Constant-Factor Time-Optimal Multi-Robot Routing on High-Dimensional Grids

Jingjin Yu
Department of Computer Science, Rutgers, the State University of New Jersey, Piscataway, New Jersey, USA
Email: jingjin.yu@cs.rutgers.edu

***Abstract*—Let $G = (V, E)$ be an $m_1 \times \ldots \times m_k$ grid for some arbitrary constant k. We establish that $O(\sum_{i=1}^{k} m_i)$ (makespan) time-optimal labeled (i.e., each robot has a specific goal) multi-robot path planning can be realized on G in $O(|V|^2)$ running time, even when vertices of G are fully occupied by robots. When all dimensions are of equal sizes, the running time approaches $O(|V|)$. Using this base line algorithm, which provides average case $O(1)$-approximate (i.e., constant-factor) time-optimal solutions, we further develop a first worst case $O(1)$-approximate algorithm that again runs in $O(|V|^2)$ time for two and three dimensions. We note that the problem has a worst case running time lower bound of $\Omega(|V|^2)$.**

I. Introduction

We study a time-optimal multi-robot routing or path planning problem on grids and grid-like settings, with the assumption that each vertex of the grid is occupied by a robot, i.e., the robot density is maximal. Our work brings several breakthroughs. First, on a k-D grid $G = (V, E)$ for some constant k, our algorithm, iSAG, improves the running time of the *average case* $O(1)$-approximate (makespan) time-optimal SPLITANDGROUP (SAG) algorithm from [1] from $O(|V|^3)$ to a sub-quadratic $o(|V|^2)$ for most cases and $O(|V|^2)$ in the worst case (when G is degenerate and nearly one dimensional). When all dimensions are of equal sizes, iSAG runs in a nearly linear $O(|V|)$ time for large k. Second, building on iSAG, we present a *worst case* $O(1)$-approximate algorithm, PARTITIONANDFLOW (PAF), that generalizes a key result from [2] (Theorem 3 in [2]), which only works in 2D, to 3D. Our work, in both 2D and 3D cases, is developed independently of [2]. Third, multiple algorithmic techniques in our work, particularly Lemma 5 and Theorem 10, which help enable iSAG and PAF, reveal fundamental structures present in general routing problems and are of independent interest.

From the practical standpoint, our results are of significance in multiple application domains including robotics and network routing. Particularly, in robotics, our results imply that even in highly dense settings, if among a group of labeled robots the maximum distance between a robot and its goal is of distance d_g, then it is possible to compute a routing plan that solves the entire problem that requires $O(d_g)$ makespan in only quadratic time, assuming that the robots travel at no faster than unit speed. Further exploration of the algorithmic insights from our work may lead to more optimal coordination algorithms for applications including warehousing [3], automated container port management [4], and coordinated aerial flight [5]. As noted in [2], algorithms like PAF help resolve open questions regarding routing strategies for inter-connected mesh networks. Indeed, solving multi-robot routing on grid and grid-like structures is equivalent to finding vertex disjoint paths in the underlying network, extended over discrete time steps.

Our presentation is fairly compact due to limited space. Complete proofs and additional results are provided in [6].

Related work. Multi-robot path planning, from both the algorithmic and the application perspectives, has been studied extensively [7]–[21], covering many application domains [5], [22]–[30]. Multi-robot path and motion planning is known to be computationally hard under continuous settings [31], [32], even when the robots are unlabeled [33], [34]. While the general multi-robot motion planning problem seems rather difficult to tackle, relaxed unlabeled continuous problems are solvable in polynomial time even near optimally [16], [35].

Restricting our attention to the discrete and labeled setting, in contrast to the continuous setting, feasible solutions are more readily computable. Seminal work by Kornhauser et al. [36], which builds on the work by Wilson [37], establishes that a discrete instance can be checked and solved in $O(|V|^3)$ time on a graph $G = (V, E)$. Feasibility test can in fact be completed in linear time [38]–[40]. Optimal solutions remain difficult to compute in the discrete settings, however, even on planar graphs [2], [41]. Whereas many algorithms have been proposed toward optimally solving the discrete labeled multi-robot path planning problems [12], [42]–[49], few provide simultaneous guarantees on solution optimality and (polynomial) running time. This leads to the development of polynomial time methods that also provide these desirable guarantees [1], [2].

II. Preliminaries

Let $G = (V, E)$ be a simple, undirected, and connected graph. A set of $n \leq |V|$ robots labeled 1-n may move synchronously on G in a collision-free manner described as follows. At integer (time) steps starting from $t = 0$, each robot must reside on a unique vertex $v \in V$, inducing a *configuration* X_t of the robots as an injective map $X_t : \{1, \ldots, n\} \to V$, specifying which robot occupies which vertex at step t (see Fig. 1). From step t to step $t + 1$, a robot may *move* from its current vertex to an adjacent one under two collision avoidance constraints: *(i)* X_{t+1} is injective, i.e., each robot occupies a unique vertex, and *(ii)* for $1 \leq i, j \leq n$, $i \neq j$, $X_t(i) = X_{t+1}(j) \to X_t(j) \neq X_{t+1}(i)$, i.e., no two robots may *swap* locations in a single step. If all individual robot

moves between some X_t and X_{t+1} are valid (i.e., collision-free), then $M_t = (X_t, X_{t+1})$ is a valid *move* for all robots. Multiple such moves can be chained together to form a sequence of moves, e.g., $(X_t, X_{t+1}, \ldots, X_{t+\Delta t})$.

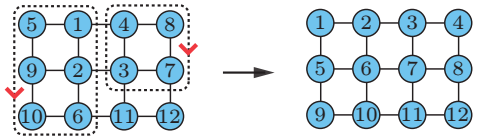


Fig. 1. Graph-theoretic formulation of the multi-robot path planning problem. (a) A configuration of 12 robots on a 4×3 grid. (b) A configuration that is reachable from (a) in a single synchronous move through simultaneous rotations of robots along two disjoint cycles.

Under the model, a multi-robot path planning (MPP) instance is fully specified by a 3-tuple (G, X_I, X_G) in which $X_I = X_0$ and X_G are the initial and goal configurations, respectively. To handle the most difficult case, we assumed that $n = |V|$, i.e., the number of robots is the maximum possible. We note that the case of $n' < |V|$ may be reduced to the $n = |V|$ case by placing $(|V| - n')$ "virtual" robots arbitrarily on vertices that are empty in X_I and X_G.

For this study, G is assumed to be an $m_1 \times \ldots \times m_k$ grid with $m_1 \geq \ldots \geq m_k \geq 2$ and $|V| = m_1 \ldots m_k \geq 6$. Such a grid graph G is also meant whenever the term *grid* is used in the paper without further specifications. Given an MPP instance and a feasible solution, as a sequence of moves $M = (X_I = X_0, X_1, \ldots, X_{t_f} = X_G)$, we define the solution's *makespan* as the length t_f of the sequence. For an instance $p = (G, X_I, X_G)$, let $d(v_1, v_2)$ denote the distance between two vertices $v_1, v_2 \in V$, assuming each edge has unit length. We define the *distance gap* between X_I and X_G as

$$d_g(p) = \max_{1 \leq i \leq |V|} d(X_I(i), X_G(i)),$$

which is an underestimate of the minimum makespan for p. The main aim of this work is to establish a polynomial time algorithm that computes solutions with $O(d_g(p))$ makespan for an arbitrary p and $k = 2, 3$. In other words, the algorithm produces, in the worst case, $O(1)$-approximate makespan optimal solutions because $d_g(p)$ is an underestimate of the makespan of p. When the instance p is clear, we use d_g in place of $d_g(p)$.

