On Computing Makespan-Optimal Solutions for Generalized Sliding-Tile Puzzles

Marcus Gozon University of Michigan

Jingjin Yu Rutgers University

Abstract

In the 15-puzzle game, 15 labeled square tiles are reconfigured on a 4×4 board through an escort, wherein each (time) step, a single tile neighboring it may slide into it, leaving the space previously occupied by the tile as the new escort. We study a generalized sliding-tile puzzle (GSTP) in which (1) there are $1+$ escorts and (2) multiple tiles can move synchronously in a single time step. Compared with popular discrete multi-agent/robot motion models, GSTP provides a more accurate model for a broad array of high-utility applications, including warehouse automation and autonomous garage parking, but is less studied due to the more involved tile interactions. In this work, we analyze optimal GSTP solution structures, establishing that computing makespan-optimal solutions for GSTP is NP-complete and developing polynomial time algorithms yielding makespans approximating the minimum with expected/high probability constant factors, assuming randomized start and goal configurations.

1 Introduction

The 15-puzzle (Loyd 1959) is a sliding-tile puzzle in which fifteen interlocked square tiles, labeled 1-15, and an empty escort square are located on a 4×4 square game board (see Fig. 1). In each time step, a tile neighboring the escort may slide into it, leaving an empty square that becomes the new escort. The game's goal is to reconfigure the tiles to realize a row-major ordering of the labeled tiles. We study a natural generalization of the 15-puzzle, in which the game board is an arbitrarily large rectangular grid with $1+$ escorts. In addition, tiles can move synchronously in a given time step assuming no collision under uniform movement. We call this problem the *generalized sliding-tile puzzle* or GSTP.

GSTP provides a high-fidelity discretized model for multi-robot applications operating in grid-like environments, including the efficient coordination of a large number of robots in warehouses for order fulfillment (Wurman, D'Andrea, and Mountz 2008; Mason 2019), motion planning in autonomous parking garages (Guo and Yu 2023), and so on. A particularly important feature of GSTP is that, given two neighboring tiles sharing a side, one tile may only move in the direction toward the second tile if the second tile moves in the same direction. Otherwise, if the second tile moves in a perpendicular direction, a collision occurs, which we call the *corner following constraint* or CFC. Consideration of CFC renders GSTP different from popular multi-agent/robot pathfinding (MAPF) problems (Stern et al. 2019) in which a classical formulation allows the second tile to move in a direction perpendicular to the moving direction of the first tile. Ignoring CFC significantly reduces the steps required to solve a tile reconfiguration problem, making computing optimal solutions less challenging, but is less accurate in modeling many real-world applications.

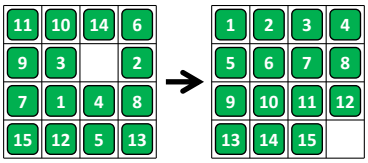


Figure 1: Start and goal configurations of a 15-puzzle instance. In GSTP, there can be $1+$ escorts and multiple tiles may move synchronously, e.g., tile 3 and 9 may move to the right in a single step in the left configuration.

Given the strong connections between GSTP and today's grid-based multi-robot applications seeking ever more optimal solutions, we must have a firm grasp on the fundamental optimality structure of GSTP. Towards achieving such an understanding, this work studies the induced optimality structure in computing makespan-optimal solutions for GSTP, and brings forth the following main contributions:

- We establish that computing makespan-optimal solutions for GSTP is NP-complete with or without an enclosing grid. The problem remains NP-complete when there are $\lfloor |G|^{\epsilon} \rfloor$ escorts, where $|G|$ is the grid size (i.e., the total number of grid cells) and $0 < \epsilon < 1$ is a constant.
- We establish tighter makespan lower bounds for GSTP for all possible numbers of escorts. On an $m_1 \times m_2$ grid with k escorts, in expectation, solving GSTP requires $\Omega(\frac{m_1 m_2}{k})$ steps for $1 \leq k < \min(m_1, m_2)$ and $\Omega(m_1 + m_2)$ steps for $k \geq \min(m_1, m_2)$.
- We establish tighter makespan upper bounds for GSTP for all possible numbers of escorts that match the corresponding makespan lower bounds, asymptotically, thus closing the makespan optimality gap for GSTP. This

leverages a key intermediate result showing that GSTP instances on $2 \times m$ and $3 \times m$ grids can be solved in $O(m)$ steps. For all upper bounds, via careful analysis, we further provide a constant factor that is relatively low, considering CFC's severe restrictions on tile movements.

Some proofs are sketched or omitted; see supplementary materials for additional details.

2 Related Work

Modern studies on MAPF and related problems originated from the investigation of the generalization of the 15-puzzle (Loyd 1959) to the $(N^2 - 1)$-puzzle, with work addressing both computational complexity (Ratner and Warmuth 1990) and the computation of optimal solutions (Culberson and Schaeffer 1994, 1998). Gradually, graph-theoretic abstractions emerged that introduced non-grid-based environments and allowed more escorts (i.e., there can be more than one empty vertex on the underlying graph). Whereas such problems are solvable in polynomial time if only a feasible solution is desired (Kornhauser, Miller, and Spirakis 1984; Auletta et al. 1999; Yu 2013), computing optimal solutions are generally NP-hard (Wilson 1974; Goldreich 2011; Surynek 2010; Yu and LaValle 2013; Demaine et al. 2019). With the graph-based generalization, CFC is generally not enforced as the geometric constraint lengthens a motion plan and complicates the reasoning.

