Journal of Artificial Intelligence Research 81 (2024) 443-479 Submitted 04/2024; published 10/2024

Expected $1.x$ Makespan-Optimal Multi-Agent Path Finding on Grid Graphs in Low Polynomial Time

Teng Guo TENG.GUO@RUTGERS.EDU
Jingjin Yu JINGJIN.YU@RUTGERS.EDU
Rutgers, the State University of New Jersey, Piscataway, NJ, USA.

Abstract

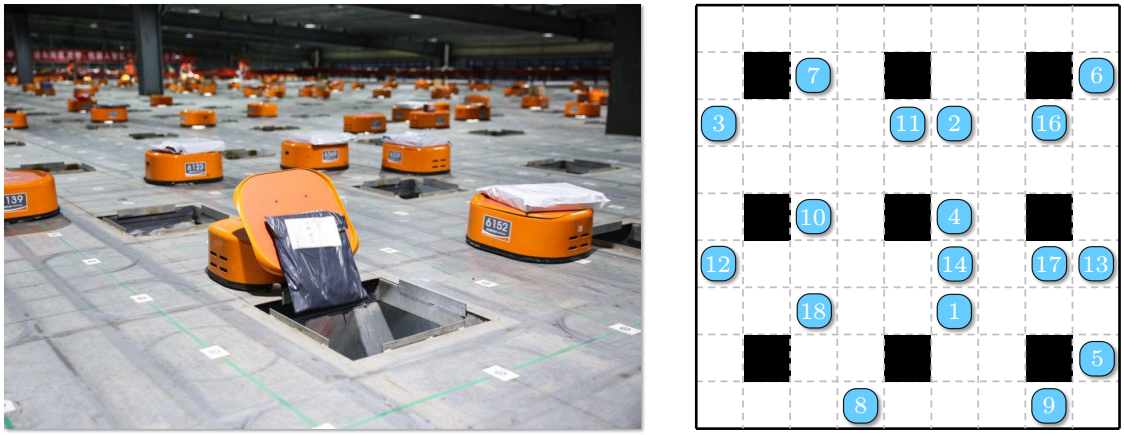
Multi-Agent Path Finding (MAPF) is NP-hard to solve optimally, even on graphs, suggesting no polynomial-time algorithms can compute exact optimal solutions for them. This raises a natural question: How optimal can polynomial-time algorithms reach? Whereas algorithms for computing constant-factor optimal solutions have been developed, the constant factor is generally very large, limiting their application potential. In this work, among other breakthroughs, we propose the first low-polynomial-time MAPF algorithms delivering 1-1.5 (resp., 1-1.67) asymptotic makespan optimality guarantees for 2D (resp., 3D) grids for random instances at a very high 1/3 agent density, with high probability. Moreover, when regularly distributed obstacles are introduced, our methods experience no performance degradation. These methods generalize to support 100% agent density.

Regardless of the dimensionality and density, our high-quality methods are enabled by a unique hierarchical integration of two key building blocks. At the higher level, we apply the labeled Grid Rearrangement Algorithm (GRA), capable of performing efficient reconfiguration on grids through row/column shuffles. At the lower level, we devise novel methods that efficiently simulate row/column shuffles returned by GRA. Our implementations of GRA-based algorithms are highly effective in extensive numerical evaluations, demonstrating excellent scalability compared to other SOTA methods. For example, in 3D settings, GRA-based algorithms readily scale to grids with over $370,000$ vertices and over $120,000$ agents and consistently achieve conservative makespan optimality approaching 1.5, as predicted by our theoretical analysis.

1. Introduction

We examine multi-agent pathfinding (MAPF (Stern, et al., 2019)) on two- and three-dimensional grids with potentially regularly distributed obstacles (see Fig. 1). The main objective of MAPF is to find collision-free paths for routing many agents from a start configuration to a goal configuration. In practice, solution optimality is of key importance, yet optimally solving MAPF is NP-hard (Surynek, 2010; Yu & LaValle, 2013b), even in planar settings (Yu, 2015) and on obstacle-less grids (Demaine, et al., 2019). MAPF algorithms apply to a diverse set of practical scenarios, including formation reconfiguration (Poduri & Sukhatme, 2004), object transportation (Rus, Donald, & Jennings, 1995), swarm robotics (Preiss, et al., 2017; Hönig, et al., 2018), to list a few. Even when constrained to grid-like settings, MAPF algorithms still find many large-scale applications, including warehouse automation for general order fulfillment (Wurman, D'Andrea, & Mountz, 2008), grocery order fulfillment (Mason, 2019), and parcel sorting (Wan, et al., 2018).

Motivated by applications including order fulfillment and parcel sorting, we focus on grid-like settings with extremely high agent density, i.e., with $\frac{1}{3}$ or more graph vertices



(a) (b)

Figure 1: (a) Real-world parcel sorting system (at JD.com) using many agents on a large grid-like environment with holes for dropping parcels; (b) A snapshot of a similar MAPF instance our methods can solve in polynomial-time with provable optimality guarantees. In practice, our algorithms scale to 2D maps of size over 450×300, supporting over 45K agents and achieving better than 1.5-optimality (see, e.g., Fig. 14). They scale further in 3D settings.

occupied by agents. Whereas recent studies (Yu, 2018; Demaine et al., 2019) show such problems can be solved in polynomial time yielding constant-factor optimal solutions, the constant factor is prohibitively high ($\gg 1$) to be practical. In this research, we tear down this MAPF planning barrier, achieving $(1+\delta)$-makespan optimality asymptotically with high probability in polynomial time where $\delta \in (0, 0.5]$ for 2D grids and $\delta \in (0, \frac{2}{3}]$ for 3D grids.

This study's primary theoretical and algorithmic contributions are outlined below and summarized in Table 1. Through judicious applications of a novel row/column-shuffle-based global Grid Rearrangement (GR) method called the Rubik Table Algorithm (Szegedy & Yu, 2023)[1], employing many classical algorithmic techniques, and combined with careful analysis, we establish that in *low polynomial* time:

- For $m_1 m_2$ agents on a 2D $m_1 \times m_2$ grid, $m_1 \geq m_2$, i.e., at maximum agent density, GRM (Grid Rearrangement for MAPF) computes a solution for an arbitrary MAPF instance under a makespan of $4m_1 + 8m_2$; For $m_1 m_2 m_3$ agents on a 3D $m_1 \times m_2 \times m_3$ grids, $m_1 \geq m_2 \geq m_3$, GRM computes a solution under a makespan of $4m_1 + 8m_2 + 8m_3$;
- For $\frac{1}{3}$ agent density with uniformly randomly distributed start/goal configurations, GRH (Grid Rearrangement with Highways) computes a solution with a makespan of $m_1 + 2m_2 + o(m_1)$ on 2D grids and $m_1 + 2m_2 + 2m_3 + o(m_1)$ on 3D grids, with high probability. In contrast, such an instance has a minimum makespan of $m_1 + m_2 - o(m_1)$ for 2D grids and $m_1 + m_2 + m_3 - o(m_1)$ for 3D grids with high probability. This implies that, as

1. It was noted in (Szegedy & Yu, 2023) that their Rubik Table Algorithm can be applied to solve MAPF; we do not claim this as a contribution of our work. Rather, our contribution is to dramatically bring down the constant factor through a combination of meticulous algorithm design and careful analysis.

$m_1 \to \infty$, an asymptotic optimality guarantee of $\frac{m_1+2m_2}{m_1+m_2} \in (1, 1.5]$ is achieved for 2D grids and $\frac{m_1+2m_2+2m_3}{m_1+m_2+m_3} \in (1, \frac{5}{3}]$ for 3D grids, with high probability;

- For $\frac{1}{2}$ agent density, the same optimality guarantees as in the $\frac{1}{3}$ density setting can be achieved using GRLM (Grid Rearrangements with Line Merge), using a merge-sort inspired agent routing heuristic;
- Without modification, GRH achieves the same $\frac{m_1+2m_2}{m_1+m_2}$ optimality guarantee on 2D grids with up to $\frac{m_1m_2}{9}$ regularly distributed obstacles together with $\frac{2m_1m_2}{9}$ agents (e.g., Fig. 1(b)). Similar guarantees hold for 3D settings;
- For agent density up to $\frac{1}{2}$, for an arbitrary (i.e., not necessarily random) instance, a solution can be computed with a makespan of $3m_1 + 4m_2 + o(m_1)$ on 2D grids and $3m_1 + 4m_2 + 4m_3 + o(m_1)$ on 3D grids.
- With minor numerical updates to the relevant guarantees, the above-mentioned results also generalize to grids of arbitrary dimension $k \geq 4$.

Moreover, we have developed many effective and principled heuristics that work together with the GRA-based algorithms to further reduce the computed makespan by a significant margin. These heuristics include (1) An optimized matching scheme, required in the application of the grid rearrangement algorithm, based on linear bottleneck assignment (LBA), (2) A polynomial time path refinement method for compacting the solution paths to improve their quality. As demonstrated through extensive evaluations, our methods are highly scalable and capable of tackling instances with tens of thousands of densely packed agents. Simultaneously, the achieved optimality matches/exceeds our theoretical prediction. With the much-enhanced scalability, our approach unveils a promising direction toward the development of practical, provably optimal multi-agent routing algorithms that run in low polynomial time.

Algorithms	GRM 2x3	GRM 4x2	GRLM	GRH
Applicable Density	≤ 1	≤ 1	$\leq \frac{1}{2}$	$\leq \frac{1}{3}$
Makespan Upperbound	$7(2m_2 + m_1)$	$4(2m_2 + m_1)$	$4m_2 + 3m_1$	$4m_2 + 3m_1$
Asymptotic Makespan	$7(2m_2 + m_1)$	$4(2m_2 + m_1)$	$2m_2 + m_1 + o(m_1)$	$2m_2 + m_1 + o(m_1)$
Asymptotic Optimality	$7(1 + \frac{m_2}{m_1+m_2})$	$4(1 + \frac{m_2}{m_1+m_2})$	$1 + \frac{m_2}{m_1+m_2}$	$1 + \frac{m_2}{m_1+m_2}$

Table 1: Summary results of the algorithms proposed in this work. All algorithms operate on grids of dimensions $m_1 \times m_2$. GRH, GRM, and GRLM are all further derived from GRA2D, each a variant characterized by distinct low-level shuffle movements. Specifically for GRM, the low-level shuffle movement is tailored for facilitating full-density robot movement

This paper builds on two conference publications (Guo & Yu, 2022; Guo, Feng, & Yu, 2022). Besides providing a unified treatment of the problem that streamlined the description of GRA-based algorithms under 2D/3D/kD settings for journal archiving, the manuscript further introduces many new results, including (1) The baseline GRM method (for the full-density case) is significantly improved with a much stronger makespan optimality guarantee, by a factor of $\frac{7}{4}$; (2) A new and general polynomial-time path refinement technique is

developed that significantly boosts the optimality of the plans generated by all GRA-based algorithms; (3) Complete and substantially refined proofs are provided for all theoretical developments in the paper; and (4) The evaluation section is fully revamped to reflect the updated theoretical and algorithmic development.

Related work. Literature on multi-agent path and motion planning (Hopcroft, Schwartz, & Sharir, 1984; Erdmann & Lozano-Perez, 1987) is expansive; here, we mainly focus on graph-theoretic (i.e., the state space is discrete) studies (Yu & LaValle, 2016; Stern et al., 2019). As such, in this paper, MAPF refers explicitly to graph-based multi-agent path planning. Whereas the feasibility question has long been positively answered (Kornhauser, Miller, & Spirakis, 1984), the same cannot be said for finding optimal solutions, as computing time- or distance-optimal solutions are NP-hard in many settings, including on general graphs (Goldreich, 2011; Surynek, 2010; Yu & LaValle, 2013b), planar graphs (Yu, 2015; Banfi, Basilico, & Amigoni, 2017), and regular grids (Demaine et al., 2019) that is similar to the setting studied in this work.

Nevertheless, given its high utility, especially in e-commerce applications (Wurman et al., 2008; Mason, 2019; Wan et al., 2018) that are expected to grow significantly (Dekhne, et al., 2019; LogisticsIQ, 2020), many algorithmic solutions have been proposed for optimally solving MAPF. Among these, combinatorial-search-based solvers (Lam, et al., 2019) have been demonstrated to be fairly effective. MAPF solvers may be classified as being optimal or suboptimal. Reduction-based optimal solvers solve the problem by reducing the MAPF problem to another problem, e.g., SAT (Surynek, 2012), answer set programming (Erdem, et al., 2013), integer linear programming (ILP) (Yu & LaValle, 2016). Search-based optimal MAPF solvers include EPEA* (Goldenberg, et al., 2014), ICTS (Sharon, et al., 2013), CBS (Sharon, et al., 2015), M* (Wagner & Choset, 2015), and many others.

Due to the inherent intractability of optimal MAPF, optimal solvers usually exhibit limited scalability, leading to considerable interest in suboptimal solvers. Unbounded solvers like push-and-swap (Luna & Bekris, 2011), push-and-rotate (De Wilde, Ter Mors, & Witteveen, 2014), windowed hierarchical cooperative A* (Silver, 2005), BIBOX (Surynek, 2009), all return feasible solutions very quickly, but at the cost of solution quality. Balancing the running time and optimality is one of the most attractive topics in the study of MAPF. Some algorithms emphasize the scalability without sacrificing as much optimality, e.g., ECBS (Barer, et al., 2014), DDM (Han & Yu, 2020), PIBT (Okumura, et al., 2019), PBS (Ma, et al., 2019). There are also learning-based solvers (Damani, et al., 2021; Sartoretti, et al., 2019; Li, et al., 2021) that scale well in sparse environments. Effective orthogonal heuristics have also been proposed (Guo, Han, & Yu, 2021). Recently, $O(1)$-approximate or constant factor time-optimal algorithms have been proposed, e.g. (Yu, 2018; Demaine et al., 2019; Han, Rodriguez, & Yu, 2018), that tackle highly dense instances. However, these algorithms only achieve a low-polynomial time guarantee at the expense of very large constant factors, rendering them theoretically interesting but impractical.

In contrast, with high probability, our methods run in low polynomial time with provable $1.x$ asymptotic optimality. To our knowledge, this paper presents the first MAPF algorithms to simultaneously guarantee polynomial running time and $1.x$ solution optimality, which works for any dimension ≥ 2.

Organization. The rest of the paper is organized as follows. In Sec. 2, starting with 2D grids, we provide a formal definition of graph-based MAPF, and introduce the Grid

Rearrangement problem and the associated baseline algorithm (GRA) for solving it. GRM, a basic adaptation of GRA for MAPF at maximum agent density which ensures a makespan upper bound of $4m_1 + 8m_2$, is described in Sec. 3. An accompanying lower bound of $m_1 + m_2 - o(m_1)$ for random MAPF instances is also established. In Sec. 4 we introduce GRH for $\frac{1}{3}$ agent density achieving a makespan upper bound of $m_1 + 2m_2 + o(m_1)$. Obstacle support is then discussed. We continue to show how $\frac{1}{2}$ agent density may be supported with similar optimality guarantees. In Sec. 5, we generalize the algorithms to work on 3+D grids. In Sec. 6, we introduce multiple optimality-boosting heuristics to significantly improve the solution quality for all variants of Grid Rearrangement-based solvers. We thoroughly evaluate the performance of our methods in Sec. 7 and conclude with Sec. 8.

