networks in cities like Manhattan. Our theoretical results implies that such environments
VI. CONCLUSION AND FUTURE WORK

In this work, we developed a low-polynomial time algorithm, SPLITANDGROUP, for solving the multi-robot path planning problem in grids and grid-like, well-connected environments. In expectation, the solution produced by SPLITANDGROUP is within a constant factor of the best possible in terms of makespan and total distance metrics. SPLITANDGROUP applies to problems with the maximum possible density in which environments may be fully occupied by robots.

The development of SPLITANDGROUP opens up many possibilities for promising future work. On the theoretical side, SPLITANDGROUP gets us closer to the goal of finding a PTAS (polynomial time approximation scheme) for optimal multi-robot path planning. Also, it would be desirable to remove the probabilistic element (i.e., the “in expectation” part) from the guarantees. On the practical side, noting that we have only looked at the case with the highest robot density, it is promising to exploit the combination of divide-and-conquer and network flow techniques to seek more optimal algorithms for cases with lower robot density.

REFERENCES

[1] M. A. Erdmann and T. Lozano-Pérez, “On multiple moving objects,” in Proceedings IEEE International Conference on Robotics & Automation, 1986, pp. 1419–1424.
[2] A. Zelinsky, “A mobile robot exploration algorithm,” IEEE Transactions on Robotics & Automation, vol. 8, no. 6, pp. 707–717, 1992.
[3] 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.
[4] 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.
[5] D. Silver, “Cooperative pathfinding,” in The 1st Conference on Artificial Intelligence and Interactive Digital Entertainment, 2005, pp. 23–28.
[6] M. R. K. Ryan, “Exploiting subgraph structure in multi-robot path planning,” Journal of Artificial Intelligence Research, vol. 31, pp. 497–542, 2008.
[7] R. Jansen and N. Sturtevant, “A new approach to cooperative pathfinding,” in In International Conference on Autonomous Agents and Multi-agent Systems, 2008, pp. 1401–1404.
[8] R. Luna and K. E. Bekris, “Push and swap: Fast cooperative pathfinding with completeness guarantees,” in Proceedings International Joint Conference on Artificial Intelligence, 2011, pp. 294–300.
[9] T. Standley and R. Korf, “Complete algorithms for cooperative pathfinding problems,” in Proceedings International Joint Conference on Artificial Intelligence, 2011, pp. 668–673.
[10] J. van den Berg, J. Snoeyink, M. Lin, and D. Manocha, “Centralized path planning for multiple robots: Optimal decoupling into sequential plans,” in Robotics: Science and Systems, 2009.
[11] K. Solovey and D. Halperin, “k-color multi-robot motion planning,” in Proceedings Workshop on Algorithmic Foundations of Robotics, 2012.
[12] J. Yu and S. M. LaValle, “Multi-agent path planning and network flow,” in Algorithmic Foundations of Robotics X, Springer Tracts in Advanced Robotics. Springer Berlin/Heidelberg, 2013, vol. 86, pp. 157–173.
[13] M. Turpin, K. Mohta, N. Michael, and V. Kumar, “CAPT: Concurrent assignment and planning of trajectories for multiple robots,” International Journal of Robotics Research, vol. 33, no. 1, pp. 98–112, 2014.
[14] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, Principles of Robot Motion: Theory, Algorithms, and Implementations. Cambridge, MA: MIT Press, 2005.
[15] J. van den Berg, M. C. Lin, and D. Manocha, “Reciprocal velocity obstacles for real-time multi-agent navigation,” in Proceedings IEEE International Conference on Robotics & Automation, 2008, pp. 1928–1935.
[16] M. S. Branicky, M. M. Curtiss, J. Levine, and S. Morgan, “Sampling-based control and verification of hybrid systems,” IEEE Proceedings Control Theory and Applications, vol. 153, no. 5, p. 575, 2006.
[17] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” The international journal of robotics research, vol. 5, no. 1, pp. 90–98, 1986.
[18] M. G. Earl and R. D’Andrea, “Iterative milp methods for vehicle-control problems,” IEEE Transactions on Robotics, vol. 21, no. 6, pp. 1158–1167, 2005.
[19] K. E. Bekris, K. I. Tsianos, and L. E. Kavraki, “A decentralized planner that guarantees the safety of communicating vehicles with complex dynamics that replan online,” in 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2007, pp. 3784–3790.
[20] J. Alonso-Mora, R. Knepper, R. Siegwart, and D. Rus, “Local motion planning for collaborative multi-robot manipulation of deformable objects,” in 2015 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2015, pp. 5495–5502.
[21] R. A. Knepper and D. Rus, “Pedestrian-inspired sampling-based multi-robot collision avoidance,” in 2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication. IEEE, 2012, pp. 94–100.
[22] D. Halperin, J.-C. Latombe, and R. Wilson, “A general framework for assembly planning: The motion space approach,” Algorithmica, vol. 26, no. 3-4, pp. 577–601, 2000.
[23] B. Naiji, Theory of Automatic Robot Assembly and Programming. Chapman & Hall, 1992.
[24] S. Rodriguez and N. M. Amato, “Behavior-based evacuation planning,” in Proceedings IEEE International Conference on Robotics & Automation, 2010, pp. 350–355.
[25] T. Balch and R. C. Arkin, “Behavior-based formation control for multi-tirobot teams,” IEEE Transactions on Robotics & Automation, vol. 14, no. 6, pp. 926–939, 1998.
[26] S. Poduri and G. S. Sukhatme, “Constrained coverage for mobile sensor networks,” in Proceedings IEEE International Conference on Robotics & Automation, 2004.
[27] B. Shucker, T. Murphy, and J. K. Bennett, “Switching rules for decentralized control with simple control laws,” in American Control Conference, Jul 2007, pp. 1485–1492.
[28] 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.
[29] H. Tanner, G. Pappas, and V. Kumar, “Leader-to-formation stability,” IEEE Transactions on Robotics & Automation, vol. 20, no. 3, pp. 445–455, Jun. 2004.
[30] D. Fox, W. Burgard, H. Kruppa, and S. Thrun, “A probabilistic approach to collaborative multi-robot localization,” Autonomous Robots, vol. 8, no. 3, pp. 325–344, Jun. 2000.
[31] J. Ding, K. Chakrabarty, and R. B. Fair, “Scheduling of microfluidic operations for reconfigurable two-dimensional electrowetting arrays,” IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, vol. 20, no. 12, pp. 1463–1468, 2001.
[32] E. J. Griffith and S. Akella, “Coordinating multiple droplets in planar array digital microfluidic systems,” International Journal of Robotics Research, vol. 24, no. 11, pp. 933–949, 2005.
[33] M. J. Matarić, M. Nilsson, and K. T. Simsarian, “Cooperative multi-robot box pushing,” in Proceedings IEEE/RSJ International Conference on Intelligent Robots & Systems, 1995, pp. 556–561.
[34] 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.
[35] J. S. Jennings, G. Whelan, and W. F. Evans, “Cooperative search and rescue with a team of mobile robots,” in Proceedings IEEE International Conference on Robotics & Automation, 1997.
[36] J. H. Reif, “Complexity of the generalized mover’s problem,” DTIC Document, Tech. Rep., 1985.
[37] J. F. Canny, The Complexity of Robot Motion Planning. Cambridge, MA: MIT Press, 1988.