Due to its close relevance to a great many high-impact applications, e.g., game AI (Pottinger 1999), warehouse automation (Wurman, D'Andrea, and Mountz 2008; Mason 2019), great interests started to develop in quickly computing (near-)optimal solutions for MAPF (Silver 2005). With this development, a variant of the $(N^2 - 1)$-puzzle was introduced, which does not require the presence of escorts (Standley 2010). In other words, in the most well-studied MAPF formulation, any non-self-intersecting chain of agents may potentially move synchronously, one following another, in a single step. In (Standley 2010), a bi-level algorithmic solution framework, *operator decomposition* (OD) + *independence detection* (ID), is built upon the general idea of *decoupling* (Erdmann and Lozano-Perez 1987), which treats each agent individually as if other agents do not exist and handles agent-agent interactions on demand. A super-majority of modern MAPF methods have generally adopted a bi-level decoupling search approach. Representative work along this line includes *increasing cost-tree search* (ICTS) (Sharon et al. 2013), *conflict-based search* (CBS) and variants (Sharon et al. 2015; Barer et al. 2014; Li, Ruml, and Koenig 2021), *priority inheritance with backtracking* (PIBT) (Okumura et al. 2022), and most recently, *lazy constraints addition search for MAPF* (LaCAM) (Okumura 2023). Besides search-driven methods, reduction-based approaches have also been proposed (Surynek 2012; Erdem et al. 2013; Yu and LaValle 2016).

In contrast, MAPF formulations similar to GSTP, i.e., considering CFC, have received relatively muted attention. On the side of computational complexity, besides the hardness result of the $(N^2 - 1)$-puzzle (Ratner and Warmuth 1990) and a recent followup (Demaine and Rudoy 2018), it has been shown that computing total distance-optimal solutions with CFC is NP-complete in environments with specially crafted obstacles (Geft and Halperin 2022). We note that sliding-tile puzzles can easily become PSPACE-hard in non-grid-based settings (Hopcroft, Schwartz, and Sharir 1984), even for unlabeled tiles (Solovey and Halperin 2015). While of practical importance, hardness for computing optimal solutions for GSTP in obstacle-free settings has not been established. On the side of computational efforts in addressing GSTP, CFC has been studied partially as part of k-robustness (Atzmon et al. 2018). A recent SoCG competition has been held (Fekete et al. 2022) that addresses exactly the GSTP problem but with a focus on computing solutions for a set of benchmark problems. A variation of GSTP was studied in (Guo and Yu 2023) targeting autonomous parking garage applications. These computational studies largely leave unanswered fundamental questions on GSTP, including computational complexity and optimality bounds.

3 Preliminaries

3.1 The Generalized Sliding-Tile Puzzle

In the *generalized sliding-tile puzzle* (GSTP), on a rectangular $m_1 \times m_2$ grid $G = (V, E)$ lies $n < m_1 m_2$ tiles, uniquely labeled $1, \ldots, n$. A *configuration* of the tiles is an injective mapping from $\{1, \ldots, n\} \rightarrow V = \{(v_y, v_x)\}$ where $1 \leq v_y \leq m_1$ and $1 \leq v_x \leq m_2$. Tiles must be reconfigured from a random configuration $\mathcal{S} = \{s_1, \ldots, s_n\}$ to some goal configuration $\mathcal{G} = \{g_1, \ldots, g_n\}$, usually a row-major ordering of the tiles, subject to certain constraints. Specifically, let the *path* of tile i, $1 \leq i \leq n$, be $p_i : \mathbb{N}_0 \rightarrow V$, and so GSTP seeks a *feasible path set* $P = \{p_1, \ldots, p_n\}$ such that the following constraints are met for all $1 \leq i, j \leq n$, $i \neq j$ and $\forall t \geq 0$:

- Continuous uniform motion: $p_i(t+1) = p_i(t)$ or $(p_i(t+1), p_i(t)) \in E$,
- Completion: $p_i(0) = s_i$ and $p_i(T) = g_i$ for some $T \geq 0$,
- No meet collision: $p_i(t) \neq p_j(t)$,
- No head-on collision: $(p_i(t) = p_j(t+1) \wedge p_i(t+1) = p_j(t)) = false$,
- Corner-following constraint: let $e_i(t) = p_i(t+1) - p_i(t)$ be the movement direction vector. If $p_i(t+1) = p_j(t)$, then $e_i(t) \not\perp e_j(t)$.

Let T_P be the smallest $T \geq 0$ such that the completion constraint is met for a given path set P. Naturally, it is desirable to compute P with minimum T_P. We define the decision version of makespan-optimal GSTP as follows.

MOGSTP
INSTANCE: A GSTP instance and a positive integer K.
QUESTION: Is there a feasible path set P with $T_P \leq K$?

3.2 2/2/4-SAT

We will need a specialized SAT instance called **2/2/4-SAT** for our hardness result, defined as follows.

2/2/4-SAT
INSTANCE: A boolean satisfiability instance with n variables $x_1, \ldots x_n$ and n clauses $c_1, \ldots c_n$. Each clause c_j has

4 literals, and each variable x_i appears across all clauses exactly 4 times in total, twice negated and twice unnegated.
QUESTION: Is there an assignment to $x_1, \ldots, x_n$ such that each clause c_i has exactly two true literals?

2/2/4-SAT was shown to be NP-complete in (Ratner and Warmuth 1990), which is subsequently employed to show the hardness of the (N^2-1)-puzzle.