2. Preliminaries

2.1 Multi-Agent Path Finding on Graphs (MAPF)

Consider an undirected graph $\mathcal{G}(V, E)$ and n agents with start configuration $S = \{s_1, \ldots, s_n\} \subseteq V$ and goal configuration $G = \{g_1, \ldots, g_n\} \subseteq V$. A *path* for agent i is a map $P_i : \mathbb{N} \to V$ where $\mathbb{N}$ is the set of non-negative integers. A feasible P_i must be a sequence of vertices that connects s_i and g_i: 1) $P_i(0) = s_i$; 2) $\exists T_i \in \mathbb{N}$, s.t. $\forall t \geq T_i, P_i(t) = g_i$; 3) $\forall t > 0$, $P_i(t) = P_i(t-1)$ or $(P_i(t), P_i(t-1)) \in E$. A path set $\{P_1, \ldots, P_n\}$ is feasible iff each P_i is feasible and for all $t \geq 0$ and $1 \leq i < j \leq n$, $P_i(t) = P$, it does not happen that: 1) $P_i(t) = P_j(t)$; 2) $P_i(t) = P_j(t+1) \wedge P_j(t) = P_i(t+1)$.

We work with $\mathcal{G}$ being 4-connected grids in 2D and 6-connected grids in 3D, aiming to mainly minimize the *makespan*, i.e., $\max_i\{|P_i|\}$ (later, a sum-of-cost objective is also briefly examined). Unless stated otherwise, $\mathcal{G}$ is assumed to be an $m_1 \times m_2$ grid with $m_1 \geq m_2$ in 2D and $m_1 \times m_2 \times m_3$ grid with $m_1 \geq m_2 \geq m_3$ in 3D. As a note, "randomness" in this paper always refers to uniform randomness. The version of MAPF we study is sometimes referred to as *one-shot* MAPF (Stern et al., 2019). We mention our results also translate to guarantees on the life-long setting (Stern et al., 2019), briefly discussed in Sec. 8.

2.2 The Grid Rearrangement Problem (GRP)

The Grid Rearrangement problem (GRP) (first proposed in (Szegedy & Yu, 2023) as the Rubik Table problem) formalizes the task of carrying out globally coordinated object reconfiguration operations on lattices, with many interesting applications. The problem has many variations; we start with the 2D form, to be generalized to higher dimensions later.

Problem 1 (**Grid Rearrangement Problem in 2D (GRP2D)** (Szegedy & Yu, 2023))**.** *Let M be an $m_1(row) \times m_2(column)$ table, $m_1 \geq m_2$, containing m_1m_2 items, one in each table cell. The items have m_2 colors with each color having a multiplicity of m_1. In a* shuffle *operation, the items in a single column or a single row of M may be permuted in an arbitrary manner. Given an arbitrary configuration X_I of the items, find a sequence of shuffles that take M from X_I to the configuration where row i, $1 \leq i \leq m_1$, contains only items of color i. The problem may also be* labeled*, i.e., each item has a unique label in $1, \ldots, m_1m_2$.*

A key result (Szegedy & Yu, 2023), which we denote here as the (labeled) Grid Rearrangement Algorithm in 2D (GRA2D), shows that a colored GRP2D can be solved using m_2 column shuffles followed by m_1 row shuffles, implying a low-polynomial time computation

time. Additional m_1 row shuffles then solve the labeled GRP2D. We illustrate how GRA2D works on an $m_1 \times m_2$ table with $m_1 = 4$ and $m_2 = 3$ (Fig. 2); we refer the readers to (Szegedy & Yu, 2023) for details. The main supporting theoretical result is given in Theorem 2.2, which depends on Theorem 2.1.

Theorem 2.1 (Hall's Matching theorem with parallel edges (Hall, 2009; Szegedy & Yu, 2023))**.** *Let B be a d-regular ($d > 0$) bipartite graph on $n + n$ nodes, possibly with parallel edges. Then B has a perfect matching.*

Theorem 2.2 (Grid Rearrangement Theorem (Szegedy & Yu, 2023)). *An arbitrary Grid Rearrangement problem on an $m_1 \times m_2$ table can be solved using $m_1 + m_2$ shuffles. The labeled Grid Rearrangement problem can be solved using $2m_1 + m_2$ shuffles.*

GRA2D operates in two phases. In the first, a bipartite graph $B(T, R)$ is constructed based on the initial table where the bipartite set T are the colors/types of items, and the set R the rows of the table (see Fig. 2(b)). An edge is added to B between $t \in T$ and $r \in R$ for every item of color t in row r. From $B(T, R)$, which is a *regular bipartite graph*, m_2 *perfect matchings* can be computed as guaranteed by Theorem 2.1. Each matching, containing m_1 edges, dictates how a table column should look like after the first phase. For example, the first set of matching (solid lines in Fig. 2(b)) says the first column should be ordered as yellow, cyan, red, and green, shown in Fig. 2(c). After all matchings are processed, we get an intermediate table, Fig. 2(c). Notice each row of Fig. 2(a) can be shuffled to yield the corresponding row of Fig. 2(c); a key novelty of GRA2D. After the first phase of m_1 row shuffles, the intermediate table (Fig. 2(c)) can be rearranged with m_2 column shuffles to solve the colored GRP2D (Fig. 2(d)). Another m_1 row shuffles then solve the labeled GRP2D (Fig. 2(e)). It is also possible to perform the rearrangement using m_2 column shuffles, followed by m_1 row shuffles, followed by another m_2 column shuffles.

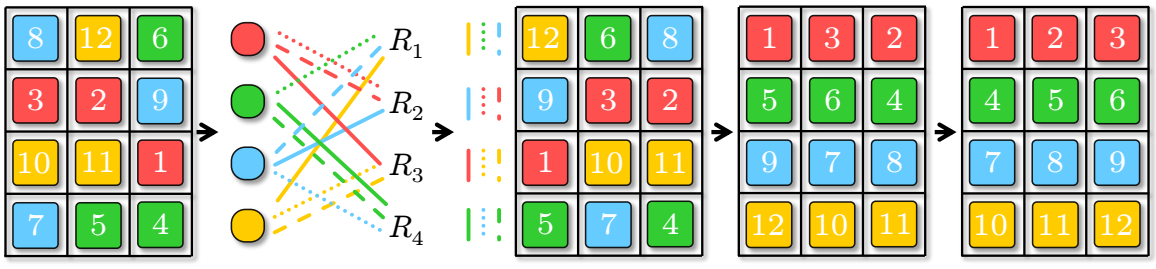


Figure 2: Illustration of applying the 11 *shuffles*. (a) The initial 4×3 table with a random arrangement of 12 items that are colored and labeled. The labels are consistent with the colors. (b) The constructed bipartite graph. It contains 3 perfect matchings, determining the 3 columns in (c); only color matters in this phase. (c) Applying 4 row shuffles to (a), according to the matching results, leads to an intermediate table where each column has one color appearing exactly once. (d) Applying 3 column shuffles to (c) solves a colored GRP2D. (e) 4 additional row shuffles fully sort the labeled items.

GRA2D runs in $O(m_1 m_2 \log m_1)$ (notice that this is nearly linear with respect to $n = m_1 m_2$, the total number of items) expected time or $O(m_1^2 m_2)$ deterministic time.

3. Solving MAPF at Maximum Density Leveraging GRA2D

The built-in global coordination capability of GRA2D naturally applies to solving makespan-optimal MAPF. Since GRA2D only requires *three rounds* of shuffles and each round involves either parallel row shuffles or parallel column shuffles, if each round of shuffles can be realized with makespan proportional to the size of the row/column, then a makespan upper bound of $O(m_1 + m_2)$ can be guaranteed. This is in fact achievable even when all of $\mathcal{G}$'s vertices are occupied by agents, by recursively applying a *labeled line shuffle algorithm* (Yu, 2018), which can arbitrarily rearrange a line of m agents embedded in a grid using $O(m)$ makespan.

Lemma 3.1 (Basic Simulated Labeled Line Shuffle (Yu, 2018))**.** *For m labeled agents on a straight path of length m, embedded in a 2D grid, they may be arbitrarily ordered in $O(m)$ steps. Moreover, multiple such reconfigurations can be performed on parallel paths within the grid.*

The key operation is based on a localized, 3-step pair swapping routine, shown in Fig. 3. For more details on the line shuffle routine, see (Yu, 2018).

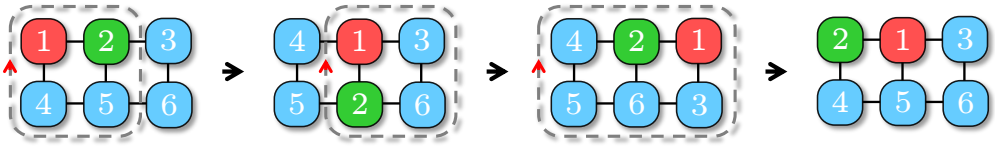


Figure 3: On a 2×3 grid, swapping two agents may be performed in three steps with three cyclic rotations.

However, the basic simulated labeled line-shuffle algorithm has a large constant factor. Each shuffle takes 3 steps; doing arbitrary shuffling of a 2×3 takes 20+ steps in general. The constant factor further compounds as we coordinate the shuffles across multiple lines. Borrowing ideas from *parallel odd-even sort* (Bitton, et al., 1984), we can greatly reduce the constant factor in Lemma 3.1. We will do this in several steps. First, we need the following lemma. By an *arbitrary horizontal swap* on a grid, we mean an arbitrary reconfiguration or a grid row.

Lemma 3.2. *It takes at most* 7, 6, 6, 7, 6, *and* 8 *steps to perform arbitrary combinations of arbitrary horizontal swaps on* 3×2, 4×2, 2×3, 3×3, 2×4, *and* 3×4 *grids, respectively.*

Proof. Using integer linear programming (Yu & LaValle, 2016), we exhaustively compute makespan-optimal solutions for arbitrary horizontal reconfigurations on 3×2 ($8 = 2^3$ possible cases), 4×2 grids (2^4 possible cases), 2×3 grids (6^2 possible cases), 3×3 grids (6^3 possible cases), 2×4 grids (24^2 possible cases), and 3×4 grids (24^3 possible cases), which confirms the claim. □

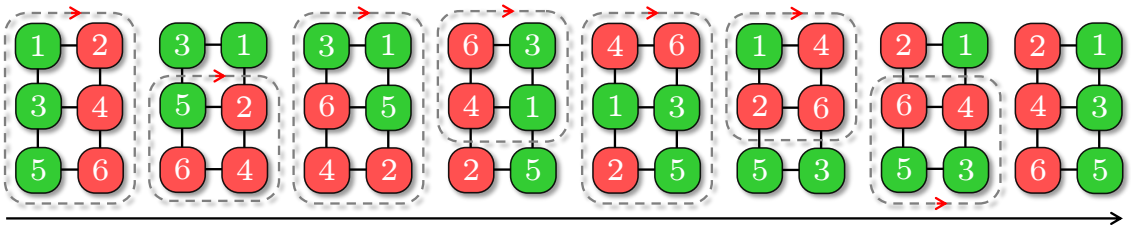


Figure 4: An example of a full horizontal "swap" on a 3×2 grid that takes seven steps, in which all three pairs are swapped.

As an example, it takes seven steps to horizontally "swap" all three pairs of agents on a 3×2 grid, as shown in Fig. 4.

Lemma 3.3 (Fast Line Shuffle)**.** *Embedded in a 2D grid, m agents on a straight path of length m may be arbitrarily ordered in $7m$ steps. Moreover, multiple such reconfigurations can be executed in parallel within the grid.*

Proof. Arranging m agents on a straight path of length m may be realized using parallel odd-even sort (Bitton et al., 1984) in $m-1$ rounds, which only requires the ability to simulate potential pairwise "swaps" interleaving odd phases (swapping agents located at positions $2k+1$ and $2k+2$ on the path for some k) and even phases (swapping agents located at positions $2k+2$ and $2k+3$ on the path for some k). Here, it does not matter whether m is odd or even. To simulate these swaps, we can partition the grid embedding the path into 3×2 grids in two ways for the two phases, as illustrated in Fig. 5.

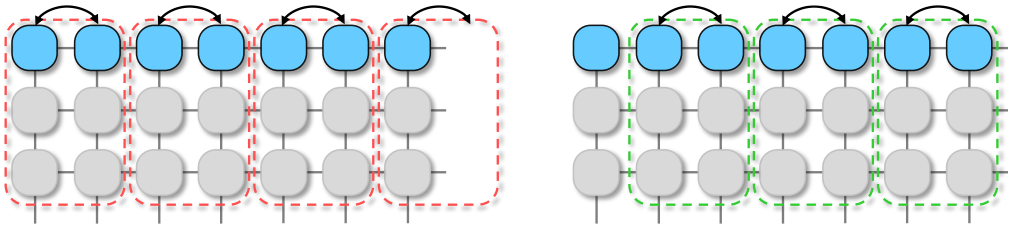


Figure 5: Partitioning a grid into disjoint 3×2 grids in two ways for simulating odd-even sort. Highlighted agent pairs may be independently "swapped" within each 3×2 grid as needed.

A perfect partition requires that the second dimension of the grid, perpendicular to the straight path, be a multiple of 3. If not, some partitions at the bottom can use 4×2 grids. By Lemma 3.2, each odd-even sorting phase can be simulated using at most $7m$ steps. Shuffling on parallel paths is directly supported. □

After introducing a $7m$ step line shuffle, we further show how it can be dropped to $4m$, using similar ideas. The difference is that an updated parallel odd-even sort will be used with different sub-grids of sizes different from 2×3 and 2×4.

The updated parallel odd-even sort operates on blocks of *four* (Fig. 6) instead of *two* (Fig. 5), which cuts down the number of parallel sorting operations from $m-1$ to about $m/2$. Here, if m is not even, a partition will leave either 1 or 3 at the end. For example, if $m=11$, it can be partitioned as $4,4,3$ and $2,4,4,1$ in the two parallel sorting phases.

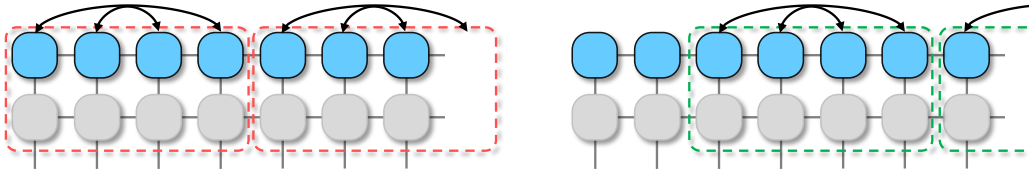


Figure 6: We can make the parallel odd-even sort work twice faster by increasing the swap block size from two to four.

With the updated parallel odd-even sort, we must be able to make swaps on blocks of four. We do this by partition an $m_1 \times m_2$ grid into 2×4, which may have leftover sub-grids of sizes 2×3, 3×4, and 3×3. Using the same reasoning in proving Lemma 3.3 and with Lemma 3.2, we have

Lemma 3.4 (Faster Line Shuffle)**.** *Embedded in a 2D grid, m agents on a straight path of length m may be arbitrarily ordered in approximately $4m$ steps. Moreover, multiple such reconfigurations can be executed in parallel within the grid.*

Proof sketch. The updated parallel odd-even sort requires a total of $m/2$ steps. Since each step operated on a 2×4, 2×3, 3×4, or 3×3 grid, which takes at most 8 steps, the total makespan is approximately $4m$. □

Combining GRA2D and fast line shuffle (Lemma 3.4) yields a polynomial time MAPF algorithm for fully occupied grids with a makespan of $4m_1 + 8m_2$.

Theorem 3.1 (MAPF on Grids under Maximum Agent Density, Upper Bound)**.** MAPF *on an $m_1 \times m_2$ grid, $m_1 \geq m_2 \geq 3$, with each grid vertex occupied by an agent, can be solved in polynomial time in a makespan of $4m_1 + 8m_2$.*

Proof Sketch. By combining the GRA2D and the line shuffle algorithms, the problem can be efficiently solved through two row-shuffle phases and one column-shuffle phase. During the row-shuffle phases, all rows can be shuffled in $4m_2$ steps, and similarly, during column-shuffle phases, all columns can be shuffled in $4m_1$ steps. Summing up these steps, the entire problem can be addressed in $4m_1 + 8m_2$ steps. The primary computational load lies in computing the perfect matchings, a task achievable in $O(m_1 m_2 \log m_1)$ expected time or $O(m_1^2 m_2)$ deterministic time. □

It is clear that, by exploiting the idea further, smaller makespans can potentially be obtained for the full-density setting, but the computation required for extending Lemma 3.2 will become more challenging. It took about two days to compute all 24^3 cases for the 3×4 grid, for example.

