Connected Reconfiguration of Polyominoes Amid Obstacles using RRT*

Javier Garcia[1], Michael Yannuzzi[1], Peter Kramer[2], Christian Rieck[2], and Aaron T. Becker[1,2]

***Abstract*— This paper investigates using a sampling-based approach, the RRT*, to reconfigure a 2D set of connected tiles in complex environments, where multiple obstacles might be present. Since the target application is automated building of discrete, cellular structures using mobile robots, there are constraints that determine what tiles can be picked up and where they can be dropped off during reconfiguration. We compare our approach to two algorithms as global and local planners, and show that we are able to find more efficient build sequences using a reasonable amount of samples, in environments with varying degrees of obstacle space.**

I. Introduction

Cellular structures are related to reconfigurable robotics work, but rather than using intelligent, powered and actuated reconfigurable modules, small robots that walk along the modules are used to move them. This allows the modules to be passive, which reduces their complexity, weight, and cost. Automated building of discrete, cellular structures has potential applications at many scales, ranging from plans for kilometer-scale manufacturing structures in space [10], to millimeter-scale smart material [18], to nano-scale assembly with DNA [17].

Building using discrete, cellular structures provides some advantages when compared to methods that require an external scaffold or an external gantry such as traditional 3D-printing. The workspace of the gantry defines the size of the structure that can be built. In contrast, the cellular structures provide their own scaffold for construction, and modules can move along this structure to increase the build area.

The robot in Figs. 1 and 2 shows the motivating hardware system. Automated building of discrete cellular structures was explored by Jenett et al. [9]. Their work featured cellular components called *tiles* used as building material and BILL-E, a robot designed to reconfigure them. Tiles are discrete structures that can be assembled and disassembled by mating the magnets on two separate tiles faces. BILL-E is a mobile robotic platform based on the inchworm archetype. This six DoF robot can walk along the structure made of tiles, and can pick up, carry, and then place one tile at a time. As with reconfigurable robotics, the reconfiguration speed can often be increased by using more robots at a time [3], [5], but this paper focuses on moving a single robot.

This work was supported by the Alexander von Humboldt Foundation and the National Science Foundation under [IIS-1553063, 1849303, 2130793].

[1]Department of Electrical and Computer Engineering, University of Houston, Houston, TX USA
{jgarciagonzalez,mcyannuzzi,atbecker}@uh.edu

[2]Department of Computer Science, TU Braunschweig, Braunschweig, Germany {kramer,rieck}@ibr.cs.tu-bs.de

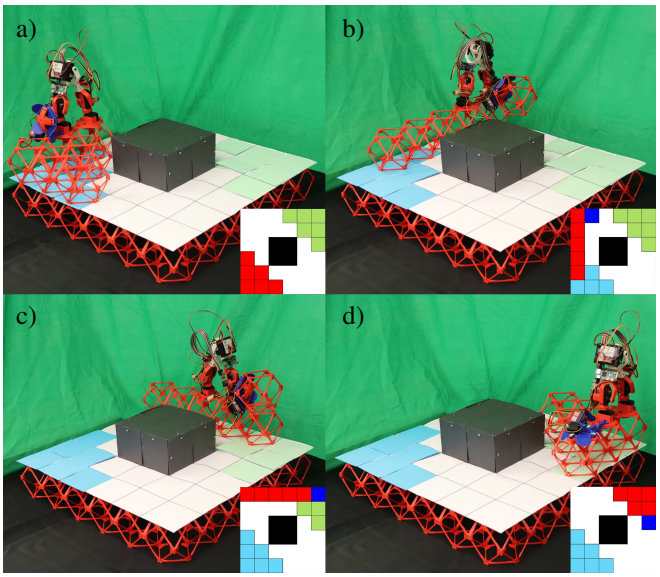


Fig. 1. This work is motivated by the challenge of reconfiguring a set of tiles using a simple robot [9] that can move one tile at a time while walking on the remaining tiles. The sequence a) to d) shows an example of a robot reconfiguring (white) tiles. The inset images are the planner's view, where red squares are tiles, light blue and light green squares are the start and goal configurations, the blue square is the location where the robot will place the next tile, and the black squares are obstacles. See overview video at https://youtu.be/Fp0MUag8po4.

We consider the problem of reconfiguring a given start into a given goal configuration of tiles using a single robot, moving one tile at a time while keeping all intermediate configurations connected, regardless of the presence of obstacles. Among other factors, power consumption motivates solving for the shortest set of moves to achieve this. Because the robot itself can only move on top of tiles (see Fig. 2), the connectivity constraint is crucial to ensure that the robot can reach every tile of the shape at all times. Furthermore, we require the tiles to be connected to make sure that the relative positions of the tiles stay the same during the reconfiguration, which is important when reconfiguring in space or water, where they can easily float away once disconnected.

II. Related Work

A. Tile reconfiguration

Reconfiguring a cellular structure is a challenging motion-planning problem, even when the problem is simplified to the placement of tiles in 2D. Tile reconfiguration has been explored by many authors. Gmyr et al. [8] explore algorithms for reconfiguring sets of hexagonal tiles, with applications in construction of nano materials.