3.3 Feasibility and Known Makespan Bounds

It is well-known that the (N^2-1)-puzzle may not always have a solution (Loyd 1959) due to the configurations forming two connected graphs. More formally, it can be shown that the configurations of an (N^2-1)-puzzle are partitioned into two groups, each of which is isomorphic to the *alternating group* A_{N^2-1} (Wilson 1974). Because moves on a GSTP instance on an $N \times N$ grid with a single escort can be "slowed down" to equivalent moves on an (N^2-1)-puzzle, they share the same feasibility. The same remains true for rectangular grids. Checking feasibility can be performed in linear time (Wilson 1974). On the other hand, also clear from (Wilson 1974), when there are two or more escorts, a GSTP instance is always feasible. To summarize,

Lemma 3.1. GSTP *with a single escort may be infeasible. The feasibility of* GSTP *with a single escort can be checked in linear time.* GSTP *with two or more escorts is feasible.*

Given a feasible (N^2-1)-puzzle, each tile can be moved to its goal in $O(N)$ steps since a tile is within $O(N)$ distance to its goal and $O(1)$ steps are needed to switch two tiles. This suggests an $O(N^3)$ algorithm, which readily extends to an $O(m_1 m_2 \max(m_1, m_2))$ step algorithm on an $m_1 \times m_2$ grid. This is also an upper bound for GSTP with a single escort. GSTP with more escorts is studied in the context of automated garages (Guo and Yu 2023), with results on $\Theta(m_1 m_2)$ escorts and $(2m_1 + 2m_2 - 4)$ escorts. To summarize, the following is known:

Number of escorts	Makespan upper bound
1	$O(m_1 m_2 \max(m_1, m_2))$
$(2m_1 + 2m_2 - 4)$	$O(m_1 m_2)$
$\Theta(m_1 m_2)$	$O(\max(m_1, m_2))$

It is easy to see that $\Omega(\max(m_1, m_2))$ is a makespan lower bound in expectation. It can be shown that the makespan lower bound is close to (m_1+m_2) with high probability when there are $\Omega(m_1 m_2)$ tiles (Guo and Yu 2022).

3.4 The Rubik Table Algorithm

A notable tool, Rubik tables (Szegedy and Yu 2023), has been applied to derive polynomial-time, $1.x$-optimal solutions to classical MAPF problems on grids (Guo and Yu 2022). This tool will also be employed in this work. We will use the following theorem with an associated algorithm.

Theorem 3.2 (Rubik Table Algorithm for 2D Grids (Szegedy and Yu 2023))**.** *Let an $m_1 \times m_2$ grid be filled with tiles labeled $1, \ldots, m_1 m_2$. A row (resp., column) shuffle can arbitrarily permute a row (resp., column) of tiles. Then, the tiles can be rearranged from any configuration to the row-major configuration using m_1 row shuffles, followed by m_2 column shuffles, and then m_1 row shuffles. Alternatively, the tiles can be rearranged using m_2 column shuffles, followed by m_1 row shuffles, and then another m_2 column shuffles.*

Fig. 2 illustrates running the Rubik table algorithm over a 4×3 grid, using a row-column-row shuffle sequence.

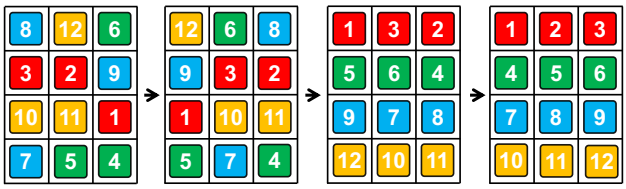


Figure 2: Applying the Rubik table algorithm to rearrange tiles on a 4×3 grid using a sequence of row shuffles, followed by column shuffles, followed by row shuffles.

4 Intractability of MOGSTP

We proceed in this section to establish the NP-completeness of **MOGSTP** on square grids, which will show

Theorem 4.1. MOGSTP *is NP-complete, with or without an enclosing grid.*

First, we sketch the proof to provide key ideas behind the reduction of hardness. Then, detailed constructions of the required gadgets and the full instance construction follow.

4.1 Proof Outline

We prove via a reduction from **2/2/4-SAT** (Ratner and Warmuth 1990) defined in Sec. 3.2. Our reduction constructs an **MOGSTP** instance to force a flow of literal tiles from variable gadgets to clause gadgets in matching pairs, forming a truth side of literals and a false side of literals (realized through a *gadget train*, see Fig. 3(a) for a sketch and explanation). For each variable x_i, $1 \leq i \leq n$, there are four sliding tiles labeled $x_i^1, x_i^2, \bar{x}_i^1, \bar{x}_i^2$ that correspond to the four literals for x_i, the first pair positive and the second pair negative. When the context is clear, we simply say *literals* instead of *literal tiles*. A variable gadget (see Fig. 3(b) and Fig. 5) is constructed that forces the pair of unnegated literals (e.g., x_i^1 and x_i^2, "+" tiles in the figure) to only exit together from one side of the gadget (e.g., left) while forcing the pair of negated literal (e.g., $\bar{x}_i^1$ and $\bar{x}_i^2$, , "-" tiles in the figure) to exit together from the opposite side, each passing through limited openings of the *rails* that flank the train and move in the opposite direction. After all $4n$ literals exit from the n variable gadgets, there are $2n$ each on the left and right side of the rails. These literals are then routed into clause gadgets (see Fig. 3(c) and Fig. 6), each allowing at most two literals to enter from each side. The overall **MOGSTP** instance is constructed such that if the **2/2/4-SAT** is satisfiable, then in the **MOGSTP**, the $2n$ literal tiles that move to the left side of the train can be chosen to be the true literals in the given truth assignment, and so all literal tiles can then be readily routed to the clause gadgets. Similarly, in the other direction of the reduction, because exactly n pairs of literal tiles must be on the left side in a makespan-optimal solution, the

corresponding $2n$ literals can be set to positive to satisfy the **2/2/4-SAT** instance.