III. Improved Average Case $O(1)$-Approximate Makespan Algorithm

Our worst case $O(1)$-approximate algorithm makes use of an average case $O(1)$-approximate algorithm for MPP that improves over the SplitAndGroup (SAG) algorithm [1]. Main properties of SAG are summarized below.

Theorem 1 (SAG [1]). *Let (G, X_I, X_G) be an MPP instance with $G = (V, E)$ being an $m_1 \times m_2$ grid. Then, a solution with $O(m_1 + m_2)$ makespan can be computed in $O(|V|^3)$ time.*

To state our significant improvements, we briefly describe how SAG operates on a $m_1 \times m_2$ grid G. SAG recursively splits G into halves along a longer dimension. During the first iteration, G is *split* into two $\frac{m_1}{2} \times m_2$ grids (assuming m_1 is even) G_1 and G_2. Then, all robots whose goals belong to G_2 will be routed to G_2. This also forces all robots whose goals belong to G_1 to G_1 because G is fully occupied. This is the *grouping* operation in SAG. The step effectively partitions all robots on G into two equivalence classes (those should be in G_1 and those should be in G_2); there is no need to distinguish robots within each class during the current iteration. Fig. 2 illustrates graphically what is to be achieved in the grouping operation in an iteration of SAG.

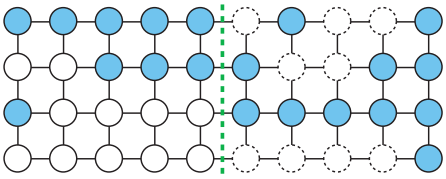


Fig. 2. On a 10×4 grid, the shaded robots have goals on the right 5×4 grid. The grouping operation of a SAG iteration seeks to move the 9 shaded robots on the left 5×4 grid to exchange with the 9 unshaded robots marked with dashed boundaries on the right 5×4 grid.

To be able to move the robots to the desired halves of G, it was noted [49] that an exchange of two robots can be realized on a 3×2 grid using a constant number of moves (Fig. 3).

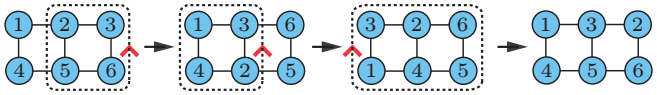


Fig. 3. Robots 2 and 3 may be "swapped" using three synchronous moves on a 3×2 grid. This implies that arbitrary configuration on a 3×2 grid can be realized in a constant number of moves.

The local "swapping" primitives can be executed in parallel on G, which implies Lemma 2 as follows. An illustration of the operation is provided in Fig. 4.

Lemma 2 (Lemma 6 in [1]). *On a length ℓ path embedded in a grid, a group of indistinguishable robots may be arbitrarily rearranged using $O(\ell)$ makespan. Multiple such rearrangements on vertex disjoint paths can be carried out in parallel.*

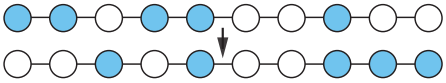


Fig. 4. On a length ℓ path is embedded in a grid, Lemma 2 guarantees that the arbitrary distribution of a group of robots can be done in $O(\ell)$ makespan.

Lemma 2 further implies Lemma 3. Fig. 5 illustrates graphically the operation realized by Lemma 3.

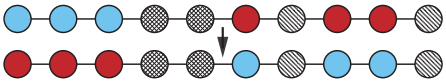


Fig. 5. Assuming the grid-embedded path has a length of ℓ, Lemma 3 guarantees that the swapping of the two separated groups of robots, up to $\frac{\ell}{2}$ per group, can be done in $O(\ell)$ makespan.

Lemma 3 (Lemma 7 in [1]). *On a length ℓ path embedded in a grid, two groups of robots, equal in number and initially located on two disjoint portions of the path, may exchange locations in $O(\ell)$ makespan. Multiple such exchanges on vertex disjoint paths can be carried out in parallel.*

Lemma 2 and Lemma 3 both demand a running time of $O(\ell^2)$; some problems require $\Omega(\ell^2)$ time to simply write down the solution, e.g., when $\frac{\ell}{2}$ robots need to be moved on

a path of length ℓ. Additional results were developed over Lemma 3 in [1] to complete the grouping operation involving routing of robots on trees, embedded in a grid, that may overlap. We provide a new algorithm, iSAG, that simplifies the process and significantly improves the running time. We note that, to complete the grouping over the example from Fig. 2, we may reconfigure robots on the left 5×4 grid so that for each row, robots to be exchanged across the split line are equal in number (see Fig. 6). Lemma 3 then applies.

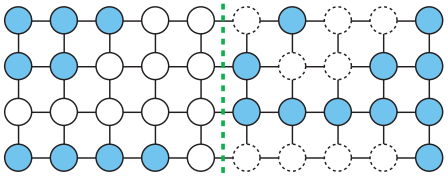


Fig. 6. We would like to reconfigure robots on the left 5×4 half of Fig. 2 to the configuration as shown. The right 5×4 portion will not be touched in the operation. In this configuration, robots do not need to move between different rows to complete the grouping operation, using Lemma 3.

To perform the reconfiguration, we begin by assigning labels to the robots as illustrated and explained in Fig. 7. These labels are only for pairing up robots for the reconfiguration; keep in mind that the shaded robots are in fact indistinguishable in the execution of the grouping operation.

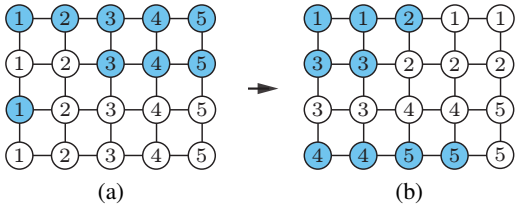


Fig. 7. (a) and (b) correspond to the left 5×4 grids from Fig. 2 and Fig. 6, respectively. We would like to reconfigure the shaded robots to go from (a) to (b) (ignoring the labels). In (a), shaded robots are assigned labels based on the column they belong to. In (b), from top to bottom and left to right, we sequentially assign each shaded labeled robot from (a) a goal.

With the labeling, we set up a bipartite graph as follows. One of the partite set $\{v_i^1\}$ (e.g., $\{v_1^1, \ldots, v_5^1\}$ in Fig. 8) represents the initial columns and the other set $\{v_j^2\}$ (e.g., $\{v_1^2, \ldots, v_5^2\}$ in Fig. 8) the goal columns. We draw an edge between v_i^1 and v_j^2 if a shaded robot labeled i ends up at a goal column j. For example, in Fig. 7, shaded robots with label 1 in (a) ends up at columns 1 and 2 in (b), yielding the edges (v_1^1, v_1^2) and (v_1^1, v_2^2) in Fig. 8. If a goal column j contains multiple shaded robots with label i, then multiple edges between v_i^1 and v_j^2 are added. Note that, if we also add the edges for the unshaded robots in Fig. 7 in a similar manner, the bipartite graph will be d-regular where d is the number of rows in the original grid ($d = 4$ in the provided example).