Similar 2D reconfiguration work examined efficient methods for *compacting* tile structures. Dumitrescu and Pach [4] introduced a method where one tile is moved at a time by sliding along the perimeter of the polyomino. Their algorithm could convert an n-tile start configuration to an n-tile goal configuration in $\mathcal{O}(n^2)$ moves if the start and goal configurations have non-zero overlap. Moreno and Sacristán [15] modified the method of [4] to be in-place, i.e., the tile is always placed within one offset of the current structure. Their method turns any configuration into a solid rectangle within the bounding box of the start configuration. Akitaya et al. [1] proved that it is NP-hard to minimize the number of sliding moves under this model. They introduced a technique to improve reconfiguration performance by splitting the reconfiguration into a gathering stage and a compacting stage. In the gathering stage the structure is retracted such that each component is well connected. In the compacting stage the polyomino is transformed into a single, solid, xy-monotone component. These works are related, but require the start and goal configurations to overlap, occur in obstacle-free environments, and allow any tile to move – while our model requires the robot to travel to the next tile to move and carry it to its destination, and is capable of handling obstacles and start and goal configurations that do not overlap.

B. Sampling-based methods

While [1], [4], [15] introduced algorithms for reconfiguration, an alternative is to search for a solution using sampling.

Rapidly-expanding random trees (RRT) are a sampling-based motion-planning method designed to efficiently explore paths in high-dimensional spaces. RRTs were developed by LaValle and Kuffner [14], and are often used for planning problems with obstacles and other constraints.

This approach is challenging because of the large configuration space. The configurations of polyominoes form a high-dimensional space that is related to placing n tiles in a λ-tile free space – every possible n-omino can potentially also be translated and rotated. The number of viable configurations becomes smaller if obstacles constrain the build area. At one extreme, if the build area is a λ-tile long, 1-wide column, there is only one possible n-omino with $\lambda - n + 1$ possible translations. If the free space is instead a $\sqrt{\lambda} \times \sqrt{\lambda}$ -sized square, it is difficult to compute the number of valid configurations. Ignoring the constraint that the configuration must be connected, there are $\frac{\lambda!}{n!(\lambda-n)!}$ placements of n tiles in a λ-tile free space. Alternately, we could count the number of free polyominoes and ignore both obstacles and the position of the polyomino. However, even the best methods for computing free polyominoes [11] require time and memory that grows exponentially in n. To address the challenges of this large configuration space, this paper relies on local planners and a simplified distance heuristic.

C. Automated building of discrete cellular structures

BILL-E can traverse the structure by locking its feet on tile faces, and it can modify the structure by picking up and placing tiles with a gripper located at the front of the robot.

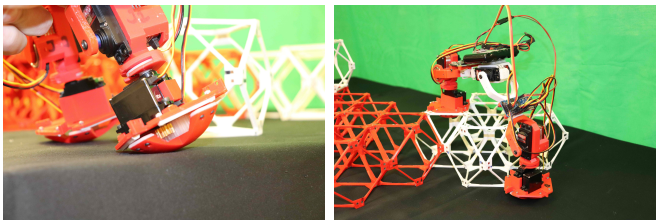


Fig. 2. The BILL-E bots are designed to walk on tiles only. The round design of the feet (left) does not allow the robot to step on other media (right). Because of this limitation, the structure must stay connected to ensure the BILL-E bot can reach every part of it. Additionally, disconnected structures could easily drift away in certain media like water or space.

This platform is easy to manufacture and assemble, making it a good candidate for implementing and testing automated building algorithms.

Previous work has explored methods to simplify complicated tile construction. Niehs et al. [16] and Fekete et al. [6] showed that the robot control could be represented as a finite automata and still enable building bounding boxes out of tiles around arbitrary polyominoes, scale and rotate them while keeping it connected at all times. These approaches are presented in a video by Abdel-Rahman et al. [2].

III. Definitions

The *workspace* is a rectangular unit grid, where each cell is either free, filled by a tile, or filled by an obstacle. This paper searches for reconfiguration sequences to convert a set of tiles from a start to a goal configuration. The start and goal configurations are each *connected* components, i.e., for every tile there is another tile that is adjacent to one of its sides. Such shapes are called *polyominoes*. As neither the robot nor the tile carried by the robot can cross an obstacle, we assume that both configurations are located within the same connected component of free space. Otherwise, no feasible reconfiguration sequence exists.

A configuration S is converted to another one by walking the robot to an adjacent position of a tile t, picking up t, and walking along a shortest edge-connected path on $S \setminus \{t\}$ before placing the tile in another location. An ordered series of these operations is called a *reconfiguration sequence*.

We refer to the distance walked before picking up a tile as the *pickup distance* d_P and the distance walked while carrying a tile as the *dropoff distance* d_D, respectively. These distances on the polyomino are determined by a *breadth-first search* tree (BFS) over the configuration.

In general, the distance between two workspace positions is defined by the length of the geodesic edge-connected path between them, taking into account the obstacles.

The *carry time* (*empty travel time*) of a reconfiguration sequence refers to the sum of all dropoff (pickup) distances. Similarly, the *total travel time* of a sequence then refers to the sum of carry time and empty travel time.