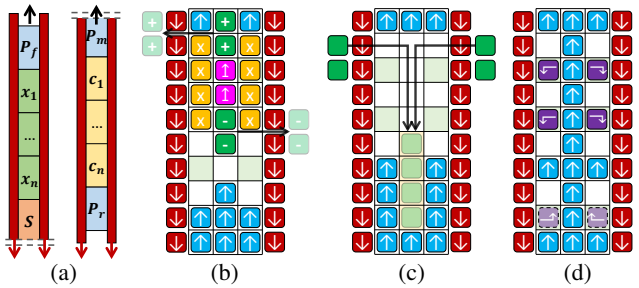


Figure 3: Pieces of **MOGSTP**. (a) Sketch of the train-like **MOGSTP** instance split into two halves. The upward-moving gadget train is surrounded by two (red) rails of tiles that move strictly downwards, with a few gaps (not shown here, see Fig. 4) to allow tiles to exit/enter. The train, from top to bottom, contains a front padding car P_f, variable cars $x_1, \ldots, x_n$, a security car S, a middle padding car P_m, clause cars $c_1, \ldots, c_n$, and a rear padding car P_r. (b) A variable gadget (center 10×3 portion) is constructed to force unnegated ("+") and negated ("-") literal tiles to exit from different sides. The exited titles will be outside the rails. (c) A clause gadget is constructed to allow at most two literals to come in from each side of the rails. (d) The security car where the upper four purple tiles will exit to block variable exits on the rails (see Fig. 4). The lower two light purple blocks are goals for two tiles initially on the rails (Fig. 4).

4.2 Gadgets

Our gadgets consist of *preset tiles* that move in a fixed direc-

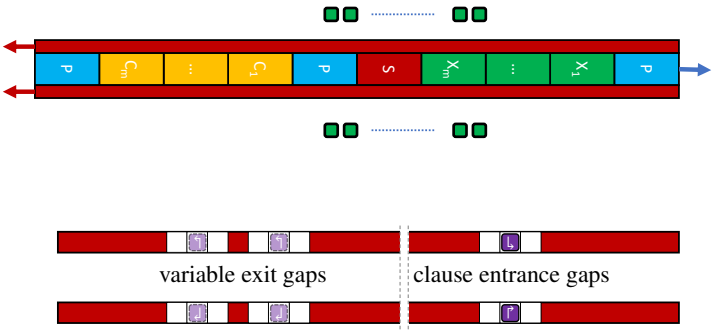


Figure 4: Part of the rails, rotated 90 degrees clockwise from Fig. 3(a), showing the variable exit and clause entrance gaps.

Variable Car (Gadget) A variable car ($x_1, \ldots, x_n$ blocks in Fig. 3) is an upwards moving 10×3 block whose start configuration is shown in Fig. 3(b). For the variable car corresponding to x_i, besides the (blue) up tiles as marked, there are two unnegated literal tiles (the two "+" tiles) corresponding to x_i^1 and x_i^2, and two negated literal tiles (the two "-" tiles) corresponding to $\bar{x}_i^1$ and $\bar{x}_i^2$. These tiles must be moved to some clause cars to be introduced shortly. There are also (pink) *single-delay* up tiles that must pause in place exactly once throughout the execution of the **MOGSTP** instance. Additionally, there are eight obstacle tiles (the x tiles) whose goal configurations are three spots lower within the same variable car. These obstacle tiles help ensure that the pairs of positive and negative literals split up onto different sides.

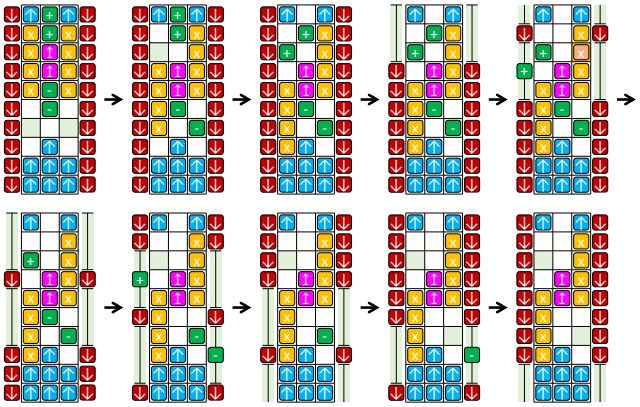


Figure 5: Illustration of how (green) literal tiles may exit a variable gadget in pairs. The bottom left subfigure shows the four lower gaps on the rails, each a 3×1 block.

Lemma 4.2. *As a variable car passes by the variable exits on the rails, the positive and negative literals can only exit to different sides of the rails.*

Proof Sketch. Only literal and obstacle tiles may move outside a variable car (the 10×3 grid). It can be shown that obstacle tiles should not change columns. Because of this, unnegated (resp., negated) literals can only exit from the 3th (resp., 7th) row. This forces the obstacle tiles to become asymmetric on the two sides of a variable car, resulting in the unnegated literals exiting from one side of the car and the negated literals exiting from the opposite side. One such exit sequence is illustrated in Fig. 5. □

Clause Car (Gadget) As shown in Fig. 3(c), the clause car is an upward-moving 10×3 subgrid entirely composed of up tiles and requires 4 literal tiles corresponding to c_i in the goal configuration. With two symmetric 3×1 gaps on the rail, it is clear that at most two literals can enter from each side, as shown in Fig. 6.

Security Car (Gadget) Shown in Fig. 3(d), the security car is an upward-moving 10×3 subgrid whose frame consists of up tiles with four additional purple tiles at rows 3 and 5 that need to be injected into the middle of the four variable exits on the rails (see Fig. 4 for reference) as they pass by.

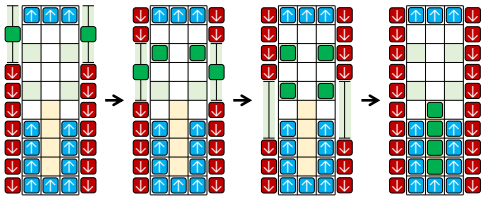


Figure 6: Green literal tiles entering a clause car gadget.

This prevents these gaps from being used by clause gadgets. It will receive the two purple tiles that are initially inside the clause entrances (see Fig. 4) to go to row 8.

