Technische Universität Braunschweig

Bachelor Thesis

Computing Motion Plans for Assembling Particles with Global Control

Patrick Blumenberg

Institut für Betriebssysteme und Rechnerverbund

Supervised by
Prof. Dr. Aaron Becker
Dr. Arne Schmidt

January 12, 2022

Statement of Originality

This thesis has been performed independently with the support of my supervisor/s. To the best of the author's knowledge, this thesis contains no material previously published or written by another person except where due reference is made in the text.

Braunschweig, January 12, 2022

Aufgabenstellung / Task Description

Deutsch: Wenn es darum geht, Formen im Micro- und Nanobereich zusammenzubauen, ist es aufgrund der Größe der auftretenden Komponenten, begrenzter Rechenleistung und limitierter Energiekapazität sehr schwierig, einzelne Partikel an ihre jeweilig gewünschte Position zu bewegen. Eine Möglichkeit wurde von Erik Winfree 1998 vorgeschlagen, bei der alle (quadratischen) Partikel durch Diffusion aufeinandertreffen und unter verschiedenen Aspekten und Bedingungen aneinander haften bleiben. Eine andere Möglichkeit besteht darin, alle Partikel gleichzeitig durch eine äußere Kraft (zum Beispiel ein Magnetfeld) zu bewegen. In diesem „Tilt-Modell“bewegen sich die Partikel dabei solange in die gewünschte Richtung, bis sie auf ein anderes Partikel oder ein Hindernis treffen. Frühere Arbeiten trafen dabei allerdings oft Annahmen über die Startkonfiguration der Partikel oder verlangte speziell konstruierte Umgebungen. Zum Beispiel können die Partikel in Depots liegen, aus denen die Partikel extrahiert werden können.

Herr Blumenberg soll sich im Rahmen seiner Abschlussarbeit mit dem Konstruieren von Formen im Tilt-Modell beschäftigen, wobei keine Annahmen über die Startkonfiguration getroffen werden sollen. Dabei ist es interessant, wie gut Instanzen in der Praxis gelöst werden können und welche Inputparameter den größten Einfluss auf das Problem haben. Dazu gehört nicht nur das Lösen mit heuristischen Ansätzen, sondern auch das Bestimmen optimaler Bewegungspläne. Eine theoretische Analyse, wie beispielsweise Schranken für die Länge eines Bewegungsplans, ist wünschenswert.

English: When assembling structures at the micro or nanoscale, moving particles to desired target locations is often challenging due to the small size, limited computing power, or limited energy capacity. In 1998, Erik Winfree proposed a model in which all particles move by diffusion and connect if they are compatible. Alternatively, in the “tilt model,” particles are actuated by an external force, e.g., gravitational or magnetic forces. The external force moves all particles in the selected direction unless blocked by another particle or an obstacle. Related work considered the tilt model but often required either the particles to have specific starting conditions such as starting all particles in depots or required specialized workspace configurations designed to make manipulation easier.

In his thesis, Mr. Blumenberg’s task is to design algorithms that assemble structures using the tilt model on randomly generated workspaces with randomly placed starting configurations. One exciting outcome will be studying how good instances can be solved in practice and analyzing which parameters have the most significant effect on the difficulty of the motion planning problem. The task includes heuristics and computing optimal motion plans that minimize the number of steps. Theoretical analysis finding bounds for the length of a motion plan is desired.

Abstract

In this thesis, we investigate motion planning algorithms for the assembly of shapes in the Tilt model in which particles move under the influence of uniform external forces and self-assemble according to certain rules. The goal is to design algorithms that can efficiently compute short input sequences that transform an initial configuration of movable square-shaped tiles on a 2D grid with immovable obstacles into another configuration containing a target shape. Furthermore, the influence of various input parameters on the performance of motion planning algorithms is evaluated.

We discuss the computational complexity of the underlying problems and prove PSPACE-completeness when the assembly of shapes requires the presence of a fixed seed tile, even under the limitation that the number of available tiles is equal to the size of the target shape. Based on methods for motion planning from the field of robotics, we design various algorithms. Approaches based on an A* search with a consistent heuristic can compute a solution of minimal length, whereas other approaches, such as greedy best-first search and rapidly-exploring random trees, give up optimality in exchange for improved runtime efficiency. Additionally, a method that can shorten an existing input sequence is proposed. The performance of the proposed algorithms is experimentally evaluated on sets of instances created randomly based on six input parameters. We analyze the influence of the parameters on the performance of motion planning algorithms and demonstrate the strength and weaknesses of each approach.

For all evaluated algorithms, the runtime largely depends on the number of tiles. Algorithms that assemble the target shape one tile at a time displayed the overall best performance, even though they are not a complete solution. A problem variant that requires a fixed seed tile is easier to solve in practice and can be solved more efficiently by a specialized algorithm.