A *minimum-weight perfect matching* (MWPM) is a matching between tiles from the start and the goal configurations of minimum sum of dropoff distances.

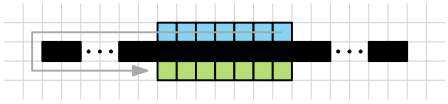


Fig. 3. Obstacles (shown in black) can force arbitrarily long "detours" using the MWPM between start (blue) and goal (green) configuration.

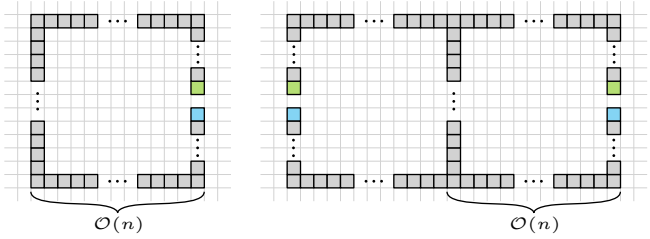


Fig. 4. Lower bound examples for carry time (left) and empty travel time (right). Note that gray tiles are contained in both start and goal configuration.

IV. Theoretical Background

Before discussing some practical methods and their results, we will briefly describe the potential impact of obstacles on the length of reconfiguration sequences, as well as introduce theoretical lower bounds on the pickup and dropoff distances that may have to be traversed.

Obstacles matter: It is easy to show that we can employ obstacles to create instances which require an arbitrarily higher number of moves than their obstacle-free counterparts. For example, consider two configurations, consisting of parallel lines that are two units apart from each another. By placing obstacles in a straight line between the two, we can increase the cost of a BFS-based MWPM by a factor that is linear in the number of obstacles, see Fig. 3. This corresponds to an increase in both the carry time and the empty travel time.

Lower bound on carry time: In the absence of obstacles, a significantly larger number of moves than the cost of an obstacle-free MWPM may still be necessary. We can bound this based on the carry time of applicable reconfiguration sequences. Consider a square-like "c-shaped" start configuration of n tiles and a goal configuration which requires moving one tile from one terminal of the "c" to the other, see Fig. 4 (left). Assuming a constant-size (i.e., $\mathcal{O}(1)$) gap between the two terminals, the MWPM of this instance has constant cost as well. Applicable strategies to reconfigure start into goal require either building a shortcut between the terminals and moving the tile from one side to the other, or picking it up and walking along the entirety of the configuration. Since both the arms of the "c" as well as its left edge are of length $\mathcal{O}(n)$, a shortcut would have to cover the same distance, implying an $\mathcal{O}(n)$ lower bound at least on the sum of dropoff distances. Similarly, the carry time spent walking a tile along the entire "c" implies an $\mathcal{O}(n)$ lower bound on the dropoff distance as well. We conclude that the carry time in applicable solutions is larger than the MWPM by a factor of $\mathcal{O}(n)$ for these configurations.

Lower bound on empty travel time: In a similar fashion, we can bound the sum of pickup distances from below. By mirroring the "c" along its left edge to form "$\mathfrak{w}$", we define a pair of configurations which still have a MWPM of constant cost, see Fig. 4 (right). While moving tiles from one terminal to the other can be achieved without empty travel, moving tiles in between both terminal pairs requires being present at all four terminals at least once. This implies that we traveled from one terminal pair to the other at some point, i.e., we traversed a pickup distance of $\mathcal{O}(n)$.

We conclude that there exist configurations where both minimal dropoff and pickup distances may be in $\mathcal{O}(n)$, even if the MWPM is of constant cost.

V. Methods

In this paper, we tackle the problem of determining a reconfiguration sequence for converting a start configuration into a goal configuration using a rapidly-expanding random tree-star (RRT*) [12]. The RRT* proceeds by building a tree where each node is a reachable configuration of the tiles. The root is the start configuration. We expand the tree by generating a random configuration (by constructing a random polyomino in the workspace), and searching for the node of the tree that is nearest to the current configuration. We then take this nearest node and attempt to reconfigure it toward the random configuration by applying at most rad dropoffs as determined by a local planner. The resulting configuration is then added to the RRT* tree. The RRT* then rewires the tree to form shortest paths.

A. Local planner algorithms

To reconfigure one configuration into another, we implemented two local planners (code at [7]). The local planner takes a start configuration S and a goal configuration G and returns the pickup location P and the dropoff location D for one tile. A *complete motion planner* either produces a solution in finite time or correctly reports that there is none.

We call any tile that can be picked up without disconnecting the remaining tiles a *leaf* tile. The shortest path along a polyomino from P to D is constructed by computing a BFS along the set of tiles $S \cup D$. The shortest path between coordinates C_i and C_j is constructed by computing a BFS along the obstacle free set $\neg O$.