With the bipartite graph constructed, we proceed to obtain a set of up to d maximum matchings, which is always possible because our bipartite graph is a sub-graph of a d-regular bipartite graph. By Hall's theorem [50], a perfect matching may be obtained on a d-regular bipartite graph, the removal of which leaves a $(d-1)$-regular bipartite graph. From the obtained set of matchings (e.g., using [51]), we apply Lemma 2 to distribute the robots vertically so that a robot matched in the i-th matching gets moved to the i-th row. In our example, the first set is $\{1-1, 2-3, 3-2, 5-4\}$, which means that a set of three robots labeled $1, 2, 3$, and 5 should be moved to the first row. Doing this for all matching sets shown in Fig. 7(a) yields the configuration in Fig. 9(a).

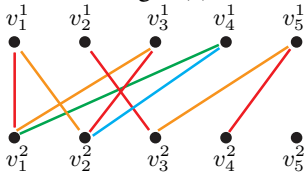


Fig. 8. A bipartite graph constructed for rearranging robots. The 4 colorings of the edges indicate a possible set of 4 matchings, which are $\{1-1, 3-2, 5-3\}$ (red), $\{1-2, 3-1, 5-4\}$ (orange), $\{2-3, 4-1\}$ (green), $\{4-2\}$ (purple).

In a second round, the robots are permuted within their row, again using the matching result. In the example, the first set $\{1-1, 2-3, 3-2, 5-4\}$ says that robots $1, 2, 3$, and 5 on the first row should be moved to columns $1, 3, 2$, and 4. Going from $1, 2, 3, 5$ to $1, 3, 2, 4$ is possible with Lemma 2 because the labels here are nominal; in effect, four indistinguishable robots must be moved to to columns $1, 2, 3$, and 4 without consideration of the absolute order. For the configuration in Fig. 9(a), this round yields the configuration in Fig. 9(b). We note that the perfect bipartite matching technique mentioned here was due to [52], in which a variation of it is used for a different reconfiguration task.

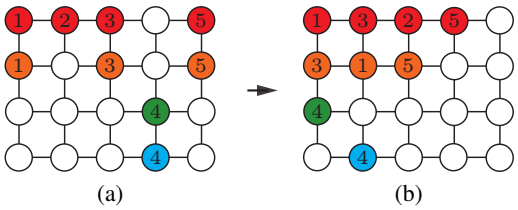


Fig. 9. (a) The initial permutation of columns of Fig. 7(a) using the bipartite matching result. (b) A second row-based permutation of (a) using the bipartite matching result. Our procedure operates following the sequence Fig. 7(a) $\rightarrow$ Fig. 9(a) $\rightarrow$ Fig. 9(b) $\rightarrow$ Fig. 7(b).

The labeled robots that need to be moved are now in the correct columns. One last column permutation then moves the robots in place. In the example, this is going from Fig. 9(b) to Fig. 7(b). We summarize the the discussion in a lemma.

Lemma 4. *On an $m_1 \times m_2$ grid, the reconfiguration of a group of indistinguishable robots between two arbitrary configurations can be completed using $O(m_1+m_2)$ makespan in $O(m_1^2 m_2 + m_1 m_2^2)$ time.*

Proof: The procedure is already fully described; here, we analyze its performance. The procedure operates in three phases, each requiring a makespan of either $O(m_1)$ or $O(m_2)$ (because only one dimension of the $m_1 \times m_2$ grid is involved in each phase). The overall makespan is then $O(m_1 + m_2)$. Regarding the computation time, each invocation of the procedure from Lemma 2 or Lemma 3 on an $m_1 \times m_2$ grid takes $O(m_1^2)$ or $O(m_2^2)$ time; doing these in parallel on the grid then takes $O(m_1^2 m_2 + m_1 m_2^2) = O(m_1^2 m_2)$ time. For doing the bipartite matching, we may invoke the regular bipartite matching algorithm [51] d times which requires a

total time $O(d|E_B|)$ where E_B is edge set of the bipartite graph. If we make m_1 the number of columns, then $d = m_2$ and $|E_B| = m_1 m_2/2$. The total time spent on matching is $O(m_1 m_2^2)$ and the total running time is then as given. ∎

Lemma 4 readily generalizes to k-D via induction.

Lemma 5 (k-D Shuffle)**.** *On an $m_1 \times \ldots \times m_k$ grid, the reconfiguration of a group of indistinguishable robots between two arbitrary configurations can be completed using $O(\sum_{i=1}^{k} m_i)$ makespan in $O((\sum_{i=1}^{k} m_i)(\prod_{i=1}^{k} m_i))$ time.*

Proof: The proof uses straightforward inductive argument. Having shown that the 2D cases holds, assume the k-D case holds. For $k+1$ dimensions, similar to how we partition the 2D case into $\frac{m_1}{2}$ columns of size m_2, here we first make a partition over the half of the $m_1 \times \ldots m_{k+1}$ grid into $\frac{m_1 \ldots m_k}{2}$ columns of size m_{k+1} each. After the bipartite matching, each of the $\frac{m_1 \ldots m_k}{2}$ columns is permuted in parallel. Now, we have m_{k+1} *layers* of size $\frac{m_1}{2} \times \ldots \times m_k$ that need to be shuffled, which we may complete via the induction hypothesis. This is then followed by another permutation of $\frac{m_1 \ldots m_k}{2}$ columns of size m_{k+1} to complete the procedure. ∎

With Lemma 5, we state the theorem that supports iSAG.

Theorem 6 (iSAG, k-D)**.** *An $O(k \sum_{i=1}^{k} m_i)$ makespan solution can be computed in $O(k(\sum_{i=1}^{k} m_i)(\prod_{i=1}^{k} m_i))$ time for an MPP instance on an $m_1 \times \ldots \times m_k$ grid.*

Proof: We iteratively apply Lemma 5 and then Lemma 3 (in parallel) to dimensions $1, \ldots, k$. After the first k iterations, due to parallelism, we incur a total makespan of $O(k \sum_{i=1}^{k} m_i)$ and a total running time of $O(k(\sum_{i=1}^{k} m_i)(\prod_{i=1}^{k} m_i))$. Now, the initial problem is partitioned into 2^k problems with grids of sizes $\frac{m_1}{2} \times \ldots \times \frac{m_k}{2}$. For the next k iterations, each will incur a makespan of $O(\frac{k}{2}(\sum_{i=1}^{k} m_i))$ and a running time $O(\frac{k}{2}(\sum_{i=1}^{k} m_i)(\prod_{i=1}^{k} m_i))$. Adding everything up, we end up with the stated makespan and running time. ∎

We note that when all dimensions are of roughly equal $O(|V|^{\frac{1}{k}})$ size, iSAG yields a makespan of $O(k^2|V|^{\frac{1}{k}})$ and a running time of $O(k^2|V|^{\frac{k+1}{k}})$. Because for fixed G, most instances have makespan of $O(\sum_{i=1}^{k} m_i))$ (see Lemma 9 of [1], which generalizes to k-D), iSAG is an average case $O(1)$-approximate optimal algorithm.