We call the resulting algorithm from the process GRM (for 2D), with "M" denoting *maximum density*, regardless of the used sub-grids. The straightforward pseudo-code is given in Alg. 1. The comments in the main GRM routine indicate the corresponding GRA2D phases. For MAPF on an $m_1 \times m_2$ grid with row-column coordinates (x, y), we say agent i belongs to color $1 \leq j \leq m_1$ if $g_i.y = j$. Function `Prepare()` in the first phase finds intermediate states $\{\tau_i\}$ for each agent through perfect matchings and routes them towards the intermediate states by (simulated) column shuffles. If the agent density is smaller than required, we may fill the table with "virtual" agents (Han et al., 2018; Yu, 2018). For each agent i, we have $\tau_i.y = s_i.y$. Function `ColumnFitting()` in the second phase routes the agents to their second intermediate states $\{\mu_i\}$ through row shuffles where $\mu_i.x = \tau_i.x$ and $\mu_i.y = g_i.y$. In the last phase, function `RowFitting()` routes the agents to their final goal positions using additional column shuffles.

Algorithm 1: Labeled Grid Rearrangement Based MAPF Algorithm for 2D (GRA2D)

Input: Start and goal vertices $S = \{s_i\}$ and $G = \{g_i\}$
Function RTA2D(S, G)**:**
  `Prepare`(S, G) ▷ Computing Fig. 2(b)
  `ColumnFitting`(S, G) ▷ Fig. 2(a) → Fig. 2(c)
  `RowFitting`(S, G) ▷ Fig. 2(c) → Fig. 2(d)
Function `Prepare`(S, G)**:**
  $A \leftarrow [1, ..., m_1 m_2]$
  for $(t, r) \in [1, ..., m_1] \times [1, ..., m_1]$ **do**
    if $\exists i \in A$ *where* $s_i.x = r \wedge g_i.y = t$ **then**
      add edge (t, r) to $B(T, R)$
      remove i from A
  compute matchings $\mathcal{M}_1, ..., \mathcal{M}_{m_2}$ of $B(T, R)$
  $A \leftarrow [1, ..., m_1 m_2]$
  foreach $\mathcal{M}_r$ *and* $(t, r) \in \mathcal{M}_r$ **do**
    if $\exists i \in A$ *where* $s_i.x = r \wedge g_i.y = t$ **then**
      $\tau_i \leftarrow (r, s_i.y)$ and remove i from A
      mark agent i to go to τ_i
  perform simulated column shuffles in parallel
Function `ColumnFitting`(S, G)**:**
  foreach $i \in [1, ..., m_1 m_2]$ **do**
    $\mu_i \leftarrow (\tau_i.x, g_i.y)$ and mark agent i to go to μ_i
  perform simulated row shuffles in parallel
Function `RowFitting`(S, G)**:**
  foreach $i \in [1, ..., m_1 m_2]$ **do**
    mark agent i to go to g_i
  perform simulated column shuffles in parallel

We now establish the optimality guarantee of GRM on 2D grids, assuming MAPF instances are randomly generated. For this, a precise lower bound is needed.

Proposition 3.1 (Precise Makespan Lower Bound of MAPF on 2D Grids)**.** *The minimum makespan of random* MAPF *instances on an $m_1 \times m_2$ grid with $\Theta(m_1 m_2)$ agents is $m_1 + m_2 - o(m_1)$ with arbitrarily high probability as $m_1 \to \infty$.*

Proof. Without loss of generality, let the constant in $\Theta(m_1 m_2)$ be some $c \in (0, 1]$, i.e., there are $cm_1 m_2$ agents. We examine the top left and bottom right corners of the $m_1 \times m_2$ grid $\mathcal{G}$. In particular, let $\mathcal{G}_{tl}$ (resp., $\mathcal{G}_{br}$) be the top left (resp., bottom right) $\alpha m_1 \times \alpha m_2$ sub-grid of $\mathcal{G}$, for some positive constant $\alpha \ll 1$. For $u \in V(\mathcal{G}_{tl})$ and $v \in V(\mathcal{G}_{br})$, assuming each grid edge has unit distance, then the Manhattan distance between u and v is at least $(1 - 2\alpha)(m_1 + m_2)$. Now, the probability that some $u \in V(\mathcal{G}_{tl})$ and $v \in V(\mathcal{G}_{br})$ are the start and goal, respectively, for a single agent, is α^4. For $cm_1 m_2$ agents, the probability that at least one agent's start and goal fall into $\mathcal{G}_{tl}$ and $\mathcal{G}_{br}$, respectively, is $p = 1 - (1 - \alpha^4)^{cm_1 m_2}$.

Because $(1 - x)^y < e^{-xy}$ for $0 < x < 1$ and $y > 0$ [2], $p > 1 - e^{-\alpha^4 cm_1 m_2}$. Therefore, for arbitrarily small α, we may choose m_1 such that p is arbitrarily close to 1. For example, we may let $\alpha = m_1^{-\frac{1}{8}}$, which decays to zero as $m_1 \to \infty$, then it holds that the makespan is $(1 - 2\alpha)(m_1 + m_2) = m_1 + m_2 - 2m_1^{-\frac{1}{8}}(m_1 + m_2) = m_1 + m_2 - o(m_1)$ with probability $p > 1 - e^{-c\sqrt{m_1 m_2}}$. □

Comparing the upper bound established in Theorem 3.1 and the lower bound from Proposition 3.1 immediately yields

Theorem 3.2 (Optimality Guarantee of GRM)**.** *For random* MAPF *instances on an $m_1 \times m_2$ grid with $\Omega(m_1 m_2)$ agents, as $m_1 \to \infty$,* GRM *computes in polynomial time solutions that are $4(1 + \frac{m_2}{m_1 + m_2})$-makespan optimal, with high probability.*

Proof Sketch. By Proposition 5.1 and Theorem 3.1, the asymptotic optimality ratio is $4\left(1 + \frac{m_2}{m_1 + m_2}\right)$. □

GRM always runs in polynomial time and has the same running time as GRA2D; the high probability guarantee only concerns solution optimality. The same is true for other high-probability algorithms proposed in this paper. We also note that high-probability guarantees imply guarantees in expectation.

4. Near-Optimally Solving MAPF with up to One-Third and One-Half Agent Densities

Though GRM runs in polynomial time and provides constant factor makespan optimality in expectation, the constant factor is 4+ due to the extreme density. In practice, a agent density of around $\frac{1}{3}$ (i.e., $n = \frac{m_1 m_2}{3}$) is already very high. As it turns out, with $n = cm_1 m_2$ for some constant $c > 0$ and $n \leq \frac{m_1 m_2}{2}$, the constant factor can be dropped to close to 1.

4.1 Up to One-Third Density: Shuffling with Highway Heuristics

For $\frac{1}{3}$ density, we work with random MAPF instances in this subsection; arbitrary instances for up to $\frac{1}{2}$ density are addressed in later subsections. Let us assume for the moment that m_1

2. This is because $\log(1 - x) < -x$ for $0 < x < 1$; multiplying both sides by a positive y and exponentiate with base e then yield the inequality.

and m_2 are multiples of three; we partition $\mathcal{G}$ into 3×3 cells (see, e.g., Fig. 1(b) and Fig. 7). We use Fig. 7, where Fig. 7(a) is a random start configuration and Fig. 7(f) is a random goal configuration, as an example to illustrate GRH– Grid Rearrangement with Highways, targeting agent density up to $\frac{1}{3}$. GRH has two phases: *unlabeled reconfiguration* and MAPF *resolution with Grid Rearrangement and highway heuristics*.

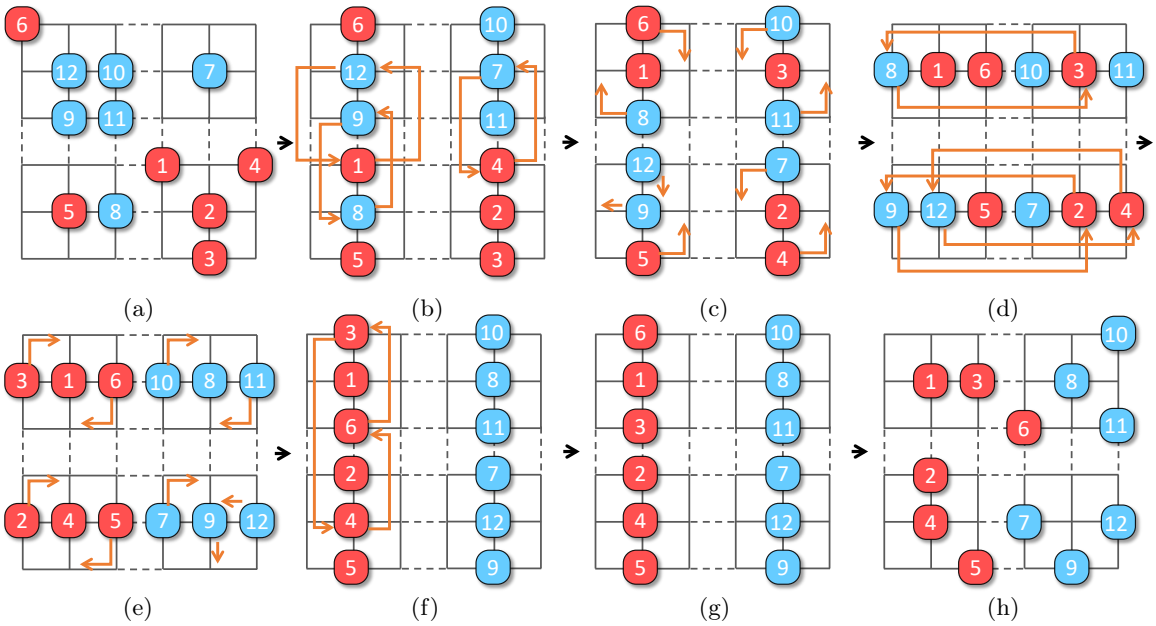


Figure 7: An example of applying GRH to solve an MAPF instance. (a) The start configuration; (b) The start balanced configuration obtained from (a); (c) The intermediate configuration obtained from the Grid Rearrangement preparation phase; (d). The intermediate configuration obtained from (c); (e) The intermediate configuration obtained from the column fitting phase; (f) Apply additional column shuffles for labeled items; (g) The goal balanced configuration obtained from the goal configuration; (h) The goal configuration.

In unlabeled reconfiguration, agents are treated as being indistinguishable. Arbitrary start and goal configurations (under $\frac{1}{3}$ agent density) are converted to intermediate configurations where each 3×3 cell contains no more than 3 agents. We call such configurations *balanced*. With high probability, random MAPF instances are not far from being balanced. To establish this result (Proposition 4.1), we need the following.

Theorem 4.1 (Minimax Grid Matching (Leighton & Shor, 1989))**.** *Consider an $m \times m$ square containing m^2 points following the uniform distribution. Let ℓ be the minimum length such that there exists a perfect matching of the m^2 points to the grid points in the square for which the distance between every pair of matched points is at most ℓ. Then $\ell = O(\log^{\frac{3}{4}} m)$ with high probability.*

Theorem 4.1 applies to rectangles with the longer side being m as well (Theorem 3 in (Leighton & Shor, 1989)).

Proposition 4.1. *On an $m_1 \times m_2$ grid, with high probability, a random configuration of $n = \frac{m_1 m_2}{3}$ agents is of distance $o(m_1)$ to a balanced configuration.*

Proof. We prove for the case of $m_1 = m_2 = 3m$ using the minimax grid matching theorem (Theorem 4.1); generalization to $m_1 \geq m_2$ can be then seen to hold using the generalized version of Theorem 4.1 that applies to rectangles (Theorem 3 of (Leighton & Shor, 1989), which applies to arbitrarily simply-connected region within a square region).

Now let $m_1 = m_2 = 3m$. We may view a random configuration of m^2 agents on a $3m \times 3m$ grid as randomly placing m^2 continuous points in an $m \times m$ square with scaling (by three in each dimension) and rounding. By Theorem 4.1, a random configuration of m^2 continuous points in an $m \times m$ square can be moved to the m^2 grid points at the center of the m^2 disjoint unit squares within the $m \times m$ square, where each point is moved by a distance no more than $O(\log^{\frac{3}{4}} m)$, with high probability. Translating this back to a $3m \times 3m$ gird, m^2 randomly distributed agents on the grid can be moved so that each 3×3 cell contains exactly one agent, and the maximum distance moved for any agent is no more than $O(\log^{\frac{3}{4}} m)$, with high probability. Applying this argument three times yields that a random configuration of $\frac{m_1^2}{3}$ agents on an $m_1 \times m_1$ gird can be moved so that each 3×3 cell contains exactly three agents, and no agent needs to move more than a $O(\log^{\frac{3}{4}} m_1)$ steps, with high probability. Because the agents are indistinguishable, overlaying three sets of reconfiguration paths will not cause an increase in the distance traveled by any agent. □

In the example, unlabeled reconfiguration corresponds to Fig. 7(a)→Fig. 7(b) and Fig. 7(f)→Fig. 7(e) (MAPF solutions are time-reversible). We simulated the process of unlabeled reconfiguration for $m_1 = m_2 = 300$, i.e., on a 300×300 grids. For $\frac{1}{3}$ agent density, the actual number of steps averaged over 100 random instances is less than 5. We call configurations like Fig. 7(b)-(e), which have all agents concentrated vertically or horizontally in the middle of the 3×3 cells, *centered balanced* or simply *centered*. Completing the first phase requires solving two unlabeled problems (Yu & LaValle, 2012; Ma & Koenig, 2016), doable in polynomial time.

In the second phase, GRA is applied with a highway heuristic to get us from Fig. 7(b) to Fig. 7(e), transforming between vertical centered configurations and horizontal centered configurations. To do so, GRA is applied (e.g., to Fig. 7(b) and (e)) to obtain two intermediate configurations (e.g., Fig. 7(c) and (d)). To go between these configurations, e.g., Fig. 7(b)→Fig. 7(c), we apply a heuristic by moving agents that need to be moved out of a 3×3 cell to the two sides of the middle columns of Fig. 7(b), depending on their target direction. If we do this consistently, after moving agents out of the middle columns, we can move all agents to their desired goal 3×3 cell without stopping nor collision. Once all agents are in the correct 3×3 cells, we can convert the balanced configuration to a centered configuration in at most 3 steps, which is necessary for carrying out the next simulated row/column shuffle. Adding things up, we can simulate a shuffle operation using no more than $m + 5$ steps where $m = m_1$ or m_2. The efficiently simulated shuffle leads to low makespan MAPF algorithms. It is clear that all operations take polynomial time; a precise running time is given at the end of this subsection.

Theorem 4.2 (Makespan Upper Bound for Random MAPF, $\leq \frac{1}{3}$ Density)**.** *For random* MAPF *instances on an $m_1 \times m_2$ grid, where $m_1 \geq m_2$ are multiples of three, for $n \leq \frac{m_1 m_2}{3}$*

agents, an $m_1 + 2m_2 + o(m_1)$ makespan solution can be computed in polynomial time, with high probability.