4.3 Complete Specification and Reductions

We construct the **MOGSTP** instance as follows. Let $d = 24n+38$; our upwards moving train is a $d\times 3$ block; from top to bottom, the three padding cards have 4, $4n$, and 24 rows of up tiles, respectively. The middle padding car allows the literal tiles to reorder before entering the clause cars. Initially, the bottoms of the rails are aligned with the bottom of the gadget train. The *variable exit* openings occupy rows $d+2$ to $d+4$ and $d+6$ to $d+8$ from the bottom. The *clause entrance* opening occupy rows $d+14n+12$ to $d+14n+14$. While a grid is not needed, we can select our grid G to be of size $4d \times 4d$ with the construct positioned in the middle horizontally, with the bottom of the construct starting at the $(d+1)$th row from the bottom of G. The makespan bound K is set to d. The **MOGSTP** instance is fully specified.

Proof of Thm. 4.1. If the **2/2/4-SAT** instance is satisfiable, we can select the positive (resp., negative) literals to exit to the left (resp., right) of the rails from the variable cars, when they pass by the literal exit gaps. Then, these literals can reorder and enter into the clause gadgets as required, reaching the target goal configuration with a makespan of d.

Similarly, in the other direction, if the **MOGSTP** instance has a solution with a makespan of d, then the up/down tiles must move uninterrupted. In this case, four literals must exit a variable car in pairs of the same truth value to different sides. Subsequently, these literals reorder and enter the clause cars as described. Therefore, we can pick literal tiles on one side of the rails, e.g., left, and make their corresponding literals positive, ensuring all clauses are true. This yields a satisfying assignment for the **2/2/4-SAT** instance.

MOGSTP is in NP since the existence of a solution can be readily checked, and a feasible solution can be computed in polynomial time similar to how (N^2-1)-puzzles are solved. Thus, **MOGSTP** is NP-complete. □

MOGSTP remains NP-hard when we specify that there are exactly $\lfloor |G|^\epsilon \rfloor$, $0 < \epsilon < 1$ escorts (where $|G|$ is the number of cells of the grid) by blowing up the grid by a polynomial amount and filling the extra space with stationary tiles to achieve the desired number of escorts. Then note that **2/2/4-SAT** is still simulated through the movement of the literal tiles around the preset tiles, and in addition, a solution routing can be constructed in the same manner from a truth assignment.

5 Tighter Makespan Lower & Upper Bounds

In this section, we first establish a tighter makespan lower bound as a function of the number of escorts. Then, we proceed to the more involved efforts of deriving tighter makespan upper bounds again as a function of the available number of escorts. The new and tighter lower and upper bounds are summarized in the table below. We further provide an exact constant for all upper bounds as a more precise characterization. In all cases, our upper and lower bounds match asymptotically, eliminating the gaps left by previous studies on GSTP. All upper bounds come with low-polynomial-time algorithms for computing the actual plan, which is clear from the corresponding proofs.

k, the number of escorts	Makespan **lower** bound
$k < \min(m_1, m_2)$	exp. $\Omega(\frac{m_1 m_2}{k})$
$k \geq \min(m_1, m_2)$	h.p. $\Omega(\max(m_1, m_2))$
k, the number of escorts	Makespan **upper** bound
$k = 1$	$(81 + o(1))m_1 m_2$
$k = 2$	$(18 + o(1))m_1 m_2$
$2 < k < \min(m_1, m_2)$	$(22 + o(1))\frac{m_1 m_2}{\lfloor k/2 \rfloor}$
$k \geq m_1 + m_2 - 1$	$34 \max(m_1, m_2)$

Table 1: Our matching makespan lower and upper bounds.

For convenience, instead of viewing GSTP through batched tile movements, we focus on the movement of the escorts, which encodes tile motion more concisely. A straight contiguous train of tiles moving in a single step may be equivalently viewed as a *jump* of an escort. Since the new escort position must remain in the same row or column, we call the jump a *row jump* or *column jump*, respectively. In addition, we use *rectangular shift* or *r-shift*, as a fundamental motion primitive in which the escort cycles through the four corners of a rectangle, thus shifting all boundary elements by one tile in the opposite direction. We call the rectangular shift *cwr-shift* (resp., *ccwr-shift*) if the escort traverses the corners in the counterclockwise (resp., clockwise) direction.

5.1 Tighter Makespan Lower Bounds

Using escort jumps instead of tile moves lets us see immediately that a single time step can only change the sum of the Manhattan distances by $k \max(m_1, m_2)$, where k is the number of escorts. The observation readily leads to a tighter makespan lower bound than the previously established $\Omega(m_1 + m_2)$ (or $\Omega(\max(m_1, m_2))$).

Lemma 5.1. *The expected minimum makespan for* GSTP *on $m_1 \times m_2$ grids with k escorts is $\Omega(\frac{m_1 m_2}{k})$.*

Proof. Consider the sum of Manhattan distances S of each tile's start and goal positions. Over all possible start and goal configurations, each tile is expected to have a Manhattan distance of $\Omega(m_1 + m_2) = \Omega(\max(m_1, m_2))$ (Santaló 2004). Because there are $m_1 m_2 - k$ tiles, we have $S = \Omega((m_1 m_2 - k)\max(m_1, m_2))$, in expectation. Because each of the k escorts can jump a distance of $\max(m_1, m_2)$ within the same row/column, altering the Manhattan distance contribution by

1 for each tile in its jump path, in a single time step, S can only change by at most $k\max(m_1, m_2)$. Thus, at least $\frac{S}{k\max(m_1,m_2)} = \Omega(\frac{m_1 m_2}{k})$ steps are needed to solve the instance in expectation. □

Combining with the previously known lower bounds yields a tighter makespan lower bound of $\Omega(\frac{m_1 m_2}{k})$ for $k \leq \min(m_1, m_2)$ and $\Omega(\max(m_1, m_2))$ for $k \geq \min(m_1, m_2)$, which match our developed upper bounds, established next.