IV. PAF IN TWO DIMENSIONS

After establishing iSAG, we now introduce PARTITIONANDFLOW (PAF) for 2D, which uses a general approach that applies to high dimensions. In sketching PAF, we remark that it works on a problem (G, X_I, X_G) by updating X_I. It first creates some intermediate X_G^1 based on X_I and X_G and solve the problem (G, X_I, X_G^1), leaving a new problem (G, X_G^1, X_G). The process is repeated until X_I is updated to match X_G. It is important to keep this in mind in reading the sketch of PAF. As the name suggests, PAF partitions an MPP instance on a grid into smaller cells and organize the flow of robots through these cells. The partition is in effect a form of decoupling that is more general than iSAG's splitting scheme.

For a given MPP instance $p = (G, X_I, X_G)$ with G being an $m_1 \times m_2$ grid, PAF starts by computing d_g, the distance gap for the problem. In the main case, $d_g = o(m_2)$. That is, for any robot i, $d(X_I(i), X_G(i)) = o(m_2)$. This means that G may be partitioned into square cells of size $5d_g \times 5d_g$. This is the *partition* operation in PAF (see Fig. 10 for an illustration). For the moment, we assume that a perfect partition can be achieved, i.e., m_1 and m_2 are both integer multiples of $5d_g$; the assumption is justified later.

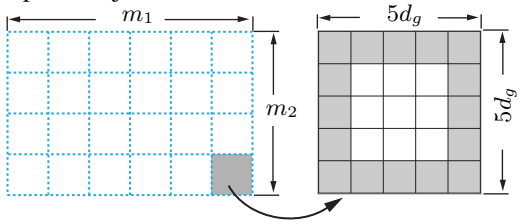


Fig. 10. Partitioning of an $m_1 \times m_2$ grid into 6×4 cells. Each cell has a size of $5d_g \times 5d_g$. Within a cell (the figure on the right), only robots located of a distance no more than d_g from the border may have goals outside the cell.

The partition scheme, as a refinement to the splitting scheme from iSAG, has the property that only robots of distance d_g from a cell boundary may have goals outside the cell by the definition of d_g (for more details, see Fig. 11). This means that between two cells that share a vertical or horizontal boundary, at most $10d_g^2$ robots need to cross that boundary. If we only count the net exchange, then the number reduces to $5d_g^2$.

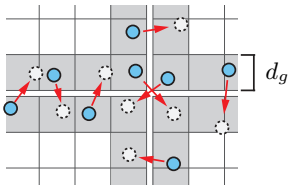


Fig. 11. An illustration of the d_g thick boundary areas of four adjacent cells. Any net robot exchange between two cells must happen in this region by the definition of d_g.

Over the partition, PAF will build a flow between the cells treating each cell as a node in a graph. To be able to translate the flow into feasible robot movements, the flow should only happen between adjacent cells that share a boundary. However, as illustrated in Fig. 11, it is possible for a robot to have initial and goal configurations that are separated into *diagonally adjacent* cells which do not share boundaries. To resolve this, we may update the goals for these robots using robots from another cell that is adjacent to both of the involved cells. Fig. 12 illustrates how one such robot can be processed. We call this operation *diagonal rerouting*, which will create a new configuration X_G^1 of the robots on G. iSAG is then invoked to solve (G, X_I, X_G^1). iSAG will do so locally on $4d_g \times 4d_g$ regions that span equal parts of four adjacent cells.

Then, PAF creates another intermediate configuration X_G^2 for moving robots between each vertical or horizontal cell boundary so that between any two cells, robots will only need to move in a single direction when crossing a cell boundary. That is, for each cell boundary, iSAG is called to "cancel out" non-net robot movements, as illustrated in Fig. 13, leaving

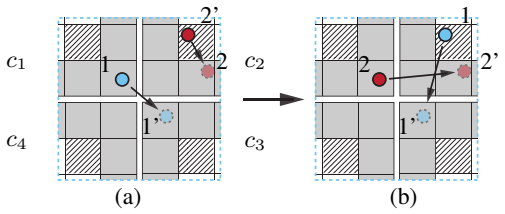


Fig. 12. (a) At the boundary between four cells, robot 1 has initial and goal configurations (vertices) spanning two diagonally adjacent cells. In the top right cell which is adjacent to both the top left and bottom right cells, there exists a robot that has its goal vertex in the same cell. (b) By swapping the robots 1 and 2 using iSAG, no robot needs to cross cell boundaries diagonally.

only uni-directional robot movements across cells. We call this operation *flow cancellation*.

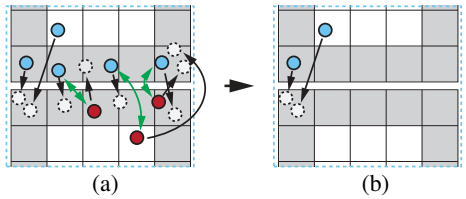


Fig. 13. (a) There are five robots in the top cell and three robots in the bottom cell that need to move across the horizontal boundary. (b) Through an arbitrary matching (indicated with double sided arrows) of three pairs of robots' initial configurations and applying iSAG to swap them, the robot movements across the boundary are now unidirectional.

The net robot movement across cell boundary induces a flow over the cells (see Fig. 14(a)). Because each cell contains a fixed number of robots, the incoming and outgoing flow at each cell (node) must be equal. This means that all such flows must form a valid *circulation*[1] over the graph formed by cells as nodes. The flow between two adjacent cells is no more than $6d_g^2$ (to be established later). The circulation can then be decomposed into $6d_g^2$ *unit circulations* (Fig. 14(b)). These unit circulations can be translated into coordinated *global* robot movements that require any robot to travel only *locally* at most a distance of $O(d_g)$. The translation amounts to creating another configuration X_G^3. (G, X_G^2, X_G^3) is also solved using iSAG.

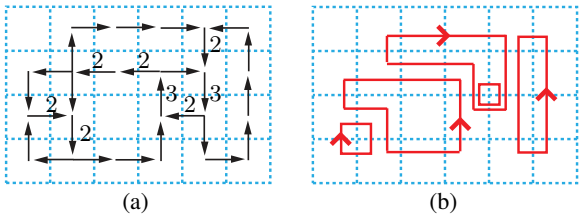


Fig. 14. (a) Induced circulation (network) from required robot movements. The numbers denote the total flow on a given edge. The edges without numbers have unit flows. (b) After decomposition, the circulation can be turned into unit circulations on simple cycles.

After the preparation phase, the scheduled global robot movements are directly executed, yielding a new configuration X_G^4, which has the property that every robot is now in the $5d_g \times 5d_g$ partitioned cell where its goal resides. iSAG can then be invoked to solve (G, X_G^4, X_G) at the cell level. Throughout, each robot only needs to move a distance of $O(d_g)$ and calls to iSAG can be performed in parallel, yielding an overall makespan of $O(d_g)$.

We now proceed to provide the proofs. Objects of minor importance, including the temporary configurations (e.g., X_G^i's) and actual robot movement plans (e.g., M^i's), are omitted in the description. Sufficient details are provided if a reader is interested in deriving these objects. The goal in the rest this section is to establish the following.

Theorem 7 (PAF, 2D)**.** *Let $p = (G, X_I, X_G)$ be an arbitrary MPP instance on an $m_1 \times m_2$ grid. A solution for p with $O(d_g(p))$ makespan can be computed in $O(m_1 m_2 d_g^2(p))$ time or $\tilde{O}(m_1 m_2 d_g(p))$ expected time.*

The cases for $d_g = o(m_1)$ are divided into two disjoint cases: *(i)* $d_g = \Omega(m_2)$ and *(ii)* $d_g = o(m_2)$, which is the main case. The first case can be readily addressed.