Proof. By Proposition 4.1, unlabeled reconfiguration requires distance $o(m_1)$ with high probability. This implies that a plan can be obtained for unlabeled reconfiguration that requires $o(m_1)$ makespan (for detailed proof, see Theorem 1 from (Yu, 2018)). For the second phase, by Theorem 2.2, we need to perform m_1 parallel row shuffles with a row width of m_2, followed by m_2 parallel column shuffles with a column width of m_1, followed by another m_1 parallel row shuffles with a row width of m_2. Simulating these shuffles require $m_1 + 2m_2 + O(1)$ steps. Altogether, a makespan of $m_1 + 2m_2 + o(m_1)$ is required, with a very high probability. □

Contrasting Theorem 4.2 and Proposition 3.1 yields

Theorem 4.3 (Makespan Optimality for Random MAPF, $\leq \frac{1}{3}$ Density)**.** *For random* MAPF *instances on an $m_1 \times m_2$ grid, where $m_1 \geq m_2$ are multiples of three, for $n = cm_1m_2$ agents with $c \leq \frac{1}{3}$, as $m_1 \to \infty$, a $(1 + \frac{m_2}{m_1+m_2})$ makespan optimal solution can be computed in polynomial time, with high probability.*

Since $m_1 \geq m_2$, $1 + \frac{m_2}{m_1+m_2} \in (1, 1.5]$. In other words, in polynomial running time, GRH achieves $(1 + \delta)$ asymptotic makespan optimality for $\delta \in (0, 0.5]$, with high probability.

From the analysis so far, if m_1 and/or m_2 are not multiples of 3, it is clear that all results in this subsection continue to hold for agent density $\frac{1}{3} - \frac{(m_1 \bmod 3)(m_2 \bmod 3)}{m_1m_2}$, which is arbitrarily close to $\frac{1}{3}$ for large m_1 and m_2. It is also clear that the same can be said for grids with certain patterns of regularly distributed obstacles (Fig. 1(b)), i.e.,

Corollary 4.1 (Random MAPF, $\frac{1}{9}$ Obstacle and $\frac{2}{9}$ Agent Density)**.** *For random* MAPF *instances on an $m_1 \times m_2$ grid, where $m_1 \geq m_2$ are multiples of three and there is an obstacle at coordinates $(3k_1 + 2, 3k_2 + 2)$ for all applicable k_1 and k_2, for $n = cm_1m_2$ agents with $c \leq \frac{2}{9}$, a solution can be computed in polynomial time that has makespan $m_1 + 2m_2 + o(m_1)$ with high probability. As $m_1 \to \infty$, the solution approaches $1 + \frac{m_2}{m_1+m_2}$ optimal, with high probability.*

Proof. Because the total density of robot and obstacles are no more than 1/3, if Theorem 4.1 extends to support regularly distributed obstacles, then Theorem 4.3 applies because the highway heuristics do not pass through the obstacles. This is true because each obstacle can only add a constant length of path detour to an agent's path. In other words, the length ℓ in Theorem 4.3 will only increase by a constant factor and will remain as $\ell = O(\log^{\frac{3}{4}} m)$. Similar arguments hold for Proposition 4.1. □

We now give the running time of GRM and GRH.

Proposition 4.2 (Running Time, GRH)**.** *For $n \leq \frac{m_1m_2}{3}$ agents on an $m_1 \times m_2$ grid,* GRH *runs in $O(nm_1^2m_2)$ time.*

Proof. The running time of GRH is dominated by the matching computation and solving unlabeled MAPF. The matching part takes $O(m_1^2m_2)$ in deterministic time or $O(m_1m_2 \log m_1)$ in expected time (Goel, Kapralov, & Khanna, 2013). Unlabeled MAPF may be tackled

using the max-flow algorithm (Ford & Fulkerson, 1956) in $O(nm_1m_2T) = O(nm_1^2m_2)$ time, where $T = O(m_1 + m_2)$ is the expansion time horizon of a time-expanded graph that allows a routing plan to complete. □

4.2 One-Half Agent Density: Shuffling with Linear Merging

Using a more sophisticated shuffle routine, $\frac{1}{2}$ agent density can be supported while retaining most of the guarantees for the $\frac{1}{3}$ density setting; obstacles are no longer supported.

To best handle $\frac{1}{2}$ agent density, we employ a new shuffle routine called *linear merge*, based on merge sort, and denote the resulting algorithm as Grid Rearrangement with Linear Merge or GRLM. The basic idea is straightforward: for m agents on a $2 \times m$ grid, we iteratively sort the agents first on 2×2 grids, then 2×4 grids, and so on, much like how merge sort works. An illustration of the process on a 2×8 grid is shown in Fig. 8.

Algorithm 2: Line Merge Algorithm

Input: An array arr representing the vertices labeled from 1 to n with current and intermediate locations of n agents.

```
Function LineMerge(arr):
    if length of arr > 1 then
        mid ← length of arr ÷ 2
        left ← arr[0 ... mid − 1]
        right ← arr[mid ... length of arr − 1]
        LineMerge(left)
        LineMerge(right)
        Merge(arr)
Function Merge(arr):
    Sort agents located at the vertices in arr to obtain intermediate states
    for i ∈ arr do
        a_i ← agent at vertex i
        if a_i.intermediate > i then
            Route a_i moving rightward using the bottom line while avoiding blocking those
              agents that are moving leftward
        else
            Route a_i moving leftward using the upper line without stopping
    Synchronize the paths
```

Lemma 4.1 (Properties of Linear Merge)**.** *On a $2 \times m$ grid, m agents, starting on the first row, can be arbitrarily ordered using $m + o(m)$ steps, using the Alg. 2, inspired from merge sort. The motion plan can be computed in polynomial time.*

Proof. We first show *feasibility*. The procedure takes $\lceil \log m \rceil$ phases; in a phase, let us denote a section of the $2 \times m$ grid where robots are treated together as a *block*. For example, the left 2×4 grid in Fig. 8(b) is a block. It is clear that the first phase, involving up to two robots per block, is feasible (i.e., no collision). Assuming that phase k is feasible, we look at phase $k + 1$. We only need to show that the procedure is feasible on one block of length up

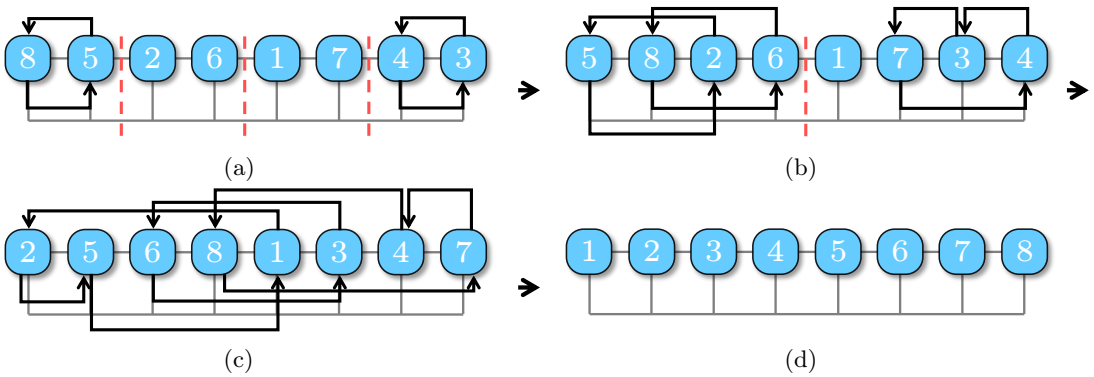


Figure 8: A demonstration of the linear merge shuffle primitive on a 2×8 grid. Agents going to the left always use the upper channel while agents going to the right always use the lower channel.

to 2^{k+1}. For such a block, the left half-block of length up to 2^k is already fully sorted as desired, e.g., in increasing order from left to right. For the $k+1$ phase, all robots in the left half-block may only stay in place or move to the right. These robots that stay must be all at the leftmost positions of the half-block and will not block the motions of any other robot. For the robots that need to move to the right, their relative orders do not need to change and, therefore, will not cause collisions among themselves. Because these robots that move in the left half-block will move down on the grid by one edge, they will not interfere with any robot from the right half-block. Because the same arguments hold for the right half-block (except the direction change), the overall process of merging a block occurs without collision.

Next, we examine the *makespan*. For any single robot r, at phase k, suppose it belongs to block b and block b is to be merged with block b'. It is clear that the robot cannot move more than $len(b') + 2$ steps, where $len(b')$ is the number of columns of b' and the 2 extra steps may be incurred because the robot needs to move down and then up the grid by one edge. This is because any move that r needs to do is to allow robots from b' to move toward b. Because there are no collisions in any phase, adding up all the phases, no robot moves more than $m + 2(\log m + 1) = m + o(m)$ steps.

Finally, the merge sort-like linear merge shuffle primitive runs in $O(m \log m)$ time since it is a standard divide-and-conquer routine with $\log m$ phases. □

We distinguish between $\frac{1}{3}$ and $\frac{1}{2}$ density settings because the overhead in GRLM is larger. Nevertheless, with the linear merge, the asymptotic properties of GRH for $\frac{1}{3}$ agent density mostly carry over to GRLM.

Theorem 4.4 (Random MAPF, $\frac{1}{2}$ Agent Density)**.** *For random* MAPF *instances on an $m_1 \times m_2$ grid, where $m_1 \geq m_2$ are multiples of two, for $\frac{m_1 m_2}{3} \leq n \leq \frac{m_1 m_2}{2}$ agents, a solution can be computed in polynomial time that has makespan $m_1 + 2m_2 + o(m_1)$ with high probability. As $m_1 \to \infty$, the solution approaches an optimality of $1 + \frac{m_2}{m_1 + m_2} \in (1, 1.5]$, with high probability.*

Proof Sketch. The proof follows by combining Proposition 5.1 and Lemma 4.1. □

4.3 Supporting Arbitrary MAPF Instances on Grids

We now examine applying GRH to arbitrary MAPF instances up to $\frac{1}{2}$ agent density. If an MAPF instance is arbitrary, all that changes to GRH is the makespan it takes to complete the unlabeled reconfiguration phase. On an $m_1 \times m_2$ grid, by computing a matching, it is straightforward to show that it takes no more than $m_1 + m_2$ steps to complete the unlabeled reconfiguration phase, starting from an arbitrary start configuration. Since two executions of unlabeled reconfiguration are needed, this adds $2(m_1 + m_2)$ additional makespan. Therefore, the following results holds.

Theorem 4.5 (Arbitrary MAPF, $\leq \frac{1}{2}$ Density)**.** *For arbitrary* MAPF *instances on an $m_1 \times m_2$ grid, $m_1 \geq m_2$, for $n \leq \frac{m_1 m_2}{2}$ agents, a $3m_1 + 4m_2 + o(m_1)$ makespan solution can be computed in polynomial time.*

5. Generalization to 3D

We now explore the 3D setting. To keep the discussion focused, we mainly show how to generalize GRH to 3D, but note that GRM and GRLM can also be similarly generalized. High-dimensional GRP (Szegedy & Yu, 2023) is defined as follows.

Problem 2 (**Grid Rearrangement in kD (GRPKD)**)**.** *Let M be an $m_1 \times \ldots \times m_k$ table, $m_1 \geq \ldots \geq m_k$, containing $\prod_{i=1}^{k} m_i$ items, one in each table cell. In a* shuffle *operation, the items in a single column in the i-th dimension of M, $1 \leq i \leq k$, may be permuted arbitrarily. Given two arbitrary configurations S and G of the items on M, find a sequence of shuffles that take M from S to G.*

We may solve GRP3D using GRA2D as a subroutine by treating GRP as an GRP2D, which is straightforward if we view a 2D slice of GRA2D as a "wide" column. For example, for the $m_1 \times m_2 \times m_3$ grid, we may treat the second and third dimensions as a single dimension. Then, each wide column, which is itself an $m_2 \times m_3$ 2D problem, can be reconfigured by applying GRA2D. With some proper counting, we obtain the following 3D version of Theorem 2.2 as follows.

Theorem 5.1 (**Grid Rearrangement Theorem, 3D**)**.** *An arbitrary Grid Rearrangement problem on an $m_1 \times m_2 \times m_3$ table can be solved using $m_1 m_2 + m_3(2m_2 + m_1) + m_1 m_2$ shuffles.*

Although (Szegedy & Yu, 2023) offers a broad framework for addressing GRPKD, it is not directly applicable to multi-agent routing in high-dimensional spaces. To overcome this limitation, we have developed the high-dimensional MAPF algorithm by incorporating and elaborating on its underlying principles similar to the 2D scenarios. Denote the corresponding algorithm for Theorem 5.1 as GRA3D, we illustrate how it works on a $3 \times 3 \times 3$ table (Fig. 9). GRA operates in three phases. First, a bipartite graph $B(T, R)$ is constructed based on the initial table where the partite set T are the colors of items representing the desired (x, y) positions, and the set R are the set of (x, y) positions of items (Fig. 2(b)). Edges are added as in the 2D case. From $B(T, R)$, m_3 *perfect matchings* are computed. Each matching contains $m_1 m_2$ edges and connects all of T to all of R. processing these matchings yields an intermediate table (Fig. 9(c)), serving a similar function as in the 2D case. After the

first phase of m_1m_2 z-shuffles, the intermediate table (Fig. 9(c)) that can be reconfigured by applying GRA2D with m_1 x-shuffles and $2m_2$ y-shuffles. This sorts each item in the correct x-y positions (Fig. 2(d)). Another m_1m_2 z-shuffles can then sort each item to its desired z position (Fig. 2(e)).

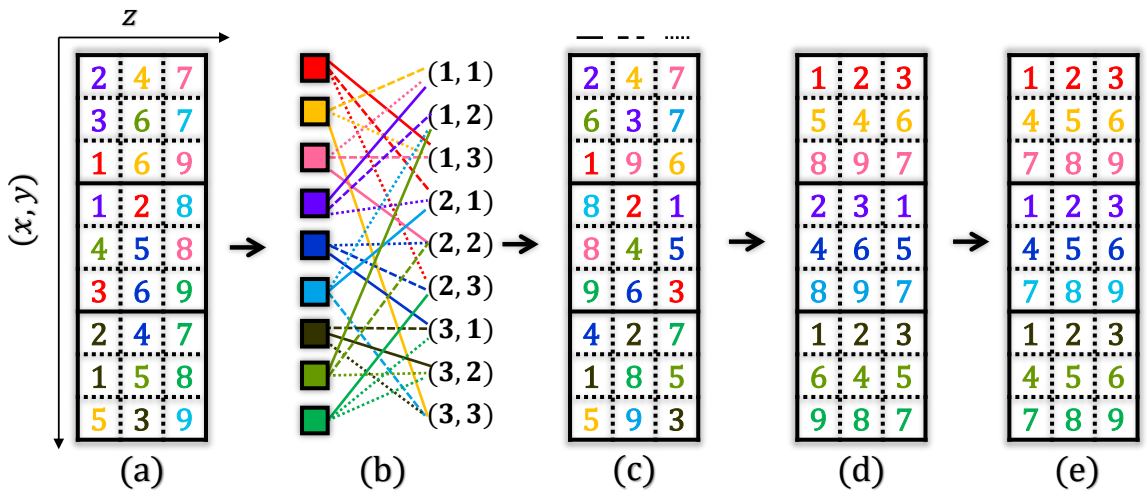


Figure 9: Illustration of applying GRA3D. (a) The initial $3 \times 3 \times 3$ table with a random arrangement of 27 items that are colored and labeled. Color represents the (x, y) position of an item. (b) The constructed bipartite graph. The right partite set contains all the possible (x, y) positions. It contains 3 perfect matchings, determining the 3 columns in (c). (c) Applying z-shuffles to (a), according to the matching results, leads to an intermediate table where each x-y plane has one color appearing exactly once. (d) Applying wide shuffles to (c) correctly places the items according to their (x, y) values (or colors). (e) Additional z-shuffles fully sort the labeled items.

5.1 Extending GRH to 3D

Alg. 3, which calls Alg. 4 and Alg. 5, outlines the high-level process of extending GRH to 3D. In each x-y plane, $\mathcal{G}$ is partitioned into 3×3 cells (e.g.,Fig. 7). Without loss of generality, we assume that m_1, m_2, m_3 are multiples of 3 and there are no obstacles. First, to make GRA3D applicable, we convert the arbitrary start/goal configurations to intermediate *balanced* configurations S_1 and G_1, treating agents as unlabeled, as we have done in GRH, wherein each 2D plane, each 3×3 cell contains no more than 3 agents (Fig. 10).