Our first local planner is called GLC(S, G, O) for *Grow Largest Component*. It behaves differently depending on if the start and goal polyominoes overlap. If the start and goal polyominoes do not overlap, the closest tiles between S and G are found. Then the closest leaf tile in S to this gap is moved to shrink the gap by one. If there is an overlap, the largest connected component in the overlap is computed. All the tiles in G that connect to this connected component are identified as the set N. All the leaf nodes in S that are not already in the connected component are identified as the set L. Then the closest pair in $\{L, N\}$ is moved. A more detailed description of this algorithm can be found in the full version [7]. While this planner is complete (see Theorem 1 below), its solution is not guaranteed to be optimum, but it is fast to compute and serves as an upper-bound on the optimal solution.

Theorem 1: GLC(S, G, O) is a complete motion planner.

Proof: In case that S and G do not overlap, let S_e and G_e be the endpoints of a shortest path between S and G. If no such path exists, the goal configuration G is not reachable from S. Since S always contains at least two leaf tiles, a leaf tile can always be picked up from the configuration and placed on the first empty position of the path towards G, reducing the distance between S_e and G_e by one. Once this distance is one, i.e., they are adjacent, the result of such a move is a non-empty overlap $S \cap G$ which contains precisely G_e for the subsequent iteration.

Once the overlap is non-empty, the following holds true. So long as there exists at least one tile $t \notin M$, i.e., a tile that is not part of the largest connected component, there exists a spanning tree of the dual graph of S which has a leaf $t' \in L \setminus M$. This directly implies that if t itself is not a leaf, t is part of a path to a leaf tile t' outside of M. We conclude that at any given point, it is possible to determine a leaf tile that can safely be moved to become part of the largest connected component M of the overlap. ■

The reconfiguration sequence determined by GLC takes total travel time $\mathcal{O}(n^2)$ for instances with start distance no more than n between S and G. This stems from the fact that every tile is moved at most once as soon as the current and goal configurations overlap, since every tile that has been used to grow the largest component remains in that position until G is reached. Both the pickup and dropoff distances for each tile are bounded from above by n, resulting in $\mathcal{O}(n^2)$ total travel time. Some instances actually require at least $\Omega(n^2)$ travel time. Consider moving a row of n tiles to the right by n tiles.

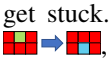
Our second planner, MWPMEXPAND(S, G, O), uses a minimum-weight perfect matching between all the tiles in S and G, where distances are calculated according to the shortest path around obstacles using BFS. The matching is sorted by distance between the pairs, and of all the leaf nodes in S, the one with the longest distance matching is moved as close as possible (along the configuration S) to its goal destination. While this planner uses a minimum-weight perfect matching, this is not a complete planner, and can get stuck. For instance, let $S \rightarrow G$ be the reconfiguration  , which seeks to move the middle tile upwards. MWPMEXPAND will only move the middle tile, but this cannot be moved without disconnecting the polyomino. GLC(S, G, O) can handle such a situation since it selects one component and grows it.

Fig. 5 demonstrates an example of polyomino reconfiguration using the GLC and MWPMEXPAND algorithms.

B. Tree nodes

The configuration of a polyomino is described by a binary occupancy grid with the same size as the workspace. These configurations constitute the nodes of the RRT*, and they can only be connected to other nodes if their configurations differ by a valid dropoff (see Fig. 6). A valid dropoff is described by a tile that can be picked up, and free path on the polyomino to carry it to a location where it can be placed. The tile that was picked up to create a node is referred to as the source, the location where it was placed is the target.

The cost of moving between connected nodes A and B is equal to the sum of d_P, the distance from A's target to B's source, and d_D, the distance from B's source to B's target. The cost to move to a node is dependent on the parent's target, so rewiring nodes in a section of the tree can affect faraway nodes.

To increase exploration rate while keeping the size of the tree manageable, nodes can be added after $rad > 1$ dropoffs. In this case, each node in the RRT* must contain the sources and targets for the intermediate configurations. Nodes can still be connected in less dropoffs if rewiring occurs or the random configuration is reached.

C. Distance heuristic

The algorithm extends the closest node in the tree toward a random configuration using a local planner. To determine the relative distance between two configurations, we use a simple distance heuristic h that describes how close they are to each other. Overlap ov is a clear factor; two configurations are the same only if their tiles occupy the same locations. Unfortunately, ov provides no information if the configurations do not overlap. To correct this, we also consider the center of mass com of each configuration, and the Euclidean distance between them.

$$h = \frac{ov + 1}{\max\{\|com_A - com_B\|_2,\, 0.1\}} \tag{1}$$

Two configurations can have the same center of mass, so a lower threshold is imposed on the denominator of Eq. (1) to avoid large results that can dominate over other candidates. Additionally, the numerator is equal to $ov + 1$ so a heuristic value can be assigned to configurations with no overlap.

D. Dynamic bias

The RRT* alternates between growing toward random configurations and the goal itself. The probability of choosing the goal is usually set to a small number, e.g., 5-10%, so the tree spends more time exploring [14]. This helps avoid converging to local minima by finding more paths.

If the start and goal configurations are initially far apart according to our heuristic, either because of little overlap or a large distance between their centers of mass, it can be advantageous for the tree to explore using a small goal bias. As tree nodes approach the goal structure, increasing the bias can accelerate finding a path.