are in fact quite optimal in their design, in terms of being able to efficiently robot routing.
[38] P. Spirakis and C. K. Yap, “Strong NP-hardness of moving many discs,” Information Processing Letters, vol. 19, no. 1, pp. 55–59, 1984.

[39] M. Saha and P. Isto, “Multi-robot motion planning by incremental coordination,” in 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2006, pp. 5960–5963.

[40] G. Sharon, R. Stern, A. Felner, and N. Sturtevant, “Conflict-Based Search for Optimal Multi-Agent Path Finding,” in Proc of the Twenty-Sixth AAAI Conference on Artificial Intelligence, 2012.

[41] 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.

[42] W. Hönig, T. S. Kumar, L. Cohen, H. Ma, H. Xu, N. Ayaniyan, and S. Koenig, “Multi-agent path finding with kinematic constraints,” in ICAPS, 2016, pp. 477–485.

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

[44] P. Surynek, “Towards optimal cooperative path planning in hard setups through satisfiability solving,” in Proceedings 12th Pacific Rim International Conference on Artificial Intelligence, 2012.

[45] 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.

[46] D. M. Kornhauser, “Coordinating pebble motion on graphs, the diameter of permutation groups, and applications,” Ph.D. dissertation, Massachusetts Institute of Technology, 1984.

[47] B. Bollobás, Modern graph theory. Springer Science & Business Media, 2013, vol. 184.

[48] H. W. Kuhn, “The Hungarian method for the assignment problem,” Naval Research Logistics Quarterly, vol. 2, pp. 83–97, 1955.