GRA3D can then be applied to coordinate the agents moving toward their intermediate goal positions G_1. Function `MatchingXY` finds a feasible intermediate configuration S_2 and routes the agents to S_2 by simulating shuffle operations along the z axis. Function `XY-Fitting` apply shuffle operations along the x and y axes to route each agent i to its desired x-y position $(g_{1i}.x, g_{1i}.y)$. In the end, the function `Z-Fitting` is called, routing each agent i to the desired g_{1i} by performing shuffle operations along the z axis and concatenating the paths computed by unlabeled MAPF planner `UnlabeledMRPP`.

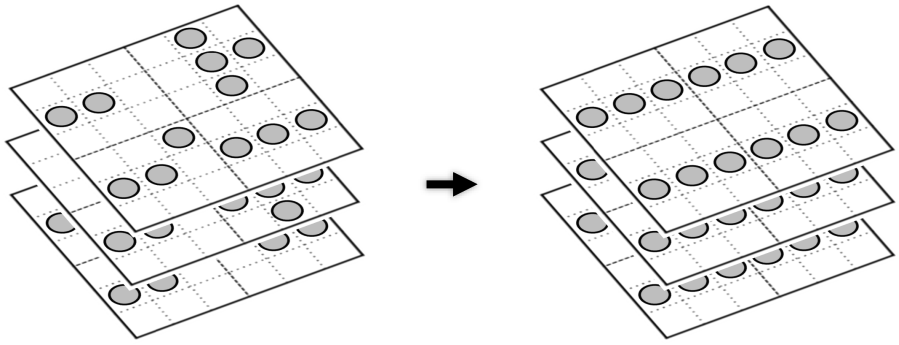


Figure 10: Applying unlabeled MAPF to convert a random configuration to a balanced centering one on $6 \times 6 \times 3$ grids.

Algorithm 3: GRH3D

Input: Start and goal vertices $S = \{s_i\}$ and $G = \{g_i\}$

1 **Function** `GRH3D`(S, G)**:**
2 $S_1, G_1 \leftarrow$`UnlabeledMRPP`(S, G)
3 `MatchingXY()`
4 `XY-Fitting()`
5 `Z-Fitting()`

We now explain each part of GRH3D. `MatchingXY` uses an extended version of GRA to find perfect matching that allows feasible shuffle operations. Here, the "item color" of an item i (agent) is the tuple $(g_{1i}.x, g_{1i}.y)$, which is the desired x-y position it needs to go. After finding the m_3 perfect matchings, the intermediate configuration S_2 is determined. Then, shuffle operations along the z direction can be applied to move the agents to S_2. The agents in each x-y plane will be reconfigured by applying x-shuffles and y-shuffles. We need to apply GRA2D for these agents in each plane, as demonstrated in Alg. 5. In GRH, for each 2D plane, the "item color" for agent i is its desired x position $g_{1i}.x$. For each plane, we compute the m_2 perfect matchings to determine the intermediate position g_2. Then each agent i moves to its g_{2i} by applying y-shuffle operations. In Line 18, each agent is routed to its desired x position by performing x-shuffle operations. In Line 19, each agent is routed to its desired y position by performing y-shuffle operations. After all the agents reach the desired x-y positions, another round of z-shuffle operations in `Z-Fitting` can route the agents to the balanced goal configuration computed by an unlabeled MAPF planner. In the end, we concatenate all the paths as the result.

GRLM and GRM can be extended to 3D in similar ways by replacing the solvers in 2D planes. For GRM, there is no need to use unlabeled MAPF for balanced reconfiguration, which yields a makespan upper bound of $4m_1 + 8m_2 + 8m_3$ (as a direct combination of Theorem 3.1 and Theorem 5.1). For arbitrary instances under half density in 3D, the makespan guarantee in Theorem 4.5 updates to $3m_1 + 4m_2 + 4m_3 + o(m_1)$.

Algorithm 4: MatchingXY

Input: Balanced start and goal vertices $S_1 = \{s_{1i}\}$ and $G_1 = \{g_{1i}\}$

Function `MatchingXY(`S_1, G_1`)`**:**
- $A \leftarrow [1, ..., n]$
- $\mathcal{T} \leftarrow$ the set of (x, y) positions in S_1
- **for** $(r, t) \in \mathcal{T} \times \mathcal{T}$ **do**
  - **if** $\exists i \in A$ *where* $(s_{1i}.x, s_{1i}.y) = r \wedge (g_{1i}.x, g_{1i}.y) = t$ **then**
    - add edge (r, t) to $B(T, R)$
    - remove i from A
- compute matchings $\mathcal{M}_1, ..., \mathcal{M}_{m_3}$ of $B(T, R)$
- $A \leftarrow [1, ..., n]$
- **foreach** $\mathcal{M}_c$ *and* $(r, t) \in \mathcal{M}_c$ **do**
  - **if** $\exists i \in A$ *where* $(s_{1i}.x, s_{1i}.y) = r \wedge (g_{1i}.x, g_{1i}.y) = t$ **then**
    - $s_{2i} \leftarrow (s_{1i}.x, s_{1i}.y, c)$ and remove i from A
    - mark agent i to go to s_{2i}
- perform simulated z-shuffles in parallel

Algorithm 5: XY-Fitting

Input: Current positions S_2 and balanced goal positions G_1

Function `XY-Fitting()`**:**
- **for** $z \leftarrow [1, ..., m_3]$ **do**
  - $A \leftarrow \{i | s_{2i}.y = z\}$
  - `GRH2D(`A, z`)`

Function `GRH2D(`A, z`)`**:**
- $\mathcal{T} \leftarrow$ the set of x positions of S_2
- **for** $(r, t) \in \mathcal{T} \times \mathcal{T}$ **do**
  - **if** $\exists i \in A$ *where* $s_{2i}.x = r \wedge g_{1i}.x = t$ **then**
    - **if** *agent* i *is not assigned* **then**
      - add edge (r, t) to $B(T, R)$
      - mark i assigned
  - compute matchings $\mathcal{M}_1, ..., \mathcal{M}_{m_2}$ of $B(T, R)$
- $A' \leftarrow A$
- **foreach** $\mathcal{M}_c$ *and* $(r, t) \in \mathcal{M}_c$ **do**
  - **if** $\exists i \in A'$ *where* $s_{2i}.x = c \wedge g_{1i}.x = t$ **then**
    - $g_{2i} \leftarrow (s_{2i}.x, c, z)$ and remove i from A'
    - mark agent i to go to g_{2i}
- route each agent $i \in A$ to g_{2i}
- route each agent $i \in A$ to $(g_{2i}.x, g_{1i}.y, z)$
- route each agent $i \in A$ to $(g_{1i}.x, g_{1i}.y, z)$

5.2 Properties of GRH3D

Computation for GRH3D is dominated by perfect matching and unlabeled MAPF. Finding m_3 "wide column" matchings takes $O(m_3 m_1^2 m_2^2)$ deterministic time or $O(m_3 m_1 m_2 \log(m_1 m_2))$ expected time. We apply GRA2D for simulating "wide column" shuffle, which requires $O(m_3 m_1^2 m_2)$ deterministic time or $O(m_1 m_2 m_3 \log m_1)$ expected time. Therefore, the total time complexity for the Grid Rearrangement part is $O(m_3 m_1^2 m_2^2 + m_1^2 m_2 m_3)$. For unlabeled MAPF, we can use the max-flow based algorithm (Yu & LaValle, 2013a) to compute the makespan-optimal solutions. The max-flow portion can be solved in $O(n|E|T) = O(n^2(m_1 + m_2 + m_3))$ where $|E|$ is the number of edges and $T = O(m_1 + m_2 + m_3)$ is the time horizon of the time expanded graph (Ford & Fulkerson, 1956). One can also perform two "wide column" shuffles plus one z shuffle, which yields $2(2m_1 + m_2) + m_3$ number of shuffles and $O(m_1 m_2 m_3^2 + m_3 m_1 m_2^2)$. This requires more shuffles but a shorter running time.

Next, we derive the optimality guarantee, for which the upper and lower bounds on the makespan are needed. These are straightforward generalizations of results in 2D.

Theorem 5.2 (Makespan Upper Bound)**.** GRH3D *returns solutions with* $m_1 + 2m_2 + 2m_3 + o(m_1)$ *asymptotic makespan for* MAPF *instances with* $\frac{m_1 m_2 m_3}{3}$ *random start and goal configurations on 3D grids, with high probability.* GRLM3D *returns solutions with* $m_1 + 2m_2 + 2m_3 + o(m_1)$ *asymptotic makespan for* MAPF *instances with* $\frac{m_1 m_2 m_3}{2}$ *random start and goal configurations on 3D grids, with high probability.* GRM3D *returns solutions with* $3m_1 + 6m_2 + 6m_3 + o(m_1)$ *asymptotic makespan for* MAPF *instances with* $m_1 m_2 m_3$ *random start and goal configurations on 3D grids, with high probability. Moreover, if* $m_1 = m_2 = m_3 = m$*,* GRH3D *and* GRLM3D *returns solutions with* $5m + o(m)$ *makespan,* GRM3D *returns solutions with* $15m + o(m)$ *makespan.*

Proposition 5.1 (Makespan Lower Bound)**.** *For* MAPF *instances on* $m_1 \times m_2 \times m_3$ *grids with* $\Theta(m_1 m_2 m_3)$ *random start and goal configurations on 3D grids, the makespan lower bound is asymptotically approaching* $m_1 + m_2 + m_3$*, with high probability.*

Theorem 5.3 (Makespan Optimality Ratio)**.** GRH3D *and* GRLM3D *yield asymptotic* $1 + \frac{m_2 + m_3}{m_1 + m_2 + m_3}$ *makespan optimality ratio for* MAPF *instances with* $\Theta(m_1 m_2 m_3) \leq \frac{m_1 m_2 m_3}{3}$ *and* $\leq \frac{m_1 m_2 m_3}{2}$ *random start and goal configurations respectively on 3D grids, with high probability.* GRM3D *yields asymptotic* $3 + \frac{3(m_1 + m_2)}{m_1 + m_2 + m_3}$ *makespan optimality ratio for* MAPF *instances with* $\Theta(m_1 m_2 m_3) \leq m_1 m_2 m_3$ *random start and goal configurations on 3D grids with high probability.*

We may further generalize the result to higher dimensions.

Theorem 5.4. *Consider a* k*-dimensional cubic grid with grid size* m*. For uniformly distributed start and goal configurations,* GRH *and* GRLM *can solve the instances with asymptotic makespan optimality being* $\frac{2^{k-1}+1}{k}$ *and* GRM *yields* $\frac{4(2^{k-1}+1)}{k}$ *asymptotic makespan optimality.*

Proof. Unlabeled MAPF takes $o(m)$ makespan (note for $k \geq 3$, the minimax grid matching distance is $O(\log^{1/k} m)$ (Shor & Yukich, 1991)). Extending proposition 5.1 to k-dimensional grid, the asymptotic lower bound is mk. We prove that the asymptotic makespan $f(k)$ is $(2^{k-1} + 1)m + o(m)$ by induction. The Grid Rearrangement algorithm solves a k-dimensional

problem by using two 1-dimensional shuffles and one $(k-1)$-dimensional "wide column" shuffle. Therefore, we have $f(k) = 2m+f(k-1)$. It's trivial to see $f(1) = m+o(m)$, $f(2) = 3m+o(m)$, which yields that $f(k) = 2^{k-1}m + m + o(m)$ and makespan optimality ratio being $\frac{2^{k-1}+1}{k}$ for GRH and GRLM while $\frac{4(2^{k-1}+1)}{k}$ for GRM. □

6. Optimality-Boosting Heuristics

6.1 Reducing Makespan via Optimizing Matching

Based on GRA, GRH has three simulated shuffle phases. The makespan is dominated by the agent needing the longest time to move, as a sum of moves in all three phases. As a result, the optimality of Grid Rearrangement methods is determined by the first preparation/matching phase. Finding arbitrary perfect matchings is fast but the process can be improved to reduce the overall makespan.

For improving matching, we propose two heuristics; the first is based on *integer programming* (IP). We create binary variables $\{x_{ri}\}$ where r represents the row number and i the agent. agent i is assigned to row r if $x_{ri} = 1$. Define single agent cost as $C_{ri}(\lambda) = \lambda|r - s_i.x| + (1-\lambda)|r - g_i.x|$. We optimize the makespan lower bound of the first phase by letting $\lambda = 0$ or the third phase by letting $\lambda = 1$. The objective function and constraints are given by

$$\max_{r,i}\{C_{ri}(\lambda = 0)x_{ri}\} + \max_{r,i}\{C_{ri}(\lambda = 1)x_{ri}\} \tag{1}$$

$$\sum_{r} x_{ri} = 1, \text{for each agent } i \tag{2}$$

$$\sum_{g_i.y=t} x_{ri} \leq 1, \text{for each row } r \text{ and each color } t \tag{3}$$

$$\sum_{s_i.x=c} x_{ri} = 1, \text{for each column } c\text{and each row } r \tag{4}$$

Eq. (1) is the summation of makespan lower bound of the first phase and the third phase. Note that the second phase can not be improved by optimizing the matching. Eq. (2) requires that agent i be only present in one row. Eq. (3) specifies that each row should contain agents that have different goal columns. Eq. (4) specifies that each vertex (r, c) can only be assigned to one agent. The IP model represents a general assignment problem which is NP-hard in general. It has limited scalability but provides a way to evaluate how optimal the matching could be in the limit.

A second matching heuristic we developed is based on *linear bottleneck assignment (LBA)* (Burkard, Dell'Amico, & Martello, 2012), which takes polynomial time. LBA differs from the IP heuristic in that the bipartite graph is weighted. For the matching assigned to row r, the edge weight of the bipartite graph is computed greedily. If column c contains agents of color t, we add an edge (c, t) and its edge cost is

$$C_{ct} = \min_{g_i.y=t} C_{ri}(\lambda = 0) \tag{5}$$

We choose $\lambda = 0$ to optimize the first phase. Optimizing the third phase ($\lambda = 1$) would give similar results. After constructing the weighted bipartite graph, an $O(\frac{m_1^{2.5}}{\log m_1})$ LBA

algorithm (Burkard et al., 2012) is applied to get a minimum bottleneck cost matching for row r. Then we remove the assigned agents and compute the next minimum bottleneck cost matching for the next row. After getting all the matchings $\mathcal{M}_r$, we can further use LBA to assign $\mathcal{M}_r$ to a different row r' to get a smaller makespan lower bound. The cost for assigning matching $\mathcal{M}_r$ to row r' is defined as

$$C_{\mathcal{M}_r r'} = \max_{i \in \mathcal{M}_r} C_{r'i}(\lambda = 0) \tag{6}$$

The total time complexity of using LBA heuristic for matching is $O(\frac{m_1^{3.5}}{\log m_1})$.

We denote GRH with IP and LBA heuristics as GRH-IP and GRH-LBA, respectively. We mention that GRM, which uses the line swap motion primitive, can also benefit from these heuristics to re-assign the goals within each group. This can lower the bottleneck path length and improve optimality.

6.2 Path Refinement

Final paths from GRA-based algorithms are concatenations of paths from multiple planning phases. This means agents are forced to "synchronize" at phase boundaries, which causes unnecessary idling for agents finishing a phase early. Noticing this, we de-synchronize the planning phases, which yields significant gains in makespan optimality.

Our path refinement method does something similar to Minimal Communication Policy (MCP) (Ma, Kumar, & Koenig, 2017a), a multi-agent execution policy proposed to handle unexpected delays without stopping unaffected agents. During execution, MCP let agents execute their next non-idling move as early as possible while preserving the relative execution orders between agents, e.g., when two agents need to enter the same vertex at different times, that ordering is preserved.