We implement a dynamic bias that changes as the tree gets closer to the goal. This leverages the advantages of a small bias at the beginning, prioritizing exploration, and a higher bias that speeds up the path creation to reduce time to first solution. This is similar to the concept of simulated annealing [13], which searches for an optimum value using a search radius that decays to zero as time increases.

We set the dynamic bias to the sum of a base value $bias_{\text{base}}$, and a value dependent on the current performance of the tree. For this particular application, the performance is the ratio

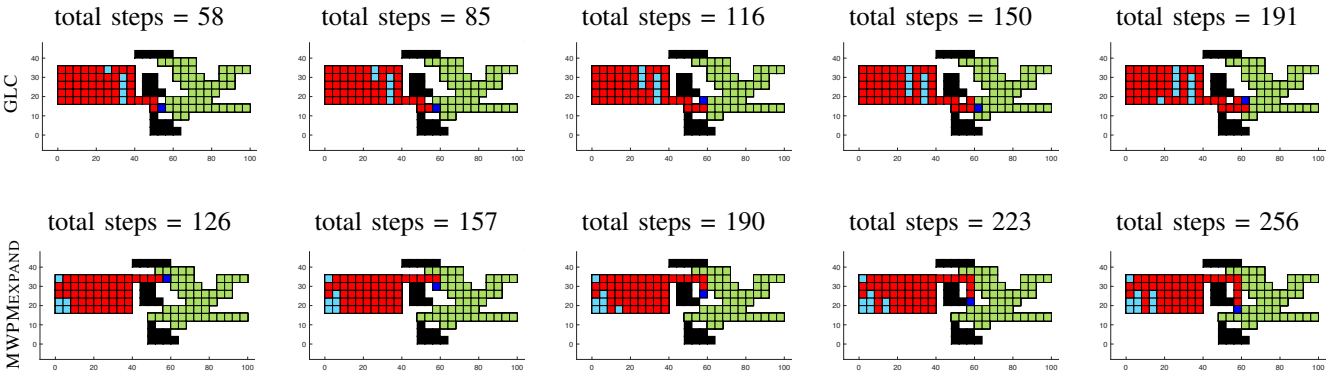


Fig. 5. Five consecutive dropoffs by the (top) GLC algorithm and (bottom) MWPMEXPAND local planners. GLC always places tiles on the goal structure when it can, while MWPMEXPAND can create bridges to reach further parts of the structure sooner.

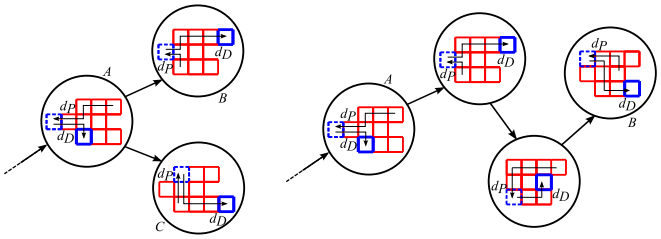


Fig. 6. An example of nodes in our RRT*. Each node represents a configuration. (Left) Node A can create nodes B and C with just one dropoff, so it connects to them. A dropoff is defined as moving to pick up a valid tile (d_P), and moving to drop it off at a valid location (d_D). However, B and C cannot create each other with just one dropoff so they are not connected. (Right) To increase exploration rate, nodes can be added after more than one dropoff. Here, B is three dropoffs away from A and is added as a node to the tree, while the intermediate configurations are not.

of μ_{ov}, the mean of the amount of overlap between the tree's nodes and the goal, and the total number of tiles n. An upper threshold $bias_{\max}$ is defined to determine the maximum bias toward the goal.

$$bias_{\text{dyn}} = bias_{\text{base}} + (bias_{\max} - bias_{\text{base}}) \frac{\mu_{ov}}{n} \qquad (2)$$

VI. RESULTS

A. Local planner performance

To evaluate the performance of the RRT* approach to polyomino reconfiguration, we created five maps with different characteristics, as shown in Fig. 7. Map 1 has the starting and goal configurations centered. In maps 2 and 3, the starting and goal configurations are adjacent but they encompass empty space. The last two maps introduce obstacles and require significant travel from the starting to the goal configuration.

The results for maps 3 and 4 shown in Fig. 7. The rest of the results, for the current and following subsection, can be found in [7]. GLC and MWPMEXPAND were used as local planners for the RRT*. For comparison, both were also used as global planners to find solutions.

For each map, five different values for $bias_{\max}$ were tested, as defined in Eq. (2). $bias_{\text{base}} = 0.1$ in all cases, so with $bias_{\max} = 0.1$ no dynamic bias was implemented. The tree continued expanding until ten thousand nodes were created. The GLC and MWPMEXPAND planners are shown with dashed and dotted lines, respectively. GLC is guaranteed to find a solution (see Theorem 1), so it appears in all of the plots. MWPMEXPAND can get stuck in local minima, and for maps 2 and 3 it did not find a solution.

An expected observation is that, for higher values of $bias_{\max}$, the tree finds a solution faster. The first point in all the plots is the average amount of nodes it needed to find a path, as well as the average cost of that initial solution. The nodes to first solution is similar for both the GLC and MWPMEXPAND as local planners, with the exception of map 3, for which the tree with MWPMEXPAND took considerably longer.