Contents

1 Introduction **1**
- 1.1 Related Work . . . 1
- 1.2 Contribution and Structure . . . 3

2 Preliminaries **5**
- 2.1 Problem Definitions . . . 6
- 2.2 Distance Definitions . . . 6

3 Algorithmic Approaches **7**
- 3.1 Hardness of the Problems . . . 7
- 3.2 Best-First Search . . . 8
  - 3.2.1 All Tiles at the Same Time . . . 9
  - 3.2.2 One Tile at a Time . . . 11
- 3.3 RRT . . . 15
- 3.4 Solution Shortening . . . 16

4 Experiments **17**
- 4.1 Method for Instance Creation . . . 17
- 4.2 Experimental Setup . . . 19
  - 4.2.1 Simulation Environment . . . 19
  - 4.2.2 Evaluated Approaches . . . 19
  - 4.2.3 Experimental Procedure . . . 20
- 4.3 Results . . . 22
  - 4.3.1 Evaluation of `InstanceSet1` . . . 22
  - 4.3.2 Evaluation of `InstanceSet2` . . . 46

5 Conclusion and Future Work **55**

List of Figures

2.1 Example instance of the Polyomino Assembly Problem 6

3.1 Reduction of the k-region relocation problem to Fixed Seed Tile Polyomino Assembly Problem 8
3.2 Example of necessary subassemblies 12
3.3 Example for MMP distance 13

4.1 Example instances for the two different board types 18
4.2 Runtime comparison by problem type for GAD 21
4.3 Runtime of the BFS motion planner on `InstanceSet1` 23
4.4 Fraction of `InstanceSet1` solved by the BFS motion planner 24
4.5 Solution lengths for BFS motion planner on `InstanceSet1` 25
4.6 Runtime of the search-based heuristic planners on `InstanceSet1` 27
4.7 Fraction of `InstanceSet1` solved by the search-based planners 28
4.8 Solution lengths for search-based heuristic planners on `InstanceSet1` . . 29
4.9 Number of steps after assembly of the final polyomino 30
4.10 Runtime of several planners on `InstanceSet1` 32
4.11 Fraction of `InstanceSet1` solved by several planners 33
4.12 Solution lengths for several planners on `InstanceSet1` 34
4.13 Runtime comparison by problem type for GAD 35
4.14 Runtime comparison by problem type for MMPT 36
4.15 Runtime comparison by problem type for RRT 36
4.16 Runtime and success rate of GAD on maze and cave boards 38
4.17 Runtime and success rate of MMPT on maze and cave boards 39
4.18 Runtime and success rate of RRT on maze and cave boards 40
4.19 Solution length by board type 41
4.20 Average pairwise distance on maze and cave boards 41
4.21 Runtime and success rate of GAD by number of glue types 42
4.22 Runtime and success rate of MMPT by number of glue types 43
4.23 Runtime and success rate of RRT by number of glue types 43
4.24 Memory usage of GAD 44
4.25 Memory usage of MMPT 45
4.26 Memory usage of RRT 45
4.27 Effectiveness of the solution shortening method 46
4.28 Runtime of several motion planners on `InstanceSet2` 48
4.29 Fraction of `InstanceSet2` solved by several planners 49
4.30 Solution lengths for several planners on `InstanceSet2` 50

4.31 Fraction of `InstanceSet2` with fixed tile solved by several planners 51
4.32 Runtime and success rate comparison of GAD and WSD 51
4.33 Runtime and success rate of planners by number of extra tiles 53
4.34 Solution length of several planners by number of extra tiles 54

1 Introduction

Motivated by the difficulty of controlling large robot swarms at the molecular scale, where the manipulation of individual robots becomes infeasible due to their small size and limited capabilities, Becker et al. [4] proposed the Tilt model in 2013. It provides a way to describe particles that move under the influence of uniform external forces, such as magnetic or gravitational fields. When combined with particles that have the ability to self-assemble into larger objects, this gives rise to several potential applications including medical applications, such as minimally invasive surgery or targeted drug delivery, as well as the construction of objects at tiny scales.

In the Tilt model, movable unit squares called tiles are placed on a 2-dimensional grid that may also contain immovable obstacles. All tiles receive the same global movement instructions which cause them to move a unit distance (or maximally) in one of the cardinal directions unless they are blocked by an obstacle. Additionally, tiles can have glues on their edges, which allow them to bond according to certain rules and assemble into polyominoes. Tiles forming a polyomino move as a unit and cannot be separated.

A difficult challenge in this context is the design of efficient motion planning algorithms that can compute a sequence of global movement instructions such that the resulting configuration contains the desired target shape. Ideally, such an algorithm should minimize the length of the computed sequences. In this thesis, we design several heuristic methods that attempt to solve this problem and evaluate them experimentally on randomly-generated boards and initial configurations. Furthermore, we analyze the impact that several characteristics of the environment have on the efficiency of motion planning algorithms.

1.1 Related Work

Motion planning is a fundamental research topic in the field of robotics, which has been studied extensively. See [19] for a textbook or [24] for a survey article. The general problem is to find a continuous path of one or multiple robots from an initial position to a target position that avoids collisions or to determine that no such path exists. Formally, this is known as a reconfiguration problem, in which a path from an initial configuration to a target configuration through a configuration space must be discovered. In this context, the configuration space is defined as the set of possible states (or configurations) of the robots. The dimensionality of the configuration space, which depends on the number of independently moving parts of the robot, often limits the efficiency of motion planning algorithms, as the number of vertices that must be explored grows exponentially in the number of dimensions. This problem is known as the

curse of dimensionality [20]. For example, in this thesis n movable tiles corresponds to a $2n$-dimensional configuration space. Furthermore, several motion planning problems are known to be computationally difficult. For example, the Warehouseman's Problem, in which multiple, independently moving, rectangular objects in 2-dimensional space must be moved to target locations, has been proved to be PSPACE-hard [15]. Nevertheless, several approaches to efficient motion planning have been developed.