We adopt the principle used in MCP to refine the paths generated by GRA-based algorithms as shown in Alg. 6, with Alg. 7 as a sub-routine. The algorithms work as follows. All idle states are removed from the initial agent but the order of visits for each vertex is kept (Line 2-3). Then we enter a loop executing the plans following the MCP principle (Line 8-10). In Alg. 7, if i is the next agent that enters vertex v_i according to the original order, we check if there is a agent currently at v_i. If there is not, we let i enter v_i. If another agent j is currently occupying v_j, we examine if j is moving to its next vertex v_j in the next step by recursively calling the function `Move`. We check if there is any cycle in the agent movements in the next original plan. If i is in a cycle, we move all the agents in this cycle to their next vertex recursively (Line 20-28). If no cycle is found and j is to enter its next vertex v_j, we let i also move to its next vertex v_i. Otherwise, i should wait at u_i. The algorithm is *deadlock-free* by construction (Ma et al., 2017a).

Let T be the makespan of the initial paths, the makespan of the solution obtained by running `Refine` is clearly no more than T. In each loop of `Refine`, we essentially run DFS on a graph that has n nodes and traverse all the nodes, for which the time complexity is $O(n)$, Therefore the time complexity of the path refinement is bounded by $O(nT)$.

Other path refinement methods, such as (Li, et al., 2021a; Okumura, Tamura, & Défago, 2021), can also be applied in principle, which iteratively chooses a small group of agents and re-plan their paths holding other agents as dynamic obstacles. However, in dense settings that we tackle, re-planning for a small group of agents has little chance of finding better paths this way.

Algorithm 6: Path Refinement

Input: Paths $\mathcal{P}$ generated by GRA

Function *Refine($\mathcal{P}$)*:
  foreach $v \in V$, $VOrder[v] \leftarrow Queue()$
  `Preprocess(`$InitialPlans, VOrder$`)`
  while *true* **do**
    for $i = 1...n$ **do**
      $Moved \leftarrow Dict()$
      $CycleCheck \leftarrow Set()$
      `Move(`i`)`
      if `AllReachedGoals()`=*true* **then**
        break

Algorithm 7: Move

Function *Move(i)*:
  if i *in* $Moved$ **then**
    return $Moved[i]$
  $u_i \leftarrow$ current position of i
  $v_i \leftarrow$ next position of i
  if $i = VOrder[v_i].front()$ **then**
    $j \leftarrow$ the agent currently at v_i
    if i *is in* $CycleCheck$ **then**
      `MoveAllAgentsInCycle(`i`)`
      return true
    $CycleCheck.add(i)$
    if *Move*$(j) = true$ *or* $j =None$ **then**
      let i enter v_i
      $VOrder[v_i].popfront()$
      $Moved[i] \leftarrow$true
      return true
  let i wait at u_i
  $Moved[i]$=false
  return false
Function *MoveAllAgentsInCycle*(i):
  $Visited \leftarrow Set()$
  $j \leftarrow i$
  while j *is not in visited* **do**
    let j enter its next vertex v_j
    $Visited.add(j)$
    $VOrder[v_j].popfront()$
    $Moved[j] \leftarrow$ true
    $j \leftarrow$ the agent currently at vertex v_j

7. Simulation Experiments

We thoroughly evaluated GRA-based algorithms and compared them with many similar algorithms. We mainly highlight comparisons with EECBS (w=1.5) (Li, Ruml, & Koenig, 2021b), LaCAM (Okumura, 2023) and DDM (Han & Yu, 2020). These two methods are, to our knowledge, two of the fastest near-optimal MAPF solvers. Beyond EECBS and DDM, we considered a state-of-the-art polynomial algorithm, push-and-swap (Luna & Bekris, 2011), which gave fairly sub-optimal results: the makespan optimality ratio is often above 100 for the densities we examine.

As a reader's guide to this section, in Sec. 7.1, as a warm-up, we show the 2D performance of all GRA-based algorithms at their baseline, i.e., without any efficiency-boosting heuristics mentioned in Sec. 6. In Sec. 7.4, for 2D square grids, we show the performance of all GRA-based algorithms with and without the two heuristics discussed in Sec. 6. We then thoroughly evaluate the performance of all versions of the GRH 2d algorithm at $\frac{1}{3}$ agent density in Sec. 7.2. Some special 2D patterns are examined in Sec. 7.3. 3D settings are briefly discussed in Sec. 7.5.

All experiments are performed on an Intel® Core™ i7-6900K CPU at 3.2GHz. Each data point is an average of over 20 runs on randomly generated instances unless otherwise stated. A running time limit of 300 seconds is imposed over all instances. The optimality ratio is estimated as compared to conservatively estimated makespan lower bounds. All the algorithms are implemented in C++. We choose Gurobi (Gurobi Optimization, LLC, 2021) as the mixed integer programming solver and ORtools (Perron & Furnon,) as the max-flow solver.

7.1 Optimality of Baseline Versions of GRA-Based Methods

First, we provide an overall evaluation of the optimality achieved by basic versions of GRM, GRLM, and GRH over randomly generated 2D instances at their maximum designed agent density. That is, these methods do not contain the heuristics from Sec. 6. We test over three $m_1 : m_2$ ratios: $1 : 1$, $3 : 2$, and $5 : 1$. For GRM, different sub-grid sizes for dividing the $m_1 \times m_2$ grid are evaluated. The result is plotted in Fig. 11. Computation time is not listed; we provide the computation time later for GRH; the running times of GRM, GRLM, and GRH are all similar. The optimality ratio is computed as the ratio between the solution makespan and the longest Manhattan distance between any pair of start and goal, which is conservative.

GRM does better and better on the optimality ratio as the sub-grid size ranges from 3×2, 3×3, 2×3, and 2×4, dropping to just above 3. In general, using "longe" sub-grids for the shuffle will decrease the optimality ratio because there are opportunities to reduce the overhead. However, the time required for computing the solutions for all possible configurations grows exponentially as the size of the sub-grids increases.

On the other hand, both GRLM and GRH achieve a sub-2 optimality ratio in most test cases, with the result for GRH dropping below 1.5 on large grids. For all settings, as the grid size increases, there is a general trend of improvement of optimality across all methods/grid aspect ratios. This is due to two reasons: (1) the overhead in the shuffle operations becomes relatively smaller as grid size increases, and (2) with more agents, the makespan lower bound becomes closer to $m_1 + m_2$. Lastly, as $m_1 : m_2$ ratio increases, the optimality ratio improves

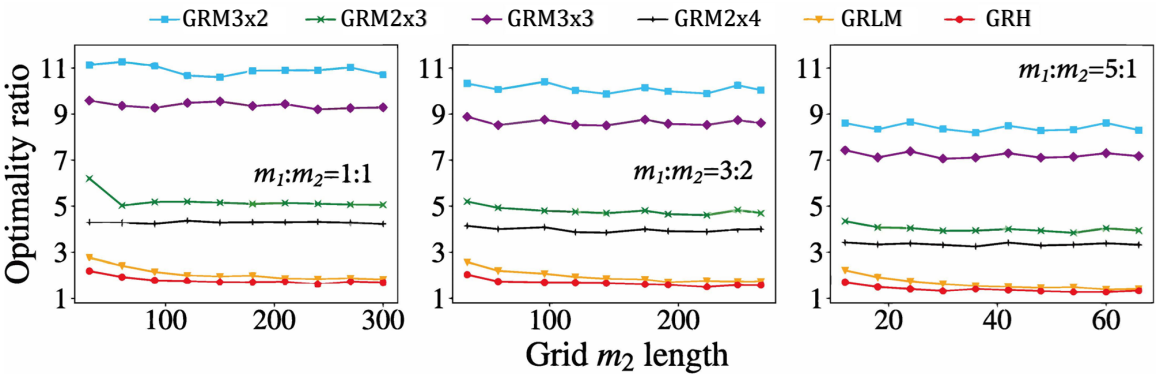


Figure 11: Makespan optimality ratio for GRM (3x2, 2x3, 3x3, 2x4), GRLM, and GRH at their maximum design density, for different m_2 values and $m_1 : m_2$ ratios. The largest GRM problems have $90,000$ agents on a 300×300 grid.

as predicted. For many test cases, the optimality ratio for GRH at $m_1 : m_2 = 5$ is around 1.3.

# of Agents	5	10	15	20	25
Optimality Gap	1	1.0025	1.004	1.011	1.073

Table 2: Optimality gap investigation on 5×5 grids

The exploration of optimality gaps is conducted on 5×5 grids, as shown in Table. 2. For every specified number of agents, we create 100 random instances and employ the ILP solver to solve them. The optimality gap is then assessed by calculating the average ratio between the optimal makespan and the makespan lower bound. The optimality gap widens with higher agent density.

7.2 Evaluation and Comparative Study of GRH

7.2.1 Impact of Grid Size

For our first detailed comparative study of the performance of GRA,GRLM and GRH at 100%, $\frac{1}{2}$ and $\frac{1}{3}$ density respectively, we set $m_1 : m_2 = 3 : 2$ in terms of computation time and makespan optimality ratio. We compare with DDM (Han & Yu, 2020), EECBS ($w = 1.5$) (Li et al., 2021b), Push and Swap(Luna & Bekris, 2011), LaCAM (Okumura, 2023) in Fig. 12-14. For EECBS, we turn on all the available heuristics and reasonings.

When at 100% agent density, GRM methods can solve huge instances, e.g., on 450×300 grids with $135,000$ agents in about 40 seconds while none of the other algorithms can. LaCAM can only solve instances when $m_2 = 30$, though the resulting makespan optimality is around 2073 and thus is not shown in the figure. When at $\frac{1}{2}, \frac{1}{3}$ agent density, GRLM and GRH still are the fastest methods among all, scaling to 45,000 agents in 10 seconds while the optimality ratio is close to 1.5 and 1.3 respectively. LaCAM also achieves great scalability and is able to solve problems when $m_2 \leq 270$. However, after that, LaCAM faces

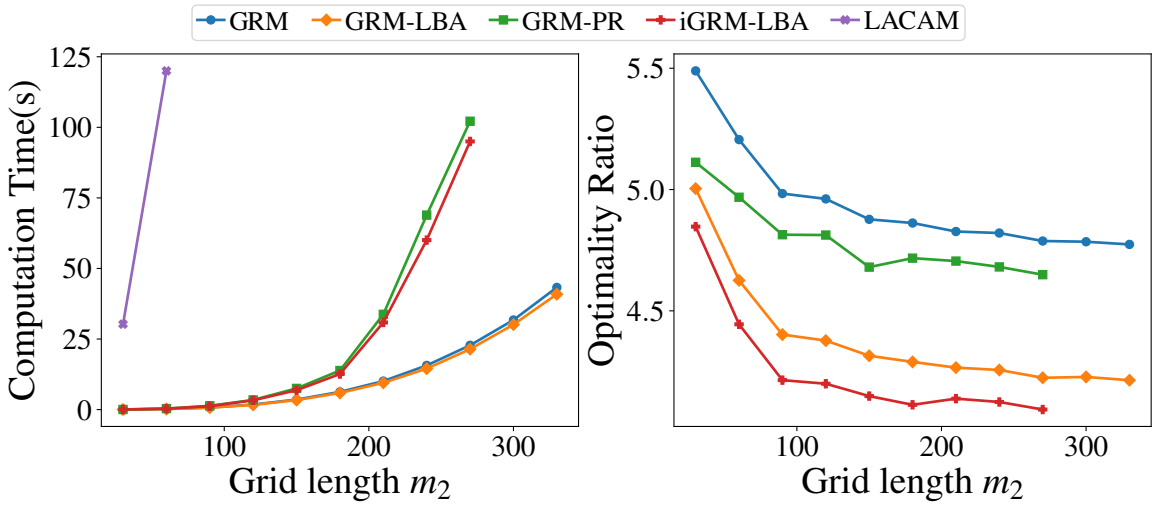


Figure 12: Computation time and optimality ratios on $m_1 \times m_2$ grids of varying sizes with $m_1 : m_2 = 3 : 2$ and agent density at 100% density.

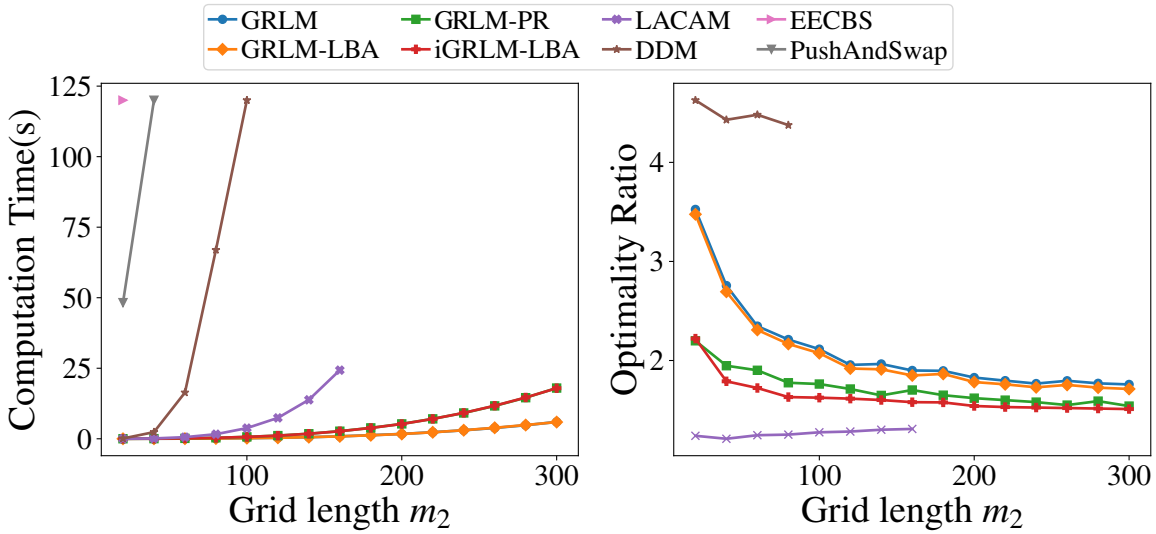


Figure 13: Computation time and optimality ratios on $m_1 \times m_2$ grids of varying sizes with $m_1 : m_2 = 3 : 2$ and agent density at $\frac{1}{2}$ density.

out-of-memory error. This is due to the fact that LaCAM is a search algorithm on joint configurations, and the required memory grows exponentially as the number of agents. In contrast, GRH, GRLM do not have the issue and solve the problems consistently despite the optimality being worse than LaCAM. EECBS, stopped working after $m_2 = 90$ at $\frac{1}{3}$ agent density and cannot solve any instance within the time limit at 100% and $\frac{1}{2}$ agent density,

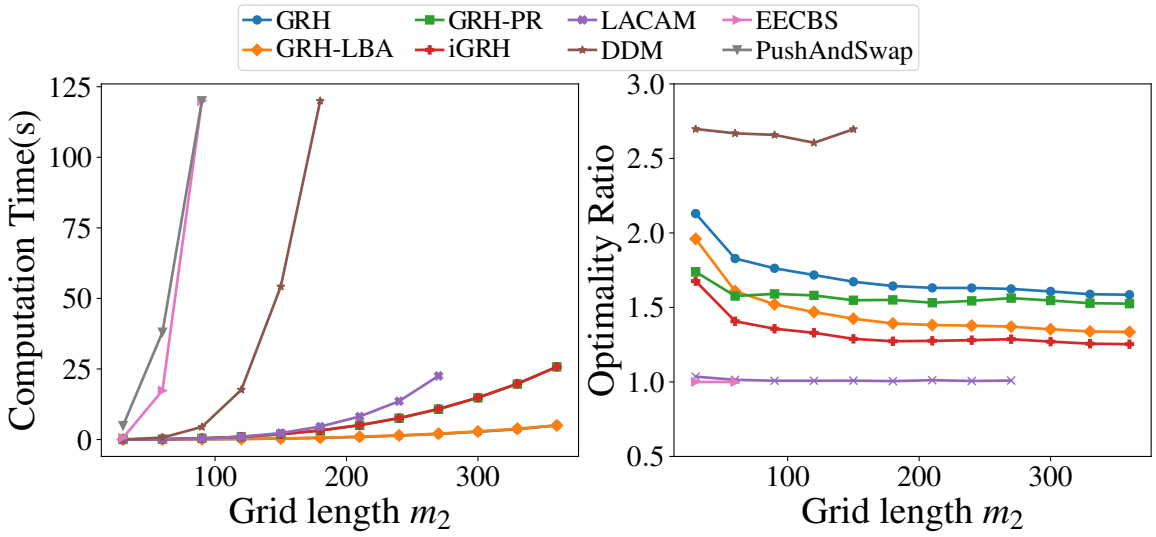


Figure 14: Computation time and optimality ratios on $m_1 \times m_2$ grids of varying sizes with $m_1 : m_2 = 3 : 2$ and agent density at 1/3 density.

while DDM stopped working after $m_2 = 180$. Push and Swap also performed poorly, and its optimality ratio is more than 30 in those scenarios and thus is not presented in the figure.