5.2 Tighter Makespan Upper Bounds: Outline

We leverage RTA (Sec. 3.4) to establish tighter makespan upper bounds for GSTP. Each round of row/column shuffles in RTA can be executed in parallel, potentially leading to a significantly reduced makespan. However, the shuffles do not readily translate to feasible sliding-tile motion; performing in-place permutation of tiles in a single row/column is impossible. To enable the application of RTA, instead working with one grid row/column, we simulate row/column shuffles by grouping multiple rows or columns together. Therefore, at a high level, we derive better upper bounds by:

- Applying RTA to obtain three batches of row or column shuffles (see, e.g., Fig. 2) with escorts treated as labeled tiles. Each batch of shuffles will be executed to completion (via simulations according to rules of GSTP) before the next batch is started.
- In a given batch of row/column shuffles, adjacent rows/columns will be grouped together (e.g., two or three rows per group), on which tile-sliding motions will be planned to realize the desired shuffles.

Performing efficient tile-sliding motions with the CFC constraint is key to establishing tighter upper bounds. We first describe subroutines for solving GSTP with 1 or 2 escorts on $3 \times m$ and $2 \times m$ grids. These subroutines will then be used to solve general GSTP instances.

5.3 Upper Bounds for 2-3 Rows with 1-2 Escorts

Our GSTP algorithms will build on subroutines for sorting multiple rows. We first prove such a routine on $3 \times m$ grids.

Lemma 5.2. *Feasible* GSTP *instances with a single escort on a* $3 \times m$ *grid can be solved in* $120m$ *steps.*

Proof. We give a procedure that sorts the right $\frac{1}{3}$ of the $3 \times m$ grid in $O(m)$ steps. A recursive application of the procedure then yields an overall $O(m + \frac{2}{3}m + \frac{4}{9}m + \ldots) = O(m)$ makespan.

To start, we move the escort to the bottom left corner for both the start and goal configurations, which takes 4 steps. These will be the new start/goal configurations. From here, for tiles on a $3 \times m$ grid, let B denote the set of tiles corresponding to the $\lfloor \frac{m-2}{3} \rfloor$ rightmost columns in the goal configuration. We refer to these tiles as B tiles and the rest as W tiles. We will treat the boundary cells as a *circular highway* moving clockwise and the inner middle line as a *workspace* to move B tiles to their destination. As an example, the B (resp., W) tiles are shown in dark gray (resp., light gray) in Fig. 7(b)-(g). The algorithm operates in three stages: (1) move B tiles to the highway, (2) arrange B tiles properly in the workspace, and (3) move B tiles to goals.

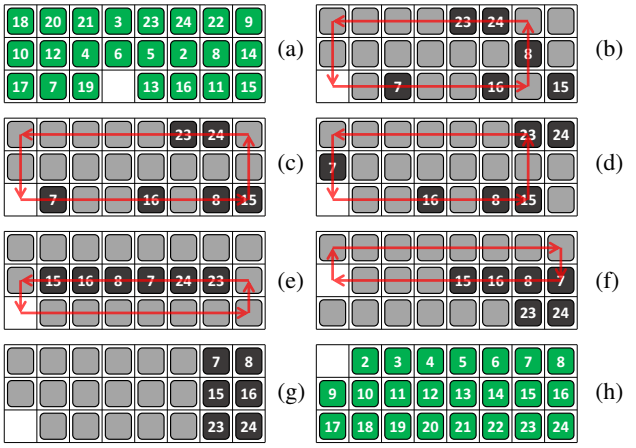


Figure 7: Sorting right $\frac{1}{3}$ on a 3×8 grid with one escort. (a) and (h) are the start and goal configurations. (b)→(c): A cwr-shift inserts B tile 8 to the circular highway. (c)→(d)→ . . . →(e): A series of r-shifts orders B tiles in the workspace. (e)→(g): Additional r-shifts move B tiles to goals.

To execute the first stage, if a B tile in the workspace has a W tile above it, then execute a cwr-shift to insert the leftmost such B tile into the highway to not affect tiles to the right (Fig. 7(b)-(c)). Otherwise, apply *adjustment* cwr-shifts to the circular highway until a B tile in the workspace has a W tile above it. Because there are $m-2$ B tiles, at most $m-2$ adjustments are needed to move a W tile over each B tile, and so the total number of steps for this stage is at most $4[(m-2)+(m-2)] = 8m-16$.

The second stage uses the same operation to insert the B tiles into the workspace. The difference is that B tiles are now being inserted in the exact spot in the workspace corresponding to the desired permutation. Through the process, a tile in B never makes a full lap around the circular highway. Therefore, at most $2m+1$ adjustments are needed, with at most $m-2$ B tile insertions, taking at most $4[(2m+1)+(m-2)] = 12m-4$ steps.

In the third stage, apply r-shifts to move B tiles to their goals as shown in Fig. 7(e)-(g), taking $4[m - \lfloor \frac{m-2}{3} \rfloor]$ steps.

Now, approximately the right third of the grid has been solved in $4[6m - 5 - \lfloor \frac{m-2}{3} \rfloor]$ steps; we recurse in the same manner for $m \geq 5$ and solve the base case of $m = 4$ in 53 times steps (Korf 2008) by treating the problem as a normal (n^2-1)-puzzle instance. Through careful counting, we can conclude that $120m$ steps are always sufficient. □

The $120m$ makespan can be significantly reduced with more careful analysis, which we omit due to limited space. The important takeaway is Lemma 5.2 shows GSTP on $3 \times m$ grids can be solved in $O(m)$ steps, sufficient for establishing the upper bounds in our claimed contribution. In what follows, we describe related results needed to get the constant factors stated in Table 1 omitting the proofs.