For low-dimensional systems, in particular, *search-based* planning algorithms such as A* and other heuristic tree search methods can be effective [8]. These algorithms build a graph structure over the configuration space and then use graph-search algorithms to find a path.

Another common class of motion planning algorithms is the *potential field method*, in which an artificial potential field represents attractive and repulsive forces that guide the path of the robots. Although this method is simple and adaptable, local minima in the potential field pose a major challenge [25].

Lastly, *sampling-based* methods like probabilistic roadmaps (PRM) [16] and rapidly exploring random trees (RRT) [18] can be used to discover a path in high-dimensional configuration spaces (e.g. humanoid robots or multi-robot systems). Sampling-based methods sample configurations at random and rely on collision checking to connect the sampled points to a graph. RRT* and related solutions try to combine the efficient exploration of sampling-based methods with the optimality of the results by continuously improving a solution.

This thesis uses search-based algorithms as well as RRTs to solve motion planning problems in the context of Tilt.

The Tilt model of motion planning [4] presents a possible solution for scenarios in which a collection of particles or robots which cannot be moved individually needs to be reconfigured. The basic idea is to control tiles on a 2-dimensional grid with fixed obstacles through the use of uniform external forces in one of the cardinal directions at a time. In practice, gravity or magnetic fields can be used for this purpose. Depending on the specific model, every control input causes all tiles to move either maximally until they hit an obstacle (tilt transformation) or by one unit distance in the corresponding direction (step transformation).

Later work by Becker et al. [7] introduced the ability to assemble polyominoes out of tiles similar to traditional models for self-assembly, such as the *abstract Tile Assembly Model* (aTAM) introduced by Erik Winfree in 1998 [23]. In self-assembly models like aTAM, particles combine through a nondeterministic process, and the possible products are determined only by the rules according to which particles bond. Practical implementations of self-assembling systems can use molecular diffusion of DNA or other substances to carry out computations. aTAM uses Wang Tiles [22] as atomic building blocks to form polyominoes according to the glues on each of their edges and the affinities defined by a glue function. In the original model, a seed tile is required to form assemblies and tiles must be added one at a time. Over the following years, many generalizations of aTAM were proposed [12, 11, 13], some of which allow subassemblies to be combined in parallel or completely abandon the concept of seed tiles.

As the above models, we use Wang tiles as building blocks and let glues on each of the edges of a Wang tile decide how tiles bond. Furthermore, both a model including a fixed seed tile that is required to form assemblies, as well as a model in which tiles can combine freely, are considered in this thesis.

Over the last years, the computational complexity and algorithmic solutions of multiple problems that arise in the Tilt model have been investigated.

Shortly after the introduction of the model, in 2014, it was shown that finding a minimum-length sequence of tilts that reconfigures an initial configuration into a target configuration is PSPACE-complete [5].

In addition to the reconfiguration problem, two other naturally arising problems received attention, namely the *relocation* and *occupancy* problem. The first one asks whether a specific tile can reach a specific target position; the second one, whether a given position on the board can be occupied by any tile. In 2019 Balanza-Martinez et al. [3] showed that the relocation problem for single tiles under tilt transformations is PSPACE-complete if a single 2×2 polyomino is allowed.

Later, in [2] the authors showed that the occupancy problem is PSPACE-complete, even for 1×1 tiles without glues.

Several papers [7, 21, 3, 2] dealt with the constructability of shapes under tilt transformations, as well as universal constructors, that can build arbitrary shapes.

Directly relevant to our work are the hardness results under single-step transformation. In [1, 9], motion planning problems under step transformation with a limited number of allowed input directions were investigated. The relocation problem was found to be NP-complete when limited to 3 directions, whereas reconfiguration was shown to be NP-complete when limited to 2 directions. On the other hand, the occupancy problem with only 1×1 tiles is solvable in polynomial time regardless of the number of allowed directions.

For the case without limited directions, Caballero et al. [10] proved that occupancy is PSPACE-complete when 1×2 polyominoes are allowed; whereas relocation is PSPACE-complete even with only 1×1 tiles.

Recent work on the constructibility of polyominoes under step transformations found efficient algorithms for certain classes of polyominoes but leaves open the question about the computational complexity of the general problem [17].

In 2020, Becker et al. investigated algorithmic solutions for collecting particles in a 2-dimensional maze through the use of uniform external forces [6]. To the best of our knowledge, no other work on the design and experimental evaluation of Tilt-related motion planning algorithms has been published.

1.2 Contribution and Structure

Chapter 2 provides definitions and introduces two motion planning problems related to the assembly of polyominoes in the Tilt model under step transformations: the Polyomino Assembly Problem and the Fixed Seed Tile Polyomino Assembly Problem. Both are special cases of a reconfiguration problem that differ in their assumptions about how

tiles bond. The first one allows polyominoes to bond whenever the glues on their edges allow it, the second one requires the presence of an immovable seed tile to assemble polyominoes.

In Chapter 3, we discuss the hardness of the problems and prove PSPACE-hardness for the Fixed Seed Tile Polyomino Assembly Problem, even under the limitations that the number of tiles on the board is limited to the size of the target shape and the number of glues is constant. Subsequently, several algorithmic approaches based on heuristics that aim to solve these problems in practice are introduced. Both search-based methods, such as A* search and a sampling-based method using RRTs are explored. Furthermore, a method to shorten an existing step sequence is briefly discussed.