Lemma 8. *Let $p = (G, X_I, X_G)$ be an arbitrary MPP instance in which G is an $m_1 \times m_2$ grid with $d_g(p) = o(m_1)$ and $d_g(p) = \Omega(m_2)$. The instance admits a solution with a makespan of $O(d_g(p))$, computable in $O(m_1 m_2 d_g(p))$ time.*

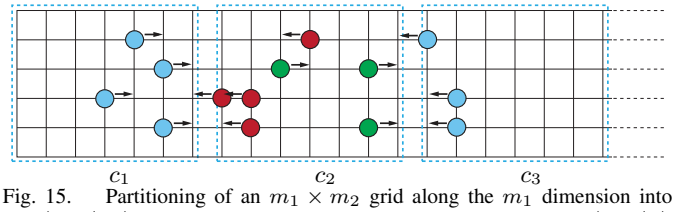


Fig. 15. Partitioning of an $m_1 \times m_2$ grid along the m_1 dimension into $q = \lfloor m_1/d_g \rfloor$ cells of roughly the same size of $w \times m_2$ with $w \approx \lfloor m_1/q \rfloor$. Three partitioned cells c_1, c_2 and c_3 are shown. Four robots need to move from c_1 to c_2 and three robots need to move from c_2 to c_3. Equal number of robots must move in the opposite direction. The goals of the robots are not illustrated in the drawing.

Proof: When $d_g = \Omega(m_2)$, We compute $q = \lfloor m_1/d_g \rfloor$ and $w = \lfloor m_1/q \rfloor$ (note that $w \geq d_g$). Partition G into q grid cells along the direction of m_1; each cell is of size $m_2 \times w$ or $m_2 \times (w+1)$ (see Fig. 15). We label these cells $c_1, \ldots, c_q$. By the definition of d_g, a robot initially located in cell c_i may only have its goal in either c_{i-1}, c_i, or c_{i+1}, when applicable. This further implies that for any applicable i, the number of robots must be equal between cells. The MPP instance can then be solved in two rounds through first invoking iSAG on the combined cells $c_i + c_{i+1}$ for all applicable odd i and then for all even i. The total makespan is clearly $O(d_g)$. For running time, each round of iSAG application requires $O(q(d_g^2 m_2 + d_g m_2^2)) = O(m_1 m_2 d_g)$ time. ■

The rest of this section is devoted to the case $d_g = o(m_2)$. Because $d_g = o(m_2)$, without loss of generality, we assume that $m_1 \geq m_2 \geq 5d_g$. Furthermore, we may assume without loss of generality that m_1 and m_2 are multiples of $5d_g$. If that is not the case, assuming that PAF is correct, then we can apply PAF up to four times without adding makespan or running time penalty. To execute this, first we compute $q_1 = \lfloor m_1/(5d_g) \rfloor$ and $q_2 = \lfloor m_2/(5d_g) \rfloor$. We note that $|V| \approx q_1 q_2 d_g^2$. Then, PAF is applied to the top left portion of G. This will fully solve the problem for the top left $(q_1 - 1) \times (q_2 - 1)$ cells of sizes $5d_g \times 5d_g$. Doing the same three more times with

[1]A circulation is essentially a valid flow over a network without source and sink nodes. That is, the incoming flows and outgoing flows at every node of the network are equal in magnitude.

each application on a different section of G, as illustrated in Fig. 16, the entire problem is then solved.

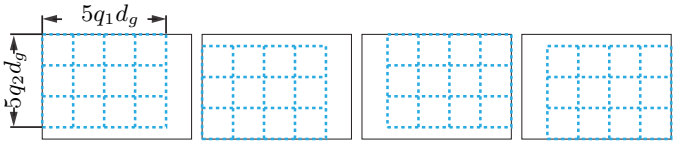


Fig. 16. For $G = m_1 \times m_2$, if m_1 or m_2 are not multiples of $5d_g$, we may apply PAF to a $q_1 \times q_2$ cell partition of G up to four times to cover G.

Henceforth, we assume $m_1 = 5q_1d_g$ and $m_2 = 5q_2d_g$ in which q_1 and q_2 are integers. G is partitioned into a $q_1 \times q_2$ *skeleton grid* G_S with its nodes being $5d_g \times 5d_g$ *cells*. We now realize what is illustrated in Fig. 14(a) from a raw partition using *diagonal rerouting* (Fig. 12) and *flow cancellation* (Fig. 13) operations.

Lemma 9 (Flow Orientation)**.** *In $O(m_1m_2d_g)$ time and $O(d_g)$ makespan, the flow of robots on the $q_1 \times q_2$ skeleton grid may be arranged to be only vertical or horizontal between adjacent cells and uni-directional. The largest total incoming flow through a cell boundary is no more than $6d_g^2$.*

Proof: We first show how to carry out diagonal rerouting (Fig. 12) and let the four involved cells, starting from the top-left one and in clockwise order, be c_1 through c_4. By the definition of d_g, if a robot 1 in c_1 has its goal in c_3, then the robot must be in the bottom right $d_g \times d_g$ region of c_1 and its goal must be in the top left $d_g \times d_g$ region of c_3. For each such robot, we pick an arbitrary robot 2 from c_2 in the diagonal-line shaded region. Any robot in this region will have its goal in c_2. By swapping the initial configurations of 1 and 2, the diagonal movement of 1 is eliminated. Going in a clockwise fashion, we apply the same procedure to all the cells to all affected robots. We may call iSAG in parallel on G on all these $4d_g \times 4d_g$ regions, which use $O(d_g)$ makespan. For running time, each $4d_g \times 4d_g$ region requires $(16d_g^2)^{\frac{3}{2}} = O(d_g^3)$ time, which is then $O(d_g m_1 m_2)$ over all q_1q_2 regions.

The flow cancellation operation (see Fig. 13) is carried out using a mechanism similar fashion and incurs similar makespan and running time; we omit the details. It is clear that the total flow through any boundary is no more than $5d_g^2 + d_g^2/2 + d_g^2/2 = 6d_g^2$. ∎

We are now left with only unidirectional flows on the skeleton grid G_S that are either vertical or horizontal between adjacent cells. To route these flows, closed disjoint cycles must be constructed for moving the robots synchronously across multiple cell boundaries. To achieve this, we will first decompose the flow into unit circulations (i.e., describing a procedure for going from Fig. 14(a) to Fig. 14(b)). Then, we will show how the cycles on the skeleton grid G_S can be grouped into a constant number of d_g^2 sized batches and turned into actual cycles on the original grid G. Our flow decomposition result, outlined below, works for arbitrary graphs.