After ten thousand nodes the lower values for $bias_{\max}$ did not find better solutions, despite prioritizing exploration. For all but map 2 in the GLC results, $bias_{\max} = 0.75$ performs best in terms of time required to find a path and the cost of the solution after ten thousand nodes. In the MWPMEXPAND results, the same $bias_{\max}$ value consistently performs better than most of the other values.

Compared to the algorithms as global planners, at least one setting of RRT* outperforms the GLC in all but map 1. The MWPMEXPAND performed worse than most RRT* settings for maps 1 and 4. Map 5, the other map for which it found a solution, was the only map where it outperformed the rest of the strategies. In addition, RRT*(MWPMEXPAND) performed much worse on maps 1 and 5.

B. Initial solution and multiple dropoffs

To increase the probability of RRT* finding a better solution, the tree can be initialized with the best solution between GLC and MWPMEXPAND as global planners. Simulations similar to the previous section were carried out with this strategy. Since MWPMEXPAND tends to get stuck often, and based on the results from the previous section, GLC is always used as the local planner for RRT*.

Additionally, the effect of multiple dropoffs between nodes is investigated. Nodes are added to the tree every rad number of dropoffs, unless the configurations being expanded toward are reached. Rewiring can also result in fewer dropoffs.

The results for maps 3 and 4 are shown in Fig. 8. For both of these maps, GLC found the best/only solution as a global planner. For map 4 the RRT* was able to considerably improve the initial solution, whereas before it only managed

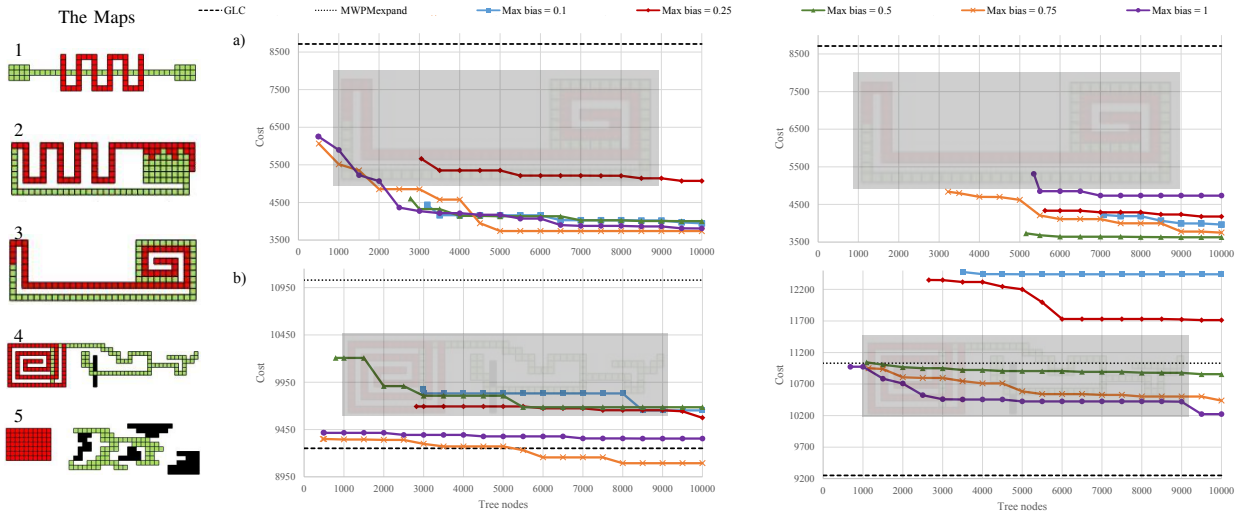


Fig. 7. Comparison of different strategies for a) map 3 and b) map 4. GLC is the local planner for the RRT* on the left plots, while MWPMEXPAND is the local planner on the right ones. RRT* was run ten times for each map, for five different $bias_{\max}$ values, for each local planner. Because GLC is a complete planner, it always finds a solution, while MWPMEXPAND gets stuck often. All maps (1-5) are shown on the left. See animation at `https://youtu.be/Fp0MUag8po4`.