7.2.2 Handling Obstacles

GRH can also handle scattered obstacles and is especially suitable for cases where obstacles are regularly distributed. For instance, problems with underlying graphs like that in Fig. 1(b), where each 3×3 cell has a hole in the middle, can be natively solved without performance degradation. Such settings find real-world applications in parcel sorting facilities in large warehouses (Wan et al., 2018; Li, et al., 2020). For this parcel sorting setup, we fix the agent density at $\frac{2}{9}$ and test EECBS, DDM, GRH, GRH-LBA, GRH-PR, iGRH on graphs with varying sizes. The results are shown in Fig 15. Note that DDM can only apply when there is no narrow passage. So we added additional "borders" to the map to make it solvable for DDM. The results are similar as before; we note that iGRH reaches a conservative optimality ratio estimate of 1.26.

7.2.3 Impact of Grid Aspect Ratios

In this section, we fix $m_1 m_2 = 90000$ and vary the $m_2 : m_1$ ratio between 0 (nearly one dimensional) and 1 (square grids). We evaluated four algorithms, two of which are GRH and iGRH. Now recall that GRP on an $m_1 \times m_2$ table can also be solved using $2m_2$ column shuffles and m_1 row shuffles. Adapting GRH and iGRH with $m_1 + 2m_2$ shuffles gives the other two variants we denote as GRH-LL and iGRH-LL, with "LL" suggesting two sets of longer shuffles are performed (each set of column shuffle work with columns of length m_1). The result is summarized in Fig. 16.

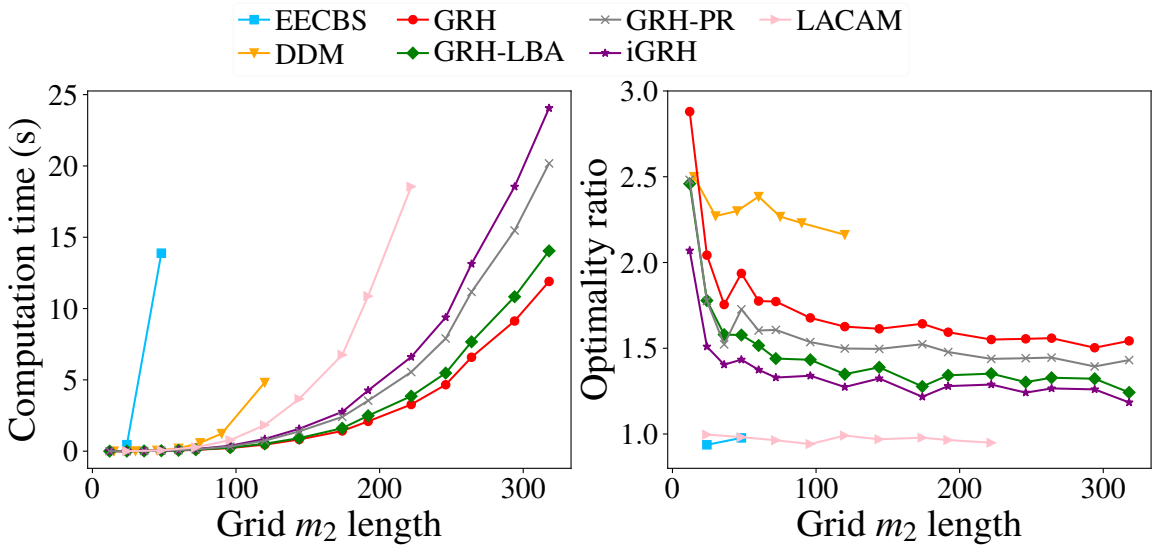


Figure 15: Computation time and optimality ratios on environments of varying sizes with regularly distributed obstacles at $\frac{1}{9}$ density and agents at $\frac{2}{9}$ density. $m_1 : m_2 = 3 : 2$.

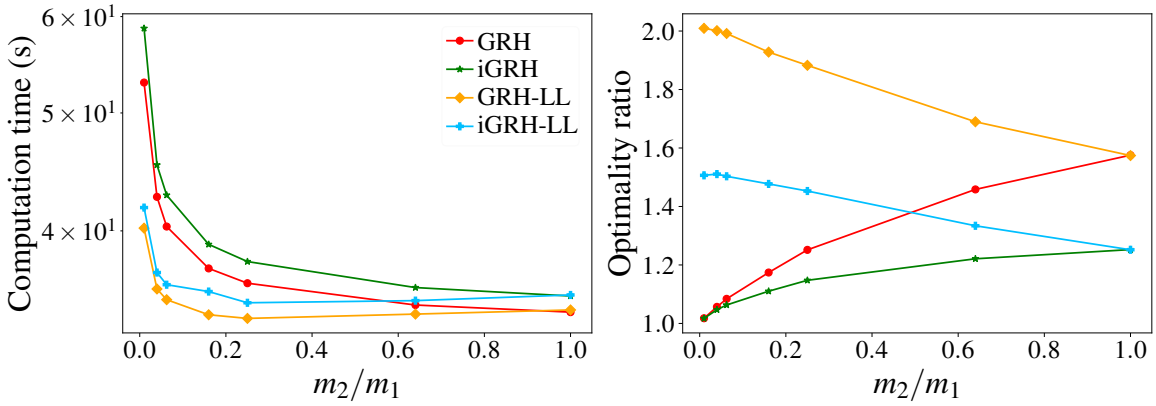


Figure 16: Computation time and optimality ratios on rectangular grids of varying aspect ratio and $\frac{1}{3}$ agent density.

Interestingly but not surprisingly, the result clearly demonstrates the trade-offs between computation effort and solution optimality. GRH and iGRH achieve better optimality ratio in comparison to GRH-LL and iGRH-LL but require more computation time. Notably, the optimality ratio for GRH and iGRH is very close to 1 when $m_2 : m_1$ is close to 0. As expected, iGRH does much better than GRH across the board.

7.3 Special Patterns

We experimented iGRH on many "special" instances, two are presented here (Fig. 17). For both settings, we set $m_1 = m_2$. In the first, the "squares" setting, agents form concentric square rings and each agent and its goal are centrosymmetric. In the second, the "blocks" setting, the grid is divided into smaller square blocks (not necessarily 3×3) containing the same number of agents. Agents from one block need to move to another randomly chosen block. iGRH achieves optimality that is fairly close to 1.0 in the square setting and 1.7 in the block setting. The computation time is similar to that of Fig. 15; EECBS performs well in terms of optimality, but its scalability is limited, working only on grids with $m \leq 90$. On the other hand, LaCAM exhibits excellent scalability and good optimality for block patterns, although its optimality is comparatively worse for square patterns. Other algorithms are excluded from consideration due to significantly poorer optimality; for example, DDM's optimality exceeds 9, and Push&Swap's optimality is greater than 40.

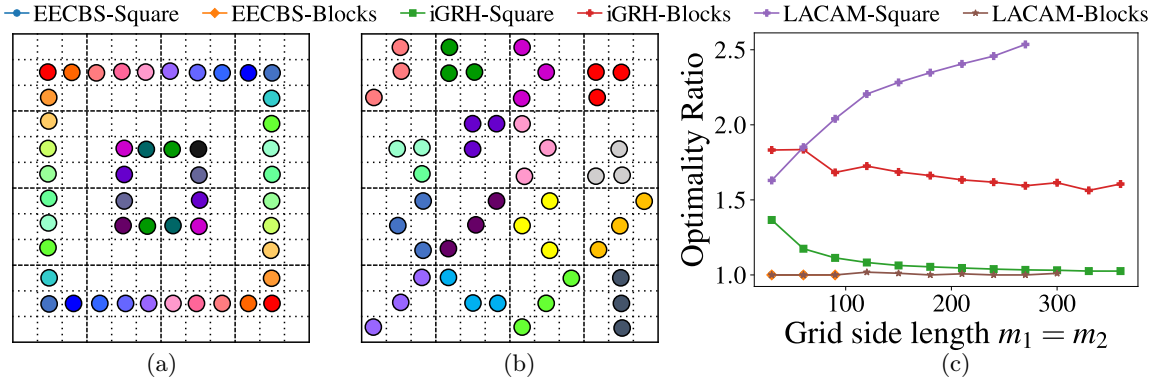


Figure 17: (a) An illustration of the "squares" setting. (b) An illustration of the "blocks" setting. (c) Optimality ratios for the two settings for EECBS, LaCAM, and GRH-LBA.

7.4 Effectiveness of Matching and Path Refinement Heuristics

In this subsection, we evaluate the effectiveness of heuristics introduced in Sec. 6 in boosting the performance of baseline GRA-based methods (we will briefly look at the IP heuristic later). We present the performance of GRM, GRLM, and GRH at these methods' maximum design density. For each method, results on all 4 combinations with the heuristics are included. For a baseline method X, X-LBA, X-PR, and iX mean the method with the LBA heuristic, the path refinement heuristic, and both heuristics, respectively. In addition to the makespan optimality ratio, we also evaluated *sum-of-cost* (SOC) optimality ratio, which may be of interest to some readers. The sum-of-cost is the sum of the number of steps taking individual agents to reach their respective goals. We tested the worst-case scenario, i.e., $m = m_1 = m_2$. The result is shown in Fig. 18.

We make two key observations based on Fig. 18. First, both heuristics provide significant individual boosts to nearly all baseline methods (except GRLM-LBA), delivering around 10%–20% improvement on makespan optimality and 30%–40% improvement on SOC optimality.

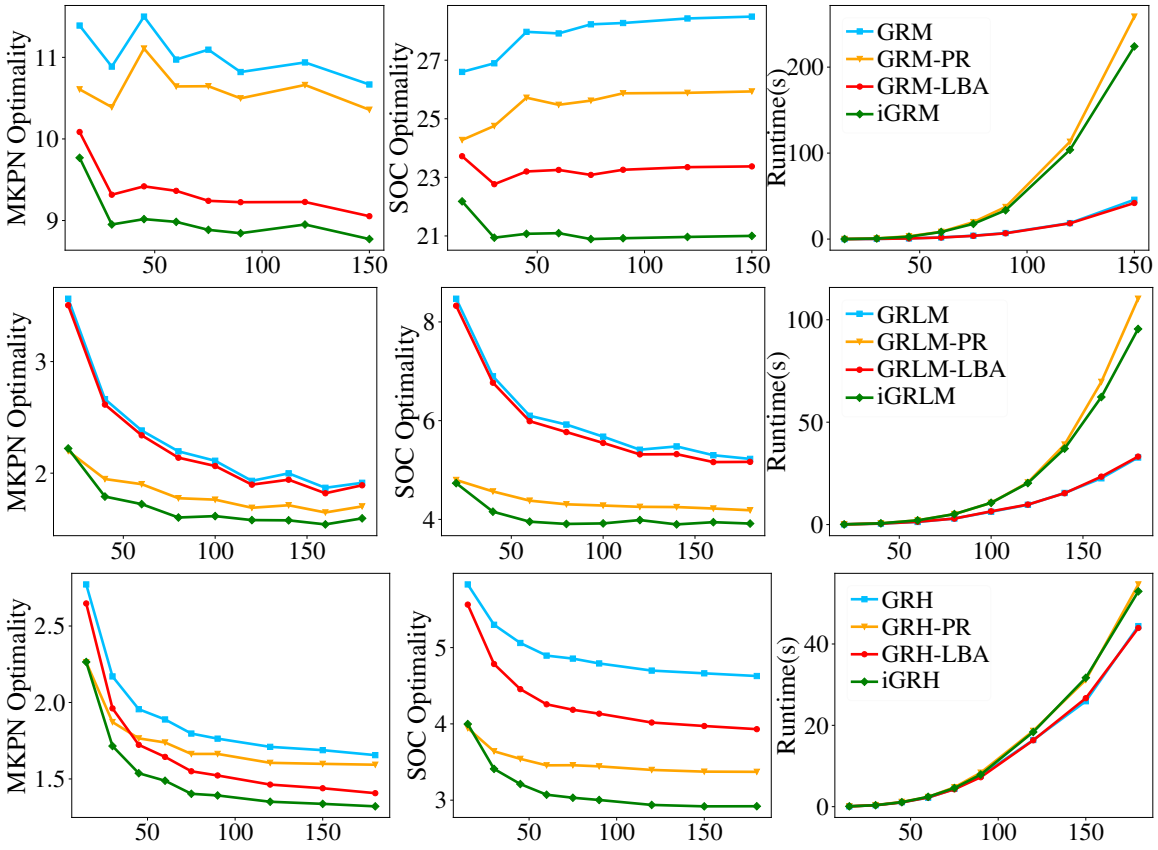


Figure 18: Effectiveness of heuristics in boosting GRA-based algorithms on $m \times m$ grids. For all figures, the x-axis is the grid side length m. Each row shows a specific algorithm (GRM, GRLM, and GRH). From left to right, makespan(MKPN) optimality ratio, SOC optimality ratio, and the computation time required for the path refinement routine are given.

Second, the combined effect of the two heuristics is nearly additive, confirming that the two heuristics are orthogonal two each other, as their designs indicate. The end result is a dramatic overall cross-the-board optimality improvement. As an example, for GRH, for the last data point, the makespan optimality ratio dropped from around 1.8 for the based to around 1.3 for iGRH. In terms of computational costs, the LBA heuristic adds negligible more time. The path refinement heuristic takes more time in full and $\frac{1}{2}$ density settings but adds little cost in the $\frac{1}{3}$ density setting.

7.5 Evaluations on 3D Grids

For the 3D setting, the performance and solution structure of our methods are largely similar to the 2D setting. As such, we provide basic evaluations for completeness, fixing the aspect ratio at $m_1 : m_2 : m_3 = 4 : 2 : 1$ and density at $\frac{1}{3}$, and examine GRH3D variants on obstacle-free grids with varying sizes. Here, because DDM only applies to 2D, we use ILP

with split heuristics (Yu & LaValle, 2016) instead of DDM. Start and goal configurations are randomly generated; the results are shown in Fig. 19. ILP with 16-split heuristic and EECBS compute solution with better optimality ratio but does not scale. In contrast, GRH3D variants readily scale to grids with over $370,000$ vertices and $120,000$ agents. Optimality ratios for GRH variants decrease as the grid size increases, approaching 1.5-1.7.