Theorem 10 (Circulation Decomposition)**.** *Let $\mathcal{C}$ be a circulation on a graph $G = (V, E)$ with the largest total incoming flow for any vertex being f. $\mathcal{C}$ can be decomposed into f unit circulations on G in $O(f^2|V|)$ time or $(f|V|\log|V|)$ expected time.*

Proof: We proceed to build a bipartite graph over two copies of $|V|$. For a vertex $v_i \in V$, we denote one of the copy v_i^1 and the other v_i^2. For any two adjacent vertices $v_i, v_j \in V$, if there is a flow of magnitude f_{ij} from v_i to v_j, then we add f_{ij} edges between v_i^1 and v_j^2. Because the largest total incoming flow to any vertex is f, the maximum degree for any v_i^j, $j = 1, 2$, is also f. Also, due to flow conservation at vertices, for fixed $v_i \in V$, v_i^1 and v_i^2 have the same degree $f_i \le f$. For all v_i with $f_i < f$, we add $f - f_i$ edges between v_i^1 and v_i^2. This brings the degrees of all vertices in the bipartite graph to f, yielding a regular bipartite graph. The bipartite graph has $2|V|$ vertices and $f|V|$ edges. An illustration of the bipartite graph construction is given in Fig. 17.

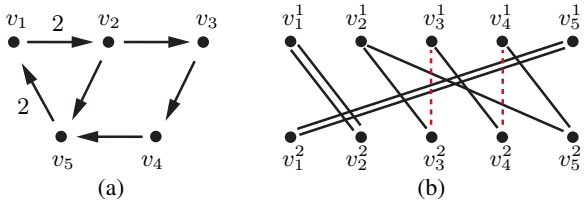


Fig. 17. (a) A graph with five vertices and a valid circulation of largest total incoming degree being 2. The flow on each edge with non-unit flow is marked on the edge. (b) The constructed bipartite graph. The dashed edges are the edges added to make the graph regular.

With the regular bipartite graph of degree f, we obtain perfect matching with [51] in $O(|E|) = O(f|V|)$ time. Each perfect matching corresponds to a unit circulation on G, which translates to either a single cycle or multiple vertex disjoint cycles. In the example, a perfect matching may be $(v_1^1, v_2^2), (v_2^1, v_5^2), (v_3^1, v_3^2), (v_4^1, v_4^2), (v_5^2, v_1^2)$, which translates to the cycle $v_1v_2v_5$. The total running time to obtain the f unit circulations is $O(f^2|V|)$. If we use the $O(|V|\log|V|)$ expected time algorithm from [53], the running time is then $O(f|V|\log|V|)$ expected time. ∎

For our setting, Theorem 10 implies $O(d_g^2)$ circulation on a $q_1 \times q_2$ skeleton grid can be decomposed into $O(d_g^2)$ unit circulations in $O(m_1m_2d_g^2)$ time or $\tilde{O}(m_1m_2)$ expected time. Because at most $6d_g^2$ flows can pass through a cell boundary, at most $12d_g^2$ flow can pass through a cell (two incoming, two outgoing). Theorem 10 gives us $12d_g^2$ unit circulations over the skeleton grid G_S. With the decomposed circulation, we may group them into batches and translate these into actual robot movements on G. To start, we handle a d_g batch.

Lemma 11 (Single Batch Global Flow Routing)**.** *A batch of up to d_g unit circulations on the $q_1 \times q_2$ skeleton grid may be translated into actual cyclic paths for robots on G to complete in a single step, using $O(m_1m_2)$ time.*

Proof: For a fixed cell, after considering various symmetries including rotation and flow directions, there are only three possible cases as illustrated in Fig. 18; note that some flow may have a value of zero. Therefore, establishing how a d_g amount of flow may be translated into feasible robot movements for the three cases in Fig. 18 encompasses all possible scenarios. We will sketch how up to d_g robots can be arranged to go through the boundaries in a single step for all three cases.

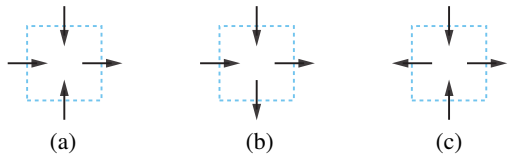


Fig. 18. Three possible flow orientations that cover all possible cases considering flow quantity (which may be zero) and symmetries (flipping of all flow directions and rotating the cell).

To route the robots, we will only use the center "+" area of d_g width of each $5d_g \times 5d_g$ cell. Fig. 19(a) illustrates the routing plan for realizing the flow given in Fig. 18(a), which may be readily verified to be correct using basic algebra, i.e., assuming the top, left, and bottom routes contain x, y, and z flows, respectively, such that $x + y + z \leq d_g$; we omit details. For arranging the robots, for horizontal cell boundaries, robots are aligned left. For vertical boundaries, robots are aligned toward the top. We note that if Fig. 18(a) is rotated, some adjustments are needed due to this choice of robot alignment but the change is minimal. Such alignments are necessary to ensure that the robot movements at cell boundaries match.

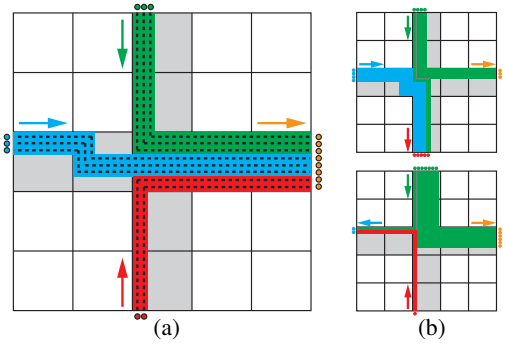


Fig. 19. Illustration of how a flow of size 8 may be translated to plans for robots for the case shown in Fig. 18. Each small square is of size $d_g \times d_g$.

We emphasize that the paths are constructed so that for the incoming and outgoing $d_g \times d_g$ boundary areas of the "+" that are involved, robots only move straight through it, which is not necessary but simplifies things when we put multiple batches together. For the cases from Fig. 18(b) and (c), illustrations of feasible routing plan construction are given in Fig. 19(b). Because all routing plans are parametrized, the main running time cost is to write done the plan, which takes $O(d_g^2)$ per cell. For $q_1 q_2$ cells, the total is $O(q_1 q_2 d_g^2) = O(m_1 m_2)$. ■

With a subroutine to push through G a batch of up to d_g unit circulations each step, d_g such batches may be further grouped for sequential execution, allowing the handling of up to d_g^2 at a time. This is established in the following lemma.

Lemma 12 (Multi-Batch Global Flow Routing)**.** *Up to d_g^2 unit circulations on the $q_1 \times q_2$ skeleton grid can be routed through G using $O(d_g)$ makespan and $O(m_1 m_2 d_g)$ time.*

Proof: For the proof, we only need to focus on a single $d_g \times d_g$ boundary area of a single cell; all other boundaries and cells will be handled similarly. Moreover, we only need to worry about robots moving out of a cell due to symmetry. With these reductions, we outline how to push up to d_g^2 robots out of the right boundary of of the "+" region of a cell, which is a $d_g \times d_g$ grid. Call this $d_g \times d_g$ grid c. After dicing up the d_g^2 circulations into d_g of d_g sized batches, we invoke Lemma 11 to generate feasible routing plans for each d_g sized batch. Because Lemma 11 guarantees that the generated paths are straight lines from left to right inside c, these batches can be sequentially arranged one after another, then compacted, for sequential routing out of the cell. An example for $d_g = 6$ is illustrated in Fig. 20. with each color representing a d_g sized batch to be moved out through the right in one step.