In Chapter 4, we evaluate the effectiveness of the proposed algorithmic approaches on randomly generated instances and investigate the effect of different parameters on the performance of each algorithm and the practical difficulty of the problems in general.

Chapter 5 concludes the thesis and gives possible directions for future research.

2 Preliminaries

Board A *board* B is a rectangular region of $\mathbb{Z}^2$ where each position is either marked as *open* or *blocked*. Blocked positions represent obstacles and cannot contain tiles. Furthermore, the induced grid graph G_B is defined as the graph in which the open positions of B are nodes and two nodes are connected if and only if their distance is 1. B is called *connected* if and only if G_B is connected.

Tiles and Glues A tile $t = (p, g)$ is a unit square centered on p with edge glues g, where p is an open position on the board and g is a quadruple $(g_N(t), g_E(t), g_S(t), g_W(t))$ of glues. $g_d(t)$ denotes the glue on the edge of t facing in the cardinal direction $d \in \{N, E, S, W\}$. Each glue is an element of a finite alphabet Σ, which also contains a special `null` glue. A glue function $G\colon \Sigma \to \{0, 1\}$ defines which glues stick to each other. Two glues $g_1, g_2 \in \Sigma$ stick to each other with respect to G if and only if $G(g_1, g_2) = 1$. Furthermore, G has the properties $G(g_1, g_2) = G(g_2, g_1)$ and $G(g_1, \texttt{null}) = 0$ for all $g_1, g_2 \in \Sigma$. Tiles *bond*, if and only if they are located on adjacent positions and the glues on the shared edge stick together. The bond is considered permanent, and the tiles hereafter move as a unit.

Polyomino A *polyomino* P is a finite set of tiles that forms a connected component in the graph in which bonded tiles are adjacent. A polyomino containing exactly one tile is called trivial. The position of P is defined as the position of the top-leftmost tile in P.

Configuration A *configuration* is a tuple $C = (B, T)$, consisting of a a board B and a set of tiles T. The positions of tiles must be non-overlapping and open in B.

Step A *step* is a transformation between two configurations that moves every tile by one unit distance in one of the cardinal directions unless that tile is part of a blocked polyomino. Formally, a polyomino is *blocked* in the direction d if and only if it is not contained in the maximal subset of polyominoes P such that moving all tiles in $\bigcup P$ by one unit distance in the direction d would neither cause tiles to overlap with each other nor with blocked positions. For simplicity, a step can be denoted by the direction $d \in \{N, E, S, W\}$ in which the polyominoes are moved. A *step sequence* is a series of steps.

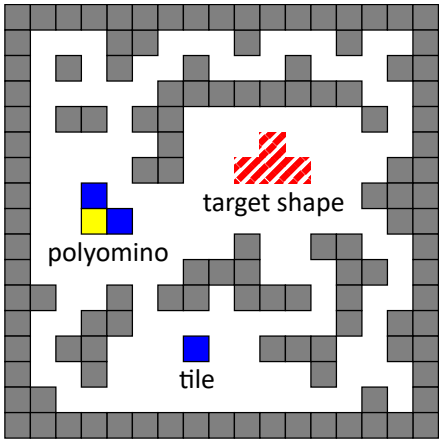


Figure 2.1: Annotated example instance of the Polyomino Assembly Problem. The blocked positions on the board are shown as grey squares and the tiles as colored squares. The three connected tiles form a non-trivial polyomino.

2.1 Problem Definitions

Polyomino Assembly Problem Given a configuration $C = (B, T)$ and a set of open positions $X \subseteq B$ that is connected in G_B the Polyomino Assembly Problem asks for either a step sequence S such that consecutively applying the steps of S to C, results in a configuration in which a polyomino P exists that satisfies $X = \{p_1, p_2, \ldots\}$, where $p_1, p_2, \ldots$ are the positions of the tiles in P, or a proof that the configuration is unreachable.

Fixed Seed Tile Polyomino Assembly Problem Given a configuration $C = (B, T)$, a set of open positions $X \subseteq B$ that is connected in G_B and a fixed tile $t_{\text{fixed}} \in T$ the Polyomino Fixed Seed Tile Polyomino Assembly Problem is defined analogously to the Polyomino Assembly Problem, except that t_{fixed} does not move under step transformation and tiles can only bond if the resulting polyomino contains t_{fixed}.

2.2 Distance Definitions

As a basis for the definition of our heuristic functions, we use the following definitions of distance between positions, tiles and the target shape. Given an instance of the Polyomino Assembly Problem with a configuration $C = (B, T)$, and target shape X. Let $p = (p_1, p_2), q = (q_1, q_2)$ be open positions in B and $t = (p_t, g_t) \in T$.

- $d(p, q)$ is the length of a shortest path from p to q in G_B.
- $d(p)$ is the length of a shortest path in G_B from $p \in B$ to the closest node in X.
- $d_1(p, q) \coloneqq |p_1 - q_1| + |p_2 - q_2|$ is called the taxicab distance.

For the sake of brevity, we define $d(t) \coloneqq d(p_t)$ and analogously for the other distance definitions.