If we have two escorts, we can cycle them on opposite

corners of their respective r-shifts to allow two cwr-shifts to happen simultaneously, leading to the following.

Corollary 5.3. GSTP *instances with two escorts on a* $3 \times m$ *grid can be solved in* $60m$ *steps.*

With significant additional efforts but following a similar line of reasoning, we can establish on $2 \times m$ grids that

Lemma 5.4. *Feasible* GSTP *instances with a single escort on a* $2 \times m$ *grid can be solved in* $58m$ *steps.*

While simulating two row or column permutations at once can be useful in solving GSTP faster, the limited amount of space may prevent us from doing so. Instead, simulating the permutation of one row or column will be much more useful.

Corollary 5.5. *Given a single escort, a* $2 \times m$ *grid can be permuted to fill one of its rows arbitrarily in* $27m$ *steps.*

With two escorts, we get significantly faster algorithms.

Lemma 5.6. GSTP *instances with two escorts on a* $2 \times m$ *grid can be solved in* $10m - 13$ *steps.*

Corollary 5.7. *Given two escorts on the left of the top row of a* $2 \times m$ *grid, the bottom row can be arbitrarily permuted in* $6m-1$ *time steps, maintaining the position of the escorts.*

Corollaries 5.5 and 5.7 will be instrumental in parallelizing row and column permutations necessitated by the RTA Shuffles without wasting additional steps in permuting the other row.

5.4 Tighter Makespan Upper Bounds for GSTP

We are now ready to tackle solving full GSTP instances. For GSTP, we will only examine the case in which grid dimensions are at least 2; the problem is otherwise trivial.

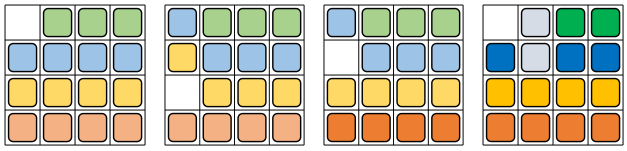
Theorem 5.8. *Feasible single-escort* GSTP *instances can be solved in* $81m_1m_2 + 6m_1 + 9m_2 - 3$ *steps.*

Proof. First, move the escort to the top left for start/goal configurations to get new start/goal configurations. Then, RTA is applied in a row-column-row fashion to yield three batches of row/column shuffles. Each batch requires sorting m_1 or m_2 rows or columns. In the 4×4 grid shown in Fig. 8(a), a batch of row shuffles must permute each of the four rows highlighted in different colors. We are done if we can successfully perform each batch of shuffles.

To execute a batch of shuffles, e.g., performing the four row shuffles on the 4×4 grid shown in Fig. 8(a), we move the escort to the top left of the bottom two rows and apply Corollary 5.5 sort the last row. The procedure is repeated with the escort moved one row above, until there are only two top rows, at which point Lemma 5.4 is invoked to arrange the two rows simultaneously. The top two rows may not be solved exactly because not all $(N^2 - 1)$-puzzles are solvable, but the issue will resolve on its own if the GSTP instance is solvable. Other shuffles are executed similarly.

Counting all steps, the total number is at most $81m_1m_2 + 6m_1 + 9m_2 - 3$. □

Theorem 5.9. *A two-escort* GSTP *instance can be solved in* $18m_1m_2 - 4m_1 - 5m_2 - 29$ *steps.*



(a) (b) (c) (d)

Figure 8: Illustrating performing a batch of row shuffles on a 4×4 grid with a single escort. (a). The (updated) start configuration, in which each row must be permuted. (b). To prepare for running Corollary 5.5, the escort is moved to the top left of the last two rows. (c). After applying Corollary 5.5 to sort the last row, the escort is shifted above for the next application. (d) The top two rows will be sorted using Lemma 5.4. Note that the top (resp., left) two rows (resp., columns) may not be fully solvable in the first two batches of shuffles, which is fine for the next set of column shuffles.

Proof Sketch. The proof is similar to the single escort case; with two escorts, we invoke Lemma 5.6 and Corollary 5.7 to speed up the process. The entire instance can be solved in $2[(m_1-2)(6m_2-1)+10m_2-13]+[(m_2-2)(6m_1-1)+10m_1-13]+4 = 18m_1m_2 - 4m_1 - 5m_2 - 29$ steps. □

Theorem 5.10. *A* GSTP *instance containing* $2 \leq k < \min(m_1, m_2)$ *escorts, where* k *is even, can be solved with a makespan less than* $\frac{44m_1m_2}{k} + m_1(5 - \frac{24}{k}) + 15m_2 - 29$.

Proof Sketch. The main strategy is distributing the escorts across the rows/columns to introduce parallelism in solving a batch of row/column shuffles. For example, given $k = 2\ell$ escorts, to solve a batch of m_1 row shuffles, we can distribute two escorts per $\frac{m_1}{\ell}$ rows. For each such $\frac{m_1}{\ell}$ rows, we invoke Lemma 5.6 and Corollary 5.7 to solve them, in parallel. This allows the entire batch of row shuffles to be completed in $O(\frac{m_1}{\ell})O(m_2) = O(\frac{m_1m_1}{\ell}) = O(\frac{m_1m_1}{k})$ steps. Tallying over the three phases, the total number of steps required is bounded by $\frac{44m_1m_2}{k} + m_1(5 - \frac{24}{k}) + 15m_2 - 29$. □

Note that if k is odd, we can simply ignore one escort. It is also clear the results continue to apply for $k \min(m_1, m_2)$ by ignoring extra escorts, but we can get additional speedups when $k \geq m_1 + m_2 - 1$ due to enough room to use 5.6 straightforwardly by having escorts along the top row and left column. Compiling everything so far yields the claimed bounds given in Table. 1.