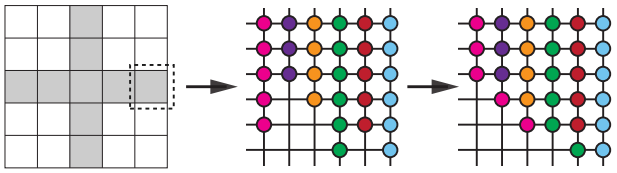


Fig. 20. (a) We are to route d_g^2 circulations through the right boundary of a cell in a $d_g \times d_g$ area, highlighted with the dashed square. (b) The plans generated for the d_g of d_g sized batches are arranged so that earlier plans appear on the right. For later plans, part of it get truncated. (c) The further compacted batches for actual execution. For robots that are not shown, they will stay in the cell and have no impact on the plan execution.

For computation time, for the d_g^2 circulation, we need to invoke the procedure from Lemma 11 for all $q_1 q_2$ cells d_g times, which incur a cost of $O(q_1 q_2 d_g^3) = O(m_1 m_2 d_g)$ running time, mostly used to write down the paths. To be able to actually prepare a cell for execution, iSAG must be invoked on the cell once, which takes $O(m_1 m_2 d_g)$ time over all cells. This is the dominating term. ■

Proof of Theorem 7: For $d_g = o(m_2)$, on a $q_1 \times q_2$ skeleton grid G_S of $5d_g \times 5d_g$ cells, we first apply Lemma 9 to ensure that flows of robots across cell boundaries are uni-direction without diagonal movements, in $O(m_1 m_2 d_g)$ time. Then, Theorem 10 computes a decomposition of the flow into up to $12d_g^2$ (vertex) unit circulations, in $O(m_1 m_2 d_g^2)$ time. Invoking Lemma 12 a constant number of times, in $O(m_1 m_2 d_g)$ time, we may globally route the robots so that all robots will be in the cell where its goal belongs to. We are then left with solving an MPP for each individual cell, which again requires $O(m_1 m_2 d_g)$ running time over all cells. Putting this together with the cases handled by Lemma 8, the overall running time is $O(m_1 m_2 d_g^2) \subset O(|V|^2)$ time, yielding a solution with $O(d_g)$ makespan. If we use randomized matching [53], the running time becomes $\tilde{O}(m_1 m_2 d_g)$ expected time. ■

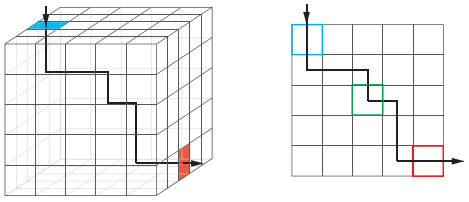
We note that a slightly better $O(m_1 m_2 d_g)$ running time is given in [2] due to a faster flow decomposition routine that explores planarity. Our routine (Theorem 10) does not use planarity and applies to arbitrary decompositions, which, together with iSAG, enables PAF to work in 3D.

V. PAF IN THREE DIMENSIONS

In 3D on an $m_1 \times m_2 \times m_3$ grid, we focus on the main case of $d_g = o(m_3)$ (recall that $m_1 \geq m_2 \geq m_3$); other cases can be readily addressed via applying PAF in 2D. For 3D, we will use a partition of cells of sizes $9d_g \times 9d_g \times 9d_g$ and assume that $9d_g$ divides m_i, i.e., $m_i = q_i 9d_g, 1 \leq i \leq 3$. It is straightforward to verify that PAF in 2D carries over except

it is not clear how to route d_g^3 flow through the faces of a $9d_g \times 9d_g \times 9d_g$ cell, which requires the routing of d_g^2 flow in a single step. Generating paths for routing robots corresponding to the flow is significantly more involved than in the 2D case. First, we match the up to six incoming and outgoing flows through a single cell so that at most one face sends flow to its opposite face. If there is a single pair of opposite faces with one having incoming flow and one having outgoing flow, nothing needs to be done. Otherwise, if there are multiple such face pairs, pick two arbitrary such pairs a_1, a_2, b_1, and b_2. Without loss of generality, assume flows $f_{a_1} > 0, f_{a_2} < 0, f_{b_1} > 0$, and $f_{b_2} < 0$ (this is similar to the case illustrated in Fig. 18(b)). If $f_{a_1} \leq |f_{b_2}|$, then we route all f_{a_1} flow into a_1 to go out through b_2, which then avoids the need for routing any flow into a_1 to go out from a_2. If $f_{a_1} > |f_{b_2}|$, we do the same, which means that no flow from b_1 needs to go out through b_2. Either way, we effectively get rid of a dimension i where $f_{i_1} * f_{i_2} < 0$. Doing this iteratively then leaves at most one such dimension where we may need to route any flow between the two opposite faces associated with that dimension.

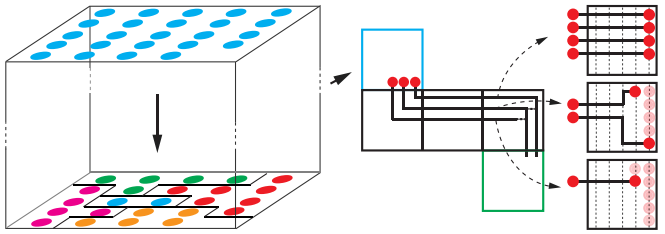
We now show how we may route flow from one face to other five faces through a $9d_g \times 9d_g \times 9d_g$ cell. Without loss of generality, we will show how to route flow coming in from the top face to the right face. Routing to the opposite face will be briefly explained afterward. We will route the flow to go through the center $d_g \times d_g$ regions on the six faces of the $9d_g \times 9d_g \times 9d_g$ cell and assume that a protocol is agreed on how the flow will be *shaped* between difference cells so the robot flows can be matched at cell boundaries. For example, on the top face, the d_g^2 flow may be ordered row by row (e.g., the 24 robots on the top of Fig. 22(a)), which result in a contiguous 2D shape inside a $d_g \times d_g$ region. Depending on the flow routing plan, this up to d_g^2 amount of flow is partitioned into 5 pieces (left, right, front, back, and center). We note that these pieces can again be made contiguous and in particular do not *interlock* with each other (bottom of Fig. 22(a)). Based on the partition, the proper amount of flow to each face is then *pivoted* to go sideways row by row (see left figure of Fig. 22(b)), except for flow that goes to the opposite face.



(a) (b)

Fig. 21. Illustration of how a certain amount of flow may be routed sideways. Only the top-right-middle $5d_g \times 5d_g \times 5d_g$ portion of the cell is shown in (a). (b) is a projective view from the front.

For the flow going to the right face, we rearrange them to a row-majored shape using a $2d_g \times d_g \times d_g$ grid, as illustrated in Fig. 22(b). At this point, we note that by symmetry, the same procedure can be applied to the flow going out of the right face in the reverse direction. Using a $d_g \times d_g \times d_g$ grid (the green one in Fig. 21(b) and Fig. 22(b)) as a buffer zone, these two separately crafted routes can be perfectly matched, completing the routing plan for a pair of faces. For routing flow to an opposite face, we simply let the flow to go down two more $d_g \times d_g \times d_g$ grids after going through the blue $d_g \times d_g \times d_g$ grid, after which we can do the same reshaping procedure. Once we can route d_g^2 flow through using a single step, we can do d_g batches of these, pushing d_g^3 flow in $O(d_g)$ makespan.



(a) (b)

Fig. 22. (a) Incoming d_g^2 flow may be broken into non-interlocking pieces going to difference faces. This $d_g \times d_g \times d_g$ grid corresponds to the cyan topped grid in Fig. 21(a).(b) A projective view (from the front) of how the three rows of red robots can be routed and reshaped into two row-major ordered rows, going downwards.