3 Algorithmic Approaches

In this section, several algorithmic approaches to solving the Polyomino Assembly Problem with and without fixed seed tiles are investigated. First, we discuss the hardness of the investigated problems; subsequently, several algorithms are developed. The main focus is on heuristic search-based algorithms, specifically best-first search with various heuristics. Some of the developed heuristics are admissible, meaning that they never overestimate the distance to the target configuration and thus lead to an A* search, which finds sequences of optimal length as solutions, whereas other approaches give up on optimality in exchange for an improved runtime. Furthermore, during the expansion of the search tree, branches that provably cannot lead to a solution can be pruned in order to avoid unnecessary computations. This means, that the configuration represented by these nodes is never used for later expansion steps. Multiple pruning methods are investigated. In addition to the search-based algorithms, we propose an approach using RRTs. Finally, a method for shortening an existing solution is briefly examined.

3.1 Hardness of the Problems

The PSPACE-completeness of the Polyomino Assembly Problem without fixed seed tile follows directly from the PSPACE-completeness of the occupancy problem. For the Fixed Seed Tile Polyomino Assembly Problem, we show, that it is PSPACE-complete even under the conditions, that there are exactly as many tiles on the board as needed to assemble the target shape and the number of glues is constant.

Theorem 1. The Fixed Seed Tile Polyomino Assembly Problem is in PSPACE

Proof. Repeated, non-deterministic selection of control inputs until the target shape is assembled solves the problem and only requires us to store the current configuration. Therefore, the problem is in NPSPACE, which is equal to PSPACE. □

Theorem 2. The Fixed Seed Tile Polyomino Assembly Problem is PSPACE-hard

Proof. To show PSPACE-hardness we adapt a proof from [10] and reduce from the *k-region relocation problem* of the full-tilt model [2]. In this problem, k disjoint regions each contain a single tile. The goal is to move all k tiles to the 1×3 region at the bottom of their respective component using tilt transformations. Analogous to [10], we start the reduction by connecting the 1×3 regions to a single bottom row and invert tile placement by placing tiles on all open spaces that did not contain a tile in the original instance of the k-region relocation problem. Additionally, we place a fixed seed tile k positions to

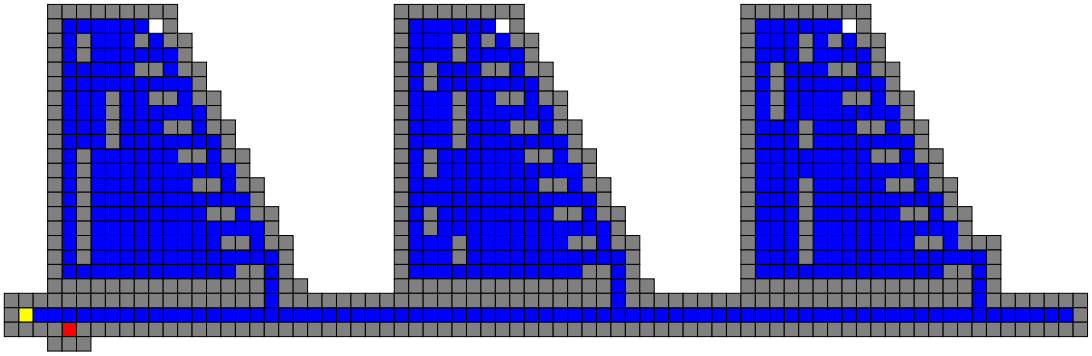


Figure 3.1: Reduction of the k-region relocation problem to the Fixed Seed Tile Polyomino Assembly Problem. The red square is the fixed seed tile, the yellow tile t_{reloc} needs to be relocated 3 positions to the right in order to assemble the target shape. The blue squares are other tiles. Figure recreated and adapted from [10].

the right and 1 down from the leftmost tile in the bottom row t_{reloc}, as shown in Figure 3.1. We assign glues as follows: The fixed seed tile has glue A on all four edges. t_{reloc} has glue B on all edges. Every other tile has glue C on all edges. Furthermore, we define the glue function G, such that $G(A, B) = G(B, C) = G(C, C) = 1$ and $G(X, Y) = 0$ for all other pairs of glues X, Y. Finally, we define the target shape as all open spaces inside the connected region, except for the k leftmost positions in the bottom row. Now a solution to the constructed instance of the Fixed Seed Tile Polyomino Assembly Problem corresponds to a solution to the original instance of the k-region relocation problem. The key idea is, that tiles can only bond if they are connected to the seed tile and that only t_{reloc} can directly bond with the seed tile according to the glue function. Therefore, t_{reloc} must be relocated k positions to the right, which was shown to solve the original k-region relocation problem (in [10]). Conversely, once t_{reloc} has been successfully moved k spaces to the right, the target shape is immediately assembled, as all open spaces, except the k leftmost positions in the bottom row, are filled with tiles that can bond according to the glue function and are connected to the fixed seed tile via t_{reloc}. □

Corollary 1. The Fixed Seed Tile Polyomino Assembly Problem with extra tiles is PSPACE-hard.

Note that this leaves open whether the problem is PSPACE-hard when only a single glue other than the `null` glue is allowed.

3.2 Best-First Search

We explore two different approaches to best-first search. The first approach aims to build the target polyomino directly by minimizing a function of the distances of tiles to

the target shape. The second approach aims to build the target polyomino iteratively by adding one tile after the other, while at the same time attempting to keep the remaining tiles separate.