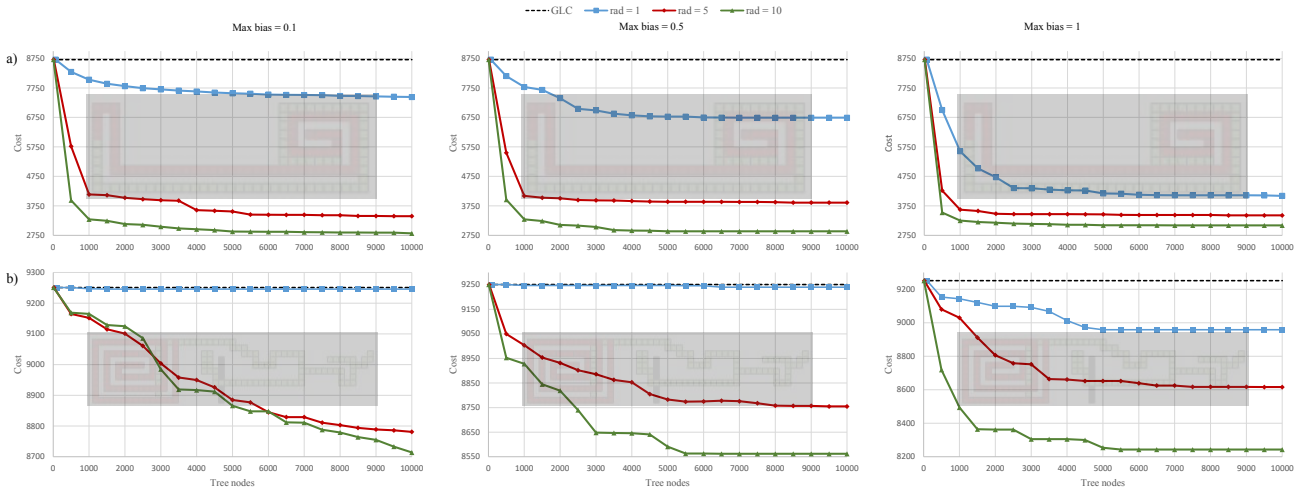


Fig. 8. RRT* results using the best initial solution for a) map 3 and b) map 4. Three different rad values were used to test the effect of multiple dropoffs, with five different $bias_{\max}$ values. Here we show the plots for (left) $bias_{\max} = 0.1$, (middle) $bias_{\max} = 0.5$ and (right) $bias_{\max} = 1$. Each point is the average of ten runs.

to do so once. The higher rad values were also able to outperform the previous solutions for map 3.

For the two maps shown, the higher rad values resulted in better performance. However, for maps 1, 2 and 5 the trend was the opposite. One downside of the higher values is that node creation takes longer, because every intermediate configuration is checked for potential rewiring.

An important thing to note is that sometimes the RRT* can worsen the initial solution, as seen in maps 1 and 5. The reason is that when a node is rewired the position of the robot changes, which can negatively affect the cost of subsequent nodes. To avoid increasing the time complexity of the algorithm, only the costs of immediate children nodes are considered. A deeper comparison, or duplicating nodes when they both lower and raise costs of child nodes, could be implemented to eliminate this issue.

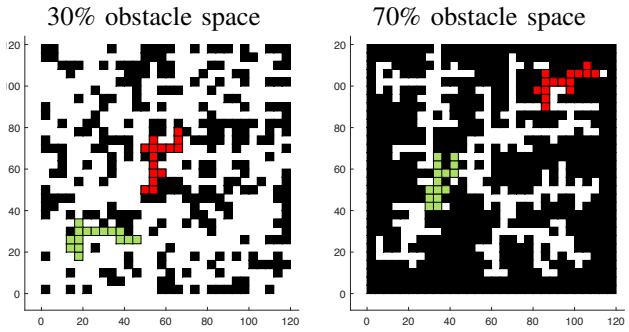


Fig. 9. Randomly generated maps for two percentages of obstacle space. The green polyomino is the start configuration, and the red one is the goal. In the maps used for testing, the workspace has a size of 30×30 tiles, and the polyominoes are composed of 15 tiles.

C. Percentage obstacle space

We continue to test how the RRT* performs as more obstacles are present in the workspace. For five different

TABLE I

COMPARISON OF PLANNING STRATEGIES

obstacle percent (%)	method returning best solution			MWPM EXPAND finds a solution
	RRT*(GLC)	RRT* (MWPM EXPAND)	GLC	
10%	50%	30%	20%	10%
30%	50%	40%	10%	10%
50%	60%	40%	0%	30%
70%	50%	50%	0%	10%
90%	40%	40%	20%	90%

TABLE II

COMPARISON OF RRT* WITH BEST INITIAL SOLUTION

obstacle percent (%)	best initial solution		improves initial solution		
	GLC	MWPM EXPAND	RRT* (GLC) $rad = 1$	RRT* (GLC) $rad = 5$	RRT* (GLC) $rad = 10$
10%	100%	0%	40%	60%	10%
30%	90%	10%	60%	50%	10%
50%	90%	10%	50%	50%	20%
70%	90%	10%	70%	60%	20%
90%	70%	30%	60%	10%	30%

percentages of obstacle space we created ten random maps (50 maps total). Examples of randomly generated maps for different percentages are shown in Fig. 9.

First, both GLC and MWPMEXPAND are used as local planners for the RRT*. Based on the results from Sec. VI-A, $bias_{\max}$ was set to 0.75, and rad to 1. The algorithm stops after ten thousand nodes are created or a time limit passes (required for higher percentage obstacle maps).

The performance of the RRT* is summarized in Table I. It returned the best solution for most maps, regardless of the density of obstacles, and RRT*(GLC) slightly outperformed RRT*(MWPMEXPAND) for the $10\%, 30\%$ and 50% obstacle maps. Additionally, RRT*(MWPMEXPAND) returns solutions with much higher costs than RRT*(GLC) in many cases, showing that GLC is more robust as a local planner. MWPMEXPAND never returned the least costly path, and in fact it had a low success rate for all but the 90% obstacles. At 90% obstacles there is little space for the tree to explore, and the possibility of additional paths to the goal is minimal so the four strategies returned paths with very similar costs.