Our main goal so far is to show that it is feasible to route d_g^3 flow in $O(d_g)$ make span. To actually create the plan, we apply the max-flow algorithm (e.g., [54]) to an augmented direct graph generated on the $9d_g \times 9d_g \times 9d_g$ grid via vertex splitting, a standard technique used in finding *vertex disjoint paths*. We summarize the results in the following theorem.

Theorem 13 (PAF, 3D)**.** *Let $G = (V, E)$ be an $m_1 \times m_2 \times m_3$ grid and p be an arbitrary MPP instance on G. Then, a solution with $O(d_g(p))$ makespan can be computed in both $O(d_g^3(p)|V|)$ and $O(|V|^2)$ time.*

Proof: We only cover the case of $d_g = o(m_3)$. Following similar analysis as in the 2D case, the algorithm produces an $O(d_g)$ makespan solution. For running time, there are three main costs: *(i)* iSAG calls, *(ii)* matching for flow decomposition, and *(iii)* max-flow based global robot routing plan generation. For *(i)*, $q_1q_2q_3$ parallel calls to 3D iSAG is needed, demanding a time of $O(q_1q_2q_3d_g^4) = O(d_g|V|)$. For *(ii)*, flow decomposition is now performed on $O(d_g^3)$ flow on a graph with $O(q_1q_2q_3d_g^3)$ edges, requiring $O(d_g^3|V|)$ time. For *(iii)*, using Ford-Fulkerson [54], the total running time is $O(q_1q_2q_3d_g^6) = O(d_g^3|V|)$. ∎

VI. Discussion

In this work, having provided a complete description of iSAG for k-D and PAF for 2D and 3D, we comment that PAF extends to k-D using arguments similar to that used for 3D PAF (see [6]). Further extensions over k-D PAF can be readily made following [2], [55], which address lower robot densities and continuous setups.

Acknowledgments

This work is supported by NSF awards IIS-1617744 and IIS-1734419. The author would like to thank Pranjal Awasthi and Mario Szegedy for helpful discussions, and the anonymous reviewers for their constructive comments.

REFERENCES

[1] J. Yu, "Average case constant factor optimal multi-robot path planning in well-connected environments," *arXiv preprint arXiv:1706.07255*, 2017, note: A preliminary version appeared in the First International Symposium on Multi-Robot and Multi-Agent Systems, 2017.

[2] E. D. Demaine, S. P. Fekete, P. Keldenich, H. Meijer, and C. Scheffer, "Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch," *arXiv preprint arXiv:1801.01689*, 2018.

[3] 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.

[4] R. Stahlbock and S. Voß, "Operations research at container terminals: a literature update," *OR spectrum*, vol. 30, no. 1, pp. 1–52, 2008.

[5] S. Tang, J. Thomas, and V. Kumar, "Hold or take optimal plan (hoop): A quadratic programming approach to multi-robot trajectory generation," *The International Journal of Robotics Research*, p. 0278364917741532, 2018.

[6] J. Yu, "Constant-factor time-optimal multi-robot routing on high-dimensional grids in mostly sub-quadratic time," *arXiv preprint arXiv:1801.10465*, 2018.

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

[8] 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.

[9] 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.

[10] 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.

[11] R. Luna and K. E. Bekris, "Push and swap: Fast cooperative path-finding with completeness guarantees," in *Proceedings International Joint Conference on Artificial Intelligence*, 2011, pp. 294–300.

[12] T. Standley and R. Korf, "Complete algorithms for cooperative pathfinding problems," in *Proceedings International Joint Conference on Artificial Intelligence*, 2011, pp. 668–673.

[13] 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.

[14] K. Solovey and D. Halperin, "k-color multi-robot motion planning," in *Proceedings Workshop on Algorithmic Foundations of Robotics*, 2012.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[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. Nnaji, *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] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] P. Spirakis and C. K. Yap, "Strong NP-hardness of moving many discs," *Information Processing Letters*, vol. 19, no. 1, pp. 55–59, 1984.

[32] J. E. Hopcroft, J. T. Schwartz, and M. Sharir, "On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the "warehouseman's problem"," *The International Journal of Robotics Research*, vol. 3, no. 4, pp. 76–88, 1984.

[33] R. A. Hearn and E. D. Demaine, "PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation," *Theoretical Computer Science*, vol. 343, no. 1, pp. 72–96, 2005.

[34] K. Solovey and D. Halperin, "On the hardness of unlabeled multi-robot motion planning," in *Robotics: Science and Systems (RSS)*, 2015.

[35] K. Solovey, J. Yu, O. Zamir, and D. Halperin, "Motion planning for unlabeled discs with optimality guarantees," in *Robotics: Science and Systems*, 2015.

[36] 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.

[37] R. M. Wilson, "Graph puzzles, homotopy, and the alternating group," *Journal of Combinatorial Theory (B)*, vol. 16, pp. 86–96, 1974.

[38] V. Auletta, A. Monti, M. Parente, and P. Persiano, "A linear-time algorithm for the feasbility of pebble motion on trees," *Algorithmica*, vol. 23, pp. 223–245, 1999.

[39] G. Goraly and R. Hassin, "Multi-color pebble motion on graph," *Algorithmica*, vol. 58, pp. 610–636, 2010.

[40] J. Yu and D. Rus, "Pebble motion on graphs with rotations: Efficient feasibility tests and planning," in *Algorithmic Foundations of Robotics XI, Springer Tracts in Advanced Robotics*, vol. 107. Springer Berlin/Heidelberg, 2015, pp. 729–746.

[41] 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.

[42] 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.

[43] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *Proceedings IEEE/RSJ International Conference on Intelligent Robots & Systems*, 2011, pp. 3260–3267.

[44] C. Ferner, G. Wagner, and H. Choset, "Odrm* optimal multirobot path planning in low dimensional search spaces," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 3854–3859.

[45] 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.

[46] 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.
[47] W. Hönig, T. S. Kumar, L. Cohen, H. Ma, H. Xu, N. Ayanian, and S. Koenig, "Multi-agent path finding with kinematic constraints." in *ICAPS*, 2016, pp. 477–485.
[48] 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.
[49] 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.
[50] P. Hall, "On representatives of subsets," *Journal of the London Mathematical Society*, vol. 1, no. 1, pp. 26–30, 1935.
[51] R. Cole, K. Ost, and S. Schirra, "Edge-coloring bipartite multigraphs in o (e log d) time," *Combinatorica*, vol. 21, no. 1, pp. 5–12, 2001.
[52] M. Szegedy and J. Yu, "The n-Stacks Problem," 2017, working manuscript.
[53] A. Goel, M. Kapralov, and S. Khanna, "Perfect matchings in o(n\logn) time in regular bipartite graphs," *SIAM Journal on Computing*, vol. 42, no. 3, pp. 1392–1404, 2013.
[54] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian journal of Mathematics*, vol. 8, no. 3, pp. 399–404, 1956.
[55] S. D. Han, E. J. Rodriguez, and J. Yu, "SEAR: A Polynomial-Time Expected Constant-Factor Optimal Algorithmic Framework for Multi-Robot Path Planning," *arXiv:1709.08215*, 2017.