To avoid repeatedly visiting the same configuration, information that allows the identification of previously visited board positions must be stored. In our implementation, we compute and store hash values based on the positions and glues of all tiles in order to uniquely identify configurations on a given board.

3.2.1 All Tiles at the Same Time

As the basis for the following heuristic functions, the concept of the n nearest available tiles is used.

Given an instance of the Polyomino Assembly Problem with configuration $C = (B, T)$ and a target shape X with $|X| = n$. Let $T_{\text{available}}$ be the set of tiles which are part of a polyomino P for which a path exists that avoids blocked positions in B and along which P can be moved to a position where it is contained in X. Assuming that $|X| \leq |T_{\text{available}}|$, d is the distance of the n-th nearest tile to the target shape in $T_{\text{available}}$ and $T' = \{t \in T_{\text{available}} \mid d(t) \leq d\}$. The branch can be pruned, if $|T_{\text{available}}|$ contains fewer tiles than needed for the target shape

Greatest Distance Heuristic: The Greatest Distance heuristic (GD) is an admissible heuristic that can be used in combination with the A* search algorithm. In the resulting algorithm, the value of the heuristic is added to the distance of the current configuration from the initial configuration, i.e., the number of moves required to get to the current configuration.

Given a configuration C and a target shape X the heuristic function h_{GD} is defined as

$$h_{\text{GD}}(C, X) \coloneqq \max_{t \in T'} d(t). \tag{3.1}$$

It is apparent that the greatest distance among the n nearest tiles is a valid lower bound for the number of required moves because every move can reduce the distance of each tile to the target shape by a maximum of 1 and at least n tiles must reach the target shape in order to complete the assembly. Furthermore, tiles that cannot be part of the target polyomino because they are part of a polyomino that does not fit into the target shape or does not have an unblocked path to the target shape can be ignored. In order to decide if a given tile is in $T_{\text{available}}$, our implementation remembers for each polyomino if it is able to reach the target shape and fits into the target shape. These functions are only reevaluated whenever the polyomino changes. Since polyominoes are expected to change relatively infrequently, the computational overhead should be limited.

A heuristic is called consistent if the estimated distance of a node to the goal is never greater than the estimate for any neighboring node plus the cost of reaching that neighbor. As a consequence, an A* search using a consistent heuristic will find the shortest path to any node when that node is processed for the first time. Since a single

step, which has a cost of 1, applied to a configuration reduces the heuristic value of GD by a maximum of 1 GD is consistent.

Average Distance Heuristic: Similar to GD the Average Distance heuristic (AD) is computed based on the distance of the n nearest tiles that can potentially be part of the target shape. A disadvantage of GD is that it only takes into account the greatest distance among the n nearest available tiles and disregards the distances of the closer tiles, which can cause GD to overestimate moves that bring the most distant tile closer but increase the distance of many other tiles to the target shape. The Average Distance heuristic attempts to also take into account the distance of the closer tiles by using the arithmetic mean.

$$h_{\mathrm{AD}}(C, X) \coloneqq \frac{\sum_{t \in T'} d(t)}{|T'|} \tag{3.2}$$

Following the same logic as above, h_{AD} is also a consistent heuristic.

Gready Greatest Distance and Greedy Average Distance Heuristic: Both h_{GD} and h_{AD} can be used in combination with a greedy best-first search approach by not adding the distance from the initial configuration to the heuristic value. If the heuristics are used in this context, we refer to the resulting algorithms as Gready Greatest Distance (GGD) and Greedy Average Distance (GAD) respectively. The idea of this approach is to give up the optimality of the solution in order to potentially improve the execution speed.

Weighted Sum of Distances Heuristic: The Weighted Sum of Distances heuristic (WSD) is a non-admissible heuristic, that is an attempt at a generalization of GGD and GAD. While GGD only takes into account the distance of the most distant tile, in GAD the distances of all n nearest available tiles each have the same impact on the heuristic value. WSD allows adjusting the influence that a tile has on the heuristic value based on its distance to the target shape by introducing an exponent $e \in [1, \infty)$.

$$h_{\mathrm{WSD}} \coloneqq \sum_{t \in T'} d(t)^e \tag{3.3}$$

For $e = 1$ WSD behaves equivalently to GAD, whereas for greater e, tiles with a greater distance to the target shape have an increased impact on the heuristic function. For a sufficiently large e, the heuristic value is essentially defined by the greatest distance among tiles and WSD behaves similar to GGD.

Pruning methods: There are multiple pruning methods that can be used together with the all-tiles-at-the-same-time approach. Firstly, if $|T_{\mathrm{available}}| < n$, the configuration can never lead to a solution. Furthermore, when it can be shown that there is no subset of existing polyominoes which can exactly cover the target shape, the branch can be pruned. Generally, however, it may not be computationally feasible to decide whether

such a tiling of the target shape exists every time a polyomino changes. In the case where there are exactly as many tiles on the board as needed to form the target shape, we determine if the k largest polyominoes can be packed into the target shape. If this is not possible then the branch can be pruned. In our implementation $k = 3$ is selected.

Alternative Stop Condition: When the configuration contains exactly as many tiles as needed for the target polyomino, an alternative stop condition can be utilized, which sometimes gets to a solution after fewer expanded nodes. If at any point during the heuristic search a configuration C contains a single polyomino P that fits exactly into the target shape and for which a path to the target shapes location exists, the sequence leading to C, concatenated with a shortest sequence that moves P from its current location to the location of the target shape is a solution to the problem. The later sequence can be found through a breadth-first search.