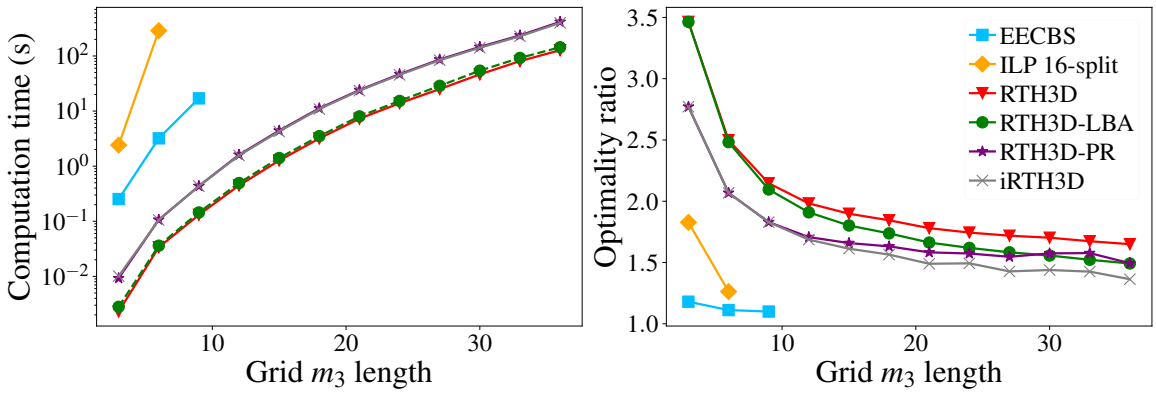


Figure 19: Computation time and optimality ratio for four methods on $m_1 \times m_2 \times m_3$ grids with varying grid size and $m_1 : m_2 : m_3 = 4 : 2 : 1$.

8. Conclusion and Discussion

In this study, we propose to apply Grid Rearrangements (Szegedy & Yu, 2023) to solve MAPF. A basic adaptation of GRA, with a more efficient line shuffle routine, enables solving MAPF on grids at maximum agent density, in polynomial time, with a previously unachievable optimality guarantee. Then, combining GRA, a highway heuristic, and additional matching heuristics, we obtain novel polynomial time algorithms that are provably asymptotically $1 + \frac{m_2}{m_1+m_2}$ makespan-optimal on $m_1 \times m_2$ grids with up to $\frac{1}{3}$ agent density, with high probability. Similar guarantees are also achieved with the presence of obstacles and at an agent density of up to one-half. These results in 2D are then shown to readily generalize to 3D and higher dimensions. In practice, our methods can solve problems on 2D graphs with over 10^5 number of vertices and 4.5×10^4 agents to 1.26 makespan-optimal (which can be better with a larger $m_1 : m_2$ ratio). Scalability is even better in 3D. To our knowledge, no previous MAPF solvers provide dual guarantees on low-polynomial running time and practical optimality.

Limitation While the GRA excels in achieving reasonably good optimality within polynomial time for dense instances, it does have limitations when applied to scenarios with a small number of robots or instances that are inherently easy to solve. In such cases, although GRA remains functional, its performance may lag behind other algorithms, and its solutions might be suboptimal compared to more specialized or efficient approaches. The algorithm's strength lies in its ability to efficiently handle complex, densely populated scenarios, leveraging its unique methodology. However, users should be mindful of its relative performance in simpler instances where alternative algorithms may offer superior solutions.

This limitation underscores the importance of considering the specific characteristics of the robotic system and task at hand when choosing an optimization algorithm, tailoring the selection to match the intricacies of the given problem instance.

Our study opens the door for many follow-up research directions; we discuss a few here.

New line shuffle routines. Currently, GRLM and GRH only use two/three rows to perform a simulated row shuffle. Among other restrictions, this requires that the sub-grids used for performing simulated shuffle be well-connected (i.e. obstacle-free or the obstacles are regularly spaced so that there are at least two rows that are not blocked by static obstacles in each motion primitive to simulate the shuffle). Using more rows or irregular rows in a simulated row shuffle, it is possible to accommodate larger obstacles and/or support density higher than one-half.

Better optimality at lower agent density. It is interesting to examine whether further optimality gains can be realized at lower agent density settings, e.g., $\frac{1}{9}$ density or even lower, which are still highly practical. We hypothesize that this can be realized by somehow merging the different phases of GRA so that some unnecessary agent travel can be eliminated after computing an initial plan.

Consideration of more realistic robot models. The current study assumes a unit-cost model in which an agent takes a unit amount of time to travel a unit distance and allows turning at every integer time step. In practice, robots will need to accelerate/decelerate and also need to make turns. Turning can be especially problematic and cause a significant increase in plan execution time if the original plan is computed using the unit-cost model mentioned above. We note that GRH returns solutions where robots move in straight lines most of the time, which is advantageous compared to all existing MAPF algorithms, such as ECBS and DDM, which have many directional changes in their computed plans. it would be interesting to see whether the performance of GRA-based MAPF algorithms will further improve as more realistic robot models are adapted.

Remark 1. *Currently, our* GRA*-based* MAPF *solves are limited to a static setting whereas e-commerce applications of multi-agent motion planning often require solving life-long settings (Ma, et al., 2017b). The metric for evaluating life-long* MAPF *is often the* throughput*, namely the number of goals reached per time step. We note that* GRH *also provides optimality guarantees for such settings, e.g., for the setting where $m_1 = m_2 = m$, the direct application of* GRH *to large-scale life-long* MAPF *on square grids yields an optimality ratio of $\frac{2}{9}$ on throughput.*

We may solve life-long MAPF *using* GRH *in* batches*. For each batch with n agents,* GRH *takes about $3m$ steps; the throughput is then $\mathcal{T}_{RTH} = \frac{n}{3m}$. As for the lower bound estimation of the throughput, the expected Manhattan distance in an $m \times m$ square, ignoring inter-agent collisions, is $\frac{2m}{3}$. Therefore, the lower bound throughput for each batch is $\mathcal{T}_{lb} = \frac{3n}{2m}$. The asymptotic optimality ratio is $\frac{\mathcal{T}_{RTH}}{\mathcal{T}_{lb}} = \frac{2}{9}$. The $\frac{2}{9}$ estimate is fairly conservative because* GRH *supports much higher agent densities not supported by known life-long* MAPF *solvers. Therefore, it appears very promising to develop an optimized Grid Rearrangement-inspired algorithms for solving life-long* MAPF *problems.*

Acknowledgments

This work is supported in part by NSF awards IIS-1845888, CCF-1934924, IIS-2132972, and CCF-2309866, and an Amazon Research Award.

References

Banfi, J., Basilico, N., & Amigoni, F. (2017). Intractability of time-optimal multirobot path planning on 2d grid graphs with holes. *IEEE Robotics and Automation Letters*, *2*(4), 1941–1947.

Barer, M., Sharon, G., Stern, R., & Felner, A. (2014). Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Seventh Annual Symposium on Combinatorial Search*.

Bitton, D., DeWitt, D. J., Hsaio, D. K., & Menon, J. (1984). A taxonomy of parallel sorting. *ACM Computing Surveys (CSUR)*, *16*(3), 287–318.

Burkard, R., Dell'Amico, M., & Martello, S. (2012). *Assignment problems: revised reprint*. SIAM.

Damani, M., Luo, Z., Wenzel, E., & Sartoretti, G. (2021). Primal _2: Pathfinding via reinforcement and imitation multi-agent learning-lifelong. *IEEE Robotics and Automation Letters*, *6*(2), 2666–2673.

De Wilde, B., Ter Mors, A. W., & Witteveen, C. (2014). Push and rotate: a complete multi-agent pathfinding algorithm. *Journal of Artificial Intelligence Research*, *51*, 443–492.

Dekhne, A., Hastings, G., Murnane, J., & Neuhaus, F. (2019). Automation in logistics: Big opportunity, bigger uncertainty. In *McKinsey Q*, pp. 1–12.

Demaine, E. D., Fekete, S. P., Keldenich, P., Meijer, H., & Scheffer, C. (2019). Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch. *SIAM Journal on Computing*, *48*(6), 1727–1762.

Erdem, E., Kisa, D. G., Oztok, U., & Schüller, P. (2013). A general formal framework for pathfinding problems with multiple agents. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*.

Erdmann, M., & Lozano-Perez, T. (1987). On multiple moving objects. *Algorithmica*, *2*(1), 477–521.

Ford, L. R., & Fulkerson, D. R. (1956). Maximal flow through a network. *Canadian journal of Mathematics*, *8*, 399–404.

Goel, A., Kapralov, M., & Khanna, S. (2013). Perfect matchings in o(n\logn) time in regular bipartite graphs. *SIAM Journal on Computing*, *42*(3), 1392–1404.

Goldenberg, M., Felner, A., Stern, R., Sharon, G., Sturtevant, N., Holte, R. C., & Schaeffer, J. (2014). Enhanced partial expansion a. *Journal of Artificial Intelligence Research*, *50*, 141–187.

Goldreich, O. (2011). Finding the shortest move-sequence in the graph-generalized 15-puzzle is np-hard. In *Studies in complexity and cryptography. Miscellanea on the interplay between randomness and computation*, pp. 1–5. Springer.

Guo, T., Feng, S. W., & Yu, J. (2022). Polynomial time near-time-optimal multi-robot path planning in three dimensions with applications to large-scale uav coordination. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10074–10080.

Guo, T., Han, S. D., & Yu, J. (2021). Spatial and temporal splitting heuristics for multi-robot motion planning. In *IEEE International Conference on Robotics and Automation*.

Guo, T., & Yu, J. (2022). Sub-1.5 Time-Optimal Multi-Robot Path Planning on Grids in Polynomial Time. In *Proceedings of Robotics: Science and Systems*, New York City, NY, USA. DOI: 10.15607/RSS.2022.XVIII.057.

Gurobi Optimization, LLC (2021). Gurobi Optimizer Reference Manual.. `https://www.gurobi.com`.

Hall, P. (2009). On representatives of subsets. In *Classic Papers in Combinatorics*, pp. 58–62. Springer.

Han, S. D., Rodriguez, E. J., & Yu, J. (2018). Sear: A polynomial-time multi-robot path planning algorithm with expected constant-factor optimality guarantee. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–9. IEEE.

Han, S. D., & Yu, J. (2020). Ddm: Fast near-optimal multi-robot path planning using diversified-path and optimal sub-problem solution database heuristics. *IEEE Robotics and Automation Letters*, *5*(2), 1350–1357.

Hönig, W., Preiss, J. A., Kumar, T. S., Sukhatme, G. S., & Ayanian, N. (2018). Trajectory planning for quadrotor swarms. *IEEE Transactions on Robotics*, *34*(4), 856–869.

Hopcroft, J. E., Schwartz, J. T., & Sharir, M. (1984). On the complexity of motion planning for multiple independent objects; pspace-hardness of the" warehouseman's problem". *The International Journal of Robotics Research*, *3*(4), 76–88.

Kornhauser, D., Miller, G., & Spirakis, P. (1984). Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *Proceedings IEEE Symposium on Foundations of Computer Science*, pp. 241–250.

Lam, E., Le Bodic, P., Harabor, D. D., & Stuckey, P. J. (2019). Branch-and-cut-and-price for multi-agent pathfinding.. In *IJCAI*, pp. 1289–1296.

Leighton, T., & Shor, P. (1989). Tight bounds for minimax grid matching with applications to the average case analysis of algorithms. *Combinatorica*, *9*(2), 161–187.

Li, J., Chen, Z., Harabor, D., Stuckey, P., & Koenig, S. (2021a). Anytime multi-agent path finding via large neighborhood search. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Li, J., Ruml, W., & Koenig, S. (2021b). Eecbs: A bounded-suboptimal search for multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

Li, J., Tinka, A., Kiesel, S., Durham, J. W., Kumar, T. S., & Koenig, S. (2020). Lifelong multi-agent path finding in large-scale warehouses.. In *AAMAS*, pp. 1898–1900.

Li, Q., Lin, W., Liu, Z., & Prorok, A. (2021). Message-aware graph attention networks for large-scale multi-robot path planning. *IEEE Robotics and Automation Letters*, *6*(3), 5533–5540.

LogisticsIQ (2020). Warehouse automation market with post-pandemic (covid-19) impact by technology, by industry, by geography - forecast to 2026. `https://www.researchandmarkets.com/r/s6basv`. Accessed: 2022-01-05.

Luna, R. J., & Bekris, K. E. (2011). Push and swap: Fast cooperative path-finding with completeness guarantees. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

Ma, H., Harabor, D., Stuckey, P. J., Li, J., & Koenig, S. (2019). Searching with consistent prioritization for multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, pp. 7643–7650.

Ma, H., & Koenig, S. (2016). Optimal target assignment and path finding for teams of agents. In *AAMAS*.

Ma, H., Kumar, T. S., & Koenig, S. (2017a). Multi-agent path finding with delay probabilities. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 31.

Ma, H., Li, J., Kumar, T. K. S., & Koenig, S. (2017b). Lifelong multi-agent path finding for online pickup and delivery tasks. In *AAMAS*.

Mason, R. (2019). Developing a profitable online grocery logistics business: Exploring innovations in ordering, fulfilment, and distribution at ocado. In *Contemporary Operations and Logistics*, pp. 365–383. Springer.

Okumura, K. (2023). Lacam: Search-based algorithm for quick multi-agent pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37, pp. 11655–11662.

Okumura, K., Machida, M., Défago, X., & Tamura, Y. (2019). Priority inheritance with backtracking for iterative multi-agent path finding. In *IJCAI*.

Okumura, K., Tamura, Y., & Défago, X. (2021). Iterative refinement for real-time multi-robot path planning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9690–9697. IEEE.

Perron, L., & Furnon, V. Or-tools.. `https://developers.google.com/optimization/`.

Poduri, S., & Sukhatme, G. S. (2004). Constrained coverage for mobile sensor networks. In *Proceedings IEEE International Conference on Robotics & Automation*.

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

Rus, D., Donald, B., & Jennings, J. (1995). Moving furniture with teams of autonomous robots. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots & Systems*, pp. 235–242.

Sartoretti, G., Kerr, J., Shi, Y., Wagner, G., Kumar, T. S., Koenig, S., & Choset, H. (2019). Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, *4*(3), 2378–2385.

Sharon, G., Stern, R., Felner, A., & Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, *219*, 40–66.

Sharon, G., Stern, R., Goldenberg, M., & Felner, A. (2013). The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, *195*, 470–495.

Shor, P. W., & Yukich, J. E. (1991). Minimax grid matching and empirical measures. *The Annals of Probability*, *19*(3), 1338–1348.

Silver, D. (2005). Cooperative pathfinding.. *Aiide*, *1*, 117–122.

Stern, R., Sturtevant, N. R., Felner, A., Koenig, S., Ma, H., Walker, T. T., Li, J., Atzmon, D., Cohen, L., Kumar, T. S., et al. (2019). Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Twelfth Annual Symposium on Combinatorial Search*.

Surynek, P. (2009). A novel approach to path planning for multiple robots in bi-connected graphs. In *2009 IEEE International Conference on Robotics and Automation*, pp. 3613–3619. IEEE.

Surynek, P. (2010). An optimization variant of multi-robot path planning is intractable. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 24.

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

Szegedy, M., & Yu, J. (2023). Rubik tables and object rearrangement. *The International Journal of Robotics Research*, *42*(6), 459–472.

Wagner, G., & Choset, H. (2015). Subdimensional expansion for multirobot path planning. *Artificial intelligence*, *219*, 1–24.

Wan, Q., Gu, C., Sun, S., Chen, M., Huang, H., & Jia, X. (2018). Lifelong multi-agent path finding in a dynamic environment. In *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp. 875–882. IEEE.

Wurman, P. R., D'Andrea, R., & Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, *29*(1), 9–9.

Yu, J. (2015). Intractability of optimal multirobot path planning on planar graphs. *IEEE Robotics and Automation Letters*, *1*(1), 33–40.

Yu, J. (2018). Constant factor time optimal multi-robot routing on high-dimensional grids. In *2018 Robotics: Science and Systems*.

Yu, J., & LaValle, M. (2012). Distance optimal formation control on graphs with a tight convergence time guarantee. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pp. 4023–4028. DOI: 10.1109/CDC.2012.6426233.

Yu, J., & LaValle, S. M. (2013a). Multi-agent path planning and network flow. In *Algorithmic foundations of robotics X*, pp. 157–173. Springer.

Yu, J., & LaValle, S. M. (2013b). Structure and intractability of optimal multi-robot path planning on graphs. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*.

Yu, J., & LaValle, S. M. (2016). Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Transactions on Robotics*, *32*(5), 1163–1177.