6 Conclusion and Discussion

We show that it is NP-complete to compute makespan-optimal solutions for the generalized sliding-tile puzzle (GSTP). We further establish matching asymptotic makespan lower and upper bounds for GSTP for all possible numbers of escorts, and provide concrete constants for all makespan upper bounds. In ongoing and future work, we are examining (1) computing optimal solutions for other objectives for GSTP, (2) related variations of the GSTP formulation, and (3) developing practical algorithms for computing different optimal solutions for large-scale GSTP instances.

Acknowledgement

We thank the reviewers and editorial staff for their insightful suggestions. This work is supported in part by the DIMACS REU program NSF CNS-2150186, NSF award CCF-1934924, NSF award IIS-1845888, and an Amazon Research Award.

References

Atzmon, D.; Stern, R.; Felner, A.; Wagner, G.; Barták, R.; and Zhou, N.-F. 2018. Robust multi-agent path finding. In *Proceedings of the International Symposium on Combinatorial Search*, volume 9(1), 2–9.

Auletta, V.; Monti, A.; Parente, M.; and Persiano, P. 1999. A linear-time algorithm for the feasibility of pebble motion on trees. *Algorithmica*, 23(3): 223–245.

Barer, M.; Sharon, G.; Stern, R.; and Felner, A. 2014. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Proceedings of the International Symposium on Combinatorial Search*, volume 5(1), 19–27.

Culberson, J.; and Schaeffer, J. 1994. Efficiently searching the 15-puzzle. Technical report, University of Alberta. Technical report TR94-08.

Culberson, J. C.; and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence*, 14(3): 318–334.

Demaine, E. D.; Fekete, S. P.; Keldenich, P.; Meijer, H.; and Scheffer, C. 2019. Coordinated Motion Planning: Reconfiguring a Swarm of Labeled Robots with Bounded Stretch. *SIAM Journal on Computing*, 48(6): 1727–1762.

Demaine, E. D.; and Rudoy, M. 2018. A simple proof that the (n^2 - 1)-puzzle is hard. *Theoretical Computer Science*, 732: 80–84.

Erdem, E.; Kisa, D.; Oztok, U.; and Schüller, P. 2013. A general formal framework for pathfinding problems with multiple agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 27(1), 290–296.

Erdmann, M.; and Lozano-Perez, T. 1987. On multiple moving objects. *Algorithmica*, 2: 477–521.

Fekete, S. P.; Keldenich, P.; Krupke, D.; and Mitchell, J. S. 2022. Computing coordinated motion plans for robot swarms: The cg: shop challenge 2021. *ACM Journal of Experimental Algorithmics (JEA)*, 27: 1–12.

Geft, T.; and Halperin, D. 2022. Refined Hardness of Distance-Optimal Multi-Agent Path Finding. arXiv:2203.07416.

Goldreich, O. 2011. *Finding the shortest move-sequence in the graph-generalized 15-puzzle is NP-hard*. Springer.

Guo, T.; and Yu, J. 2022. Sub-1.5 Time-Optimal Multi-Robot Path Planning on Grids in Polynomial Time. arXiv:2201.08976.

Guo, T.; and Yu, J. 2023. Toward Efficient Physical and Algorithmic Design of Automated Garages. arXiv:2302.01305.

Hopcroft, J. E.; Schwartz, J. T.; and 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.

Korf, R. E. 2008. Linear-time disk-based implicit graph search. *Journal of the ACM (JACM)*, 55(6): 1–40.

Kornhauser, D.; Miller, G.; and Spirakis, P. 1984. Coordinating Pebble Motion On Graphs, The Diameter Of Permutation Groups, And Applications. In *25th Annual Symposium onFoundations of Computer Science, 1984.*, 241–250. IEEE.

Li, J.; Ruml, W.; and Koenig, S. 2021. Eecbs: A bounded-suboptimal search for multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35(14), 12353–12362.

Loyd, S. 1959. *Mathematical puzzles*, volume 1. Courier Corporation.

Mason, R. 2019. Developing a profitable online grocery logistics business: Exploring innovations in ordering, fulfilment, and distribution at ocado. *Contemporary Operations and Logistics: Achieving Excellence in Turbulent Times*, 365–383.

Okumura, K. 2023. Lacam: Search-based algorithm for quick multi-agent pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37(10), 11655–11662.

Okumura, K.; Machida, M.; Défago, X.; and Tamura, Y. 2022. Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence*, 310: 103752.

Pottinger, D. 1999. Implementing coordinated movement. *Game Developer Magazine*, 48–58.

Ratner, D.; and Warmuth, M. 1990. The (n^2 - 1)-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10(2): 111–137.

Santaló, L. A. 2004. *Integral geometry and geometric probability*. Cambridge university press.

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

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2013. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial intelligence*, 195: 470–495.

Silver, D. 2005. Cooperative pathfinding. In *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment*, volume 1(1), 117–122.

Solovey, K.; and Halperin, D. 2015. On the hardness of unlabeled multi-robot motion planning. arXiv:1408.2260.

Standley, T. 2010. Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24(1), 173–178.

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the International Symposium on Combinatorial Search*, volume 10(1), 151–158.

Surynek, P. 2010. An optimization variant of multi-robot path planning is intractable. In *Proceedings of the AAAI conference on artificial intelligence*, volume 24(1), 1261–1263.

Surynek, P. 2012. Towards optimal cooperative path planning in hard setups through satisfiability solving. In *Pacific Rim international conference on artificial intelligence*, 564–576. Springer.

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

Wilson, R. M. 1974. Graph puzzles, homotopy, and the alternating group. *Journal of Combinatorial Theory, Series B*, 16(1): 86–96.

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

Yu, J. 2013. A linear time algorithm for the feasibility of pebble motion on graphs. *arXiv preprint arXiv:1301.2342*.

Yu, J.; and LaValle, S. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 27(1), 1443–1449.

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