A solution found in this way is not necessarily of optimal length, even if one of the A* search approaches with a consistent heuristic is used. However, it can only be longer than the optimal solution by at most d_{max} steps, where d_{max} is the greatest distance between any two open positions in B.

3.2.2 One Tile at a Time

In contrast to the all-tiles-at-the-same-time approach, the one-tile-at-a-time approach uses multiple consecutive best-first searches in order to add tiles one after another to a polyomino. For that reason, the heuristic function used for each of the separate best-first searches depends on the positions of the subassembly and the single tile that it is trying to combine. For that purpose, an implementation of the one-tile-at-a-time approach first needs to compute a set of tiles that can form the target shape and a building order in which these tiles can be added one at a time to construct the target polyomino. Each best-first search attempts to keep all tiles that are not involved in the current construction step separated from each other by pruning configurations with unwanted subassemblies. Importantly, this approach is not a complete solution to the (Fixed Seed Tile) Polyomino Assembly Problem, since it is not always possible to avoid creating multiple subassemblies even if the initial configuration only consists of trivial polyominoes (see Figure 3.2). Furthermore, a found solution is unlikely to be optimal, even if each construction step consists of an A* search with a consistent heuristic. Nevertheless, this method can be effective in many cases.

All the following heuristics are used in combination with an A* search.

Minimum Moves to Polyomino heuristic: Given a configuration $C = (B, T)$, a target shape X and a Polyomino P, the Minimum Moves to Polyomino (MMP) heuristic provides a lower bound on the number of moves required to move a selected tile $t = (p_t, g_t)$ to a position defined relative to the position of P. Let x be the absolute coordinates of the target position in the current configuration. For the purpose of the best-first search, the selected target position is always adjacent to a tile in P. Assuming that the absolute

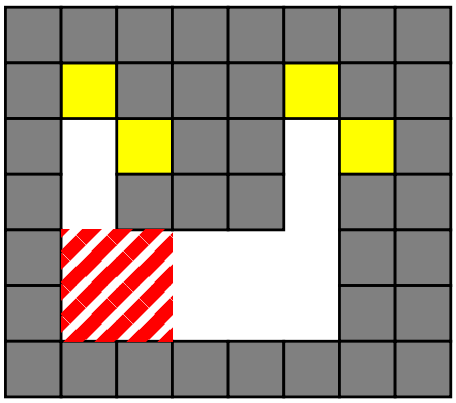


Figure 3.2: Example instance that requires multiple subassemblies to assemble the target shape (red-striped area) and thus cannot be solved using the one-tile-at-a-time approach.

target position does not overlap with an obstacle, the heuristic function is defined as follows:

$$h_{\mathrm{MMP}}(C, X, t, x) \coloneqq \left\lceil \frac{d(t, x) - d_1(t, x)}{2} \right\rceil + d_1(t, x) \tag{3.4}$$

h_{MMP} is a lower bound on the number of moves required to move t to the target position relative to P.

Proof. A single step reduces the length of a shortest path from t to the target position by a maximum of 2 because both t and P each move at most one unit distance. However, a move that reduces the distance by 2 can never reduce the taxicab distance between t and the target position because both t and P must move during the step in order for the distance to be reduced by 2. By the definition of the step transformation, both t and P must move 1 unit in the same direction. Therefore, the taxicab distance remains unchanged. Furthermore, this implies that a single step can reduce the taxicab distance by a maximum of 1. In this case, one of the components must be stationary and therefore the length of a shortest path is reduced by at most 1 too. Therefore, a solution needs to contain at least $d_1(t, x)$ moves that reduce the length of a shortest path by 1. This leaves a distance of at least $d(t, x) - d_1(t, x)$, which can be reduced by a maximum of 2 per step. □

Note that the target position can overlap with an obstacle that the target polyomino is adjacent to. In this case, the length of the shortest path to x is replaced by the length of the shortest path to an adjacent position minus 1.

A disadvantage of this heuristic is that it requires knowledge about the pairwise distance of all open positions on the board. In our implementation, these distances are

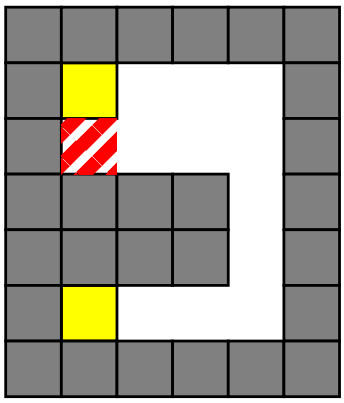


Figure 3.3: An example for the MMP distance calculation. The length of the shortest path between the bottom tile and the target position is 9. The taxicab distance is 3. Therefore, at least $(9 - 3) / 2 + 3 = 6$ steps are needed to combine the tiles in the intended way.

computed in the beginning through breadth-first searches starting on each open position. This is only feasible for small boards as the calculation is $O(m^3)$ where m is the number of open positions, and the memory is $O(m^2)$.