Secondly, the RRT* was initialized with the best solution as in Sec. VI-B. GLC was always used as the local planner, and $bias_{\max}$ was kept at 0.75. The corresponding results are summarized in Table II. Once again, GLC returned the best initial solution the majority of the time. For $rad = 1$ and $rad = 5$ the RRT* was able to improve the initial for more than half the maps for most percentages. $rad = 10$ did not perform so well, suggesting that the increased exploration rate was not very helpful in these maps with higher density of randomly generated obstacles.

VII. CONCLUSIONS AND FUTURE WORK

This paper presented two local planners, GLC and MWPMEXPAND, and an RRT* implementation using these planners to optimize reconfiguration sequences for a set of tiles in complex environments. The planners and RRT* were tested on obstacle-free and obstacle-filled environments. The results show that the RRT* often finds sequences with lower costs, and that GLC is more robust as a local planner for the tree than MWPMEXPAND.

Future work should study the complexity class of the reconfiguration, additional techniques to enhance the RRT* such as multi-query trees (for a robotic swarm as an example) and consider more accurate distance heuristics for finding nearest neighbors. Additionally, the algorithms should be extended to work for 3D reconfiguration.

REFERENCES

[1] H. A. Akitaya, E. D. Demaine, M. Korman, I. Kostitsyna, I. Parada, W. Sonke, B. Speckmann, R. Uehara, and J. Wulms, "Compacting squares: Input-sensitive in-place reconfiguration of sliding squares," in *Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, 2022.

[2] A. Abdel-Rahman, A. T. Becker, D. E. Biediger, K. C. Cheung, S. P. Fekete, N. A. Gershenfeld, S. Hugo, B. Jenett, P. Keldenich, E. Niehs, C. Rieck, A. Schmidt, C. Scheffer, and M. Yannuzzi, "Space ants: Constructing and reconfiguring large-scale structures with finite automata," in *Symposium on Computational Geometry (SoCG)*, 2020, pp. 73:1–73:6.

[3] J. Bourgeois, S. P. Fekete, R. Kosfeld, P. Kramer, B. Piranda, C. Rieck, and C. Scheffer, "Space ants: Episode II - coordinating connected catoms," in *Symposium on Computational Geometry (SoCG)*, 2022.

[4] A. Dumitrescu and J. Pach, "Pushing squares around," *Graphs and Combinatorics*, vol. 22, no. 1, pp. 37–50, 2006.

[5] S. P. Fekete, P. Keldenich, R. Kosfeld, C. Rieck, and C. Scheffer, "Connected coordinated motion planning with bounded stretch," in *International Symposium on Algorithms and Computation (ISAAC)*, 2021.

[6] S. P. Fekete, E. Niehs, C. Scheffer, and A. Schmidt, "Connected reconfiguration of lattice-based cellular structures by finite-memory robots," *Algorithmica*, vol. 84, no. 10, pp. 2954–2986, 2022.

[7] J. Garcia, M. Yannuzzi, P. Kramer, C. Rieck, and A. T. Becker, "Connected reconfiguration of polyominoes amid obstacles using RRT*," 2022. [Online]. Available: https://arxiv.org/abs/2207.01282

[8] R. Gmyr, K. Hinnenthal, I. Kostitsyna, F. Kuhn, D. Rudolph, C. Scheideler, and T. Strothmann, "Forming tile shapes with simple robots," *Natural Computing*, vol. 19, no. 2, pp. 375–390, 2020.

[9] B. Jenett, A. Abdel-Rahman, K. Cheung, and N. Gershenfeld, "Material–robot system for assembly of discrete cellular structures," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4019–4026, 2019.

[10] B. Jenett, C. Gregg, D. Cellucci, and K. Cheung, "Design of multifunctional hierarchical space structures," in *IEEE Aerospace Conference*, 2017, pp. 1–10.

[11] I. Jensen, "Enumerations of lattice animals and trees," *Journal of Statistical Physics*, vol. 102, no. 3, pp. 865–881, 2001.

[12] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[13] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[14] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[15] J. Moreno and V. Sacristán, "Reconfiguring sliding squares in-place by flooding," in *European Workshop on Computational Geometry (EuroCG)*, 2020.

[16] E. Niehs, A. Schmidt, C. Scheffer, D. E. Biediger, M. Yannuzzi, B. Jenett, A. Abdel-Rahman, K. C. Cheung, A. T. Becker, and S. P. Fekete, "Recognition and reconfiguration of lattice-based cellular structures by simple robots," in *International Conference on Robotics and Automation (ICRA)*, 2020, pp. 8252–8259.

[17] J. Song, Z. Li, P. Wang, T. Meyer, C. Mao, and Y. Ke, "Reconfiguration of dna molecular arrays driven by information relay," *Science*, vol. 357, no. 6349, 2017.

[18] P. Thalamy, B. Piranda, and J. Bourgeois, "Engineering efficient and massively parallel 3d self-reconfiguration using sandboxing, scaffolding and coating," *Robotics and Autonomous Systems*, vol. 146, p. 103875, 2021.