Minimum Moves to Polyomino and Target heuristic: The Minimum Moves to Polyomino and Target heuristic (MMPT) tries to mitigate the requirement to compute pairwise distances. For that purpose, the number of required moves as estimated by MMP is only used for the calculation of the heuristic value if both the current subassembly and the selected tile are within some threshold distance d_t of the target area. We define the target area A, as the set of all open positions that the finished target polyomino can cover with any of its tiles after a sequence of steps. The target area can be computed with a breadth-first search. The distance of a position p to the target area is called $d_\mathrm{A}(p)$ and defined as the length of a shortest path from p (or the nearest open position that p is adjacent to), to the nearest position in the target area. If the current target position or the selected tile is not near the target area, the maximum distance of the two components to the target area is used as a basis for the heuristic value instead. In this case, to prioritize configurations where both components are near the target area, we apply a factor $w = |A|$. w is proportional to the size of the target area to account for the greater possible distances within a larger target area. Alternatively w could be selected as the greatest distance within the target area.

$$h_\mathrm{MMPT}(C, X, t, x) \coloneqq \begin{cases} h_\mathrm{MMP}(C, X, t, x), & \text{if } d_\mathrm{A}(p_t) \leq d_t \text{ and } d_\mathrm{A}(x) \leq d_t \\ w \cdot \max(d_\mathrm{A}(p_t), d_\mathrm{A}(x)), & \text{otherwise} \end{cases} \tag{3.5}$$

This approach has two main advantages over MMP. Firstly, it only requires the computation of pairwise distances within the proximity of the target area. Secondly, MMP does not consider in what position the subassembly is formed in relation to the final position of the target shape. This can lead to the formation of subassemblies in locations from where they cannot reach the target shape anymore. In the MMP approach, such branches get pruned; however, the heuristic function does not actively avoid the creation of subassemblies outside the target area. In contrast, MMPT prioritizes forming subassemblies within the reachable area of the target shape.

Note that MMPT is not a consistent heuristic.

Distance to Fixed Position heuristic: In the case of a fixed seed tile, a much simpler heuristic can be used as a lower bound on the number of required moves to combine a selected tile with the fixed target polyomino. Because the target position of the tile is fixed, each move can reduce the length of a shortest path from the tile to the target position by a maximum of 1.

$$h_{\mathrm{DFP}}(C, X, t, x) \coloneqq d(t, x) \tag{3.6}$$

Additionally, during the computation of the shortest paths to the target position, the positions of tiles contained in the fixed polyomino, as well as neighboring positions that are impassable for the selected tile because of glues on the edges of a fixed tile can be marked as blocked.

Pruning methods: All of the one-tile-at-a-time approaches require the same two pruning methods. Firstly, a branch is pruned if its configuration contains an unwanted subassembly. Secondly, a branch is pruned if the correct subassembly is present, but is in a location from where it cannot reach a position where it is contained in the target shape.

Computation of the Building Order: To compute an order in which tiles can be added to the polyomino, a tiling of the target shape such that all tiles bond is determined using a brute-force algorithm. Then a recursive backtracking algorithm that removes tiles from the polyomino one after the other is utilized. If all tiles can be removed from the polyomino in this way the reversed deconstruction order is a *potential* building order. To determine if a tile can be removed two conditions are checked.

1. Do the remaining tiles form a polyomino that is connected by glues?
2. Does a path exist, along which the selected tile can be moved outside of the rectangular region minimally enclosing the polyomino, without being blocked by other tiles or the glues on their edges? This is determined through a breadth-first search.

This approach is directly inspired by research on the constructibility of polyominoes under tilt transformations [7]. If this algorithm does not find a potential building order, it is attempted again with a different tiling of the polyomino until a potential building

order is found or all possible tilings have been attempted. A building order found in this way is not guaranteed to be realizable on the board of the problem instance. If any of the best-first searches during the motion planning process terminate without finding a solution, the solver is restarted with another building order.

3.3 RRT

The basic idea of an RRT search is to select a random configuration from the configuration space and expand the configuration in the current tree that is closest according to some cost-to-go function towards the selected configuration. When designing an RRT-based motion planning algorithm for the Polyomino Assembly Problem, the main challenge is to find a cost-to-go function that is fast to compute and provides a reasonable estimate of the distance between two configurations.

Let $C_1 = (B, T_1)$ and $C_2 = (B, T_2)$ be two configurations on the same board. As a first approach, we constructed a bipartite graph $G = (U, V, E)$ in which the vertices in U represent the tiles from T_1 and the vertices in V the tiles from T_2. A vertex in U is connected to each vertex in V that represents a tile with the same glues. The weight of the edge is the distance between the positions of the tiles.

Now the cost-to-go from C_1 to C_2 is estimated by the weight of the longest edge in a bottleneck matching. A bottleneck matching is a perfect matching that minimizes the length of the longest edge between U and V. This is a lower bound on the number of steps needed to transform C_1 into C_2 because every tile in C_1 must be moved to a unique target position that contains a tile with the same glues in C_2 and each move can reduce the distance of each tile to its target position by a maximum of 1. However, the cost-to-go function needs to be evaluated many times during each expansion step of the RRT, and computing a bottleneck matching may be too expensive. Therefore a cost-to-go function that is quicker to compute was investigated.

For two tiles $t_1 = (p_1, g_1) \in T_1$ and $t_2 = (p_2, g_2) \in T_2$ we define

$$d_H(t_1, t_2) \coloneqq \begin{cases} d(p_1, p_2), & \text{if } g_1 = g_2 \\ \infty, & \text{otherwise} \end{cases} \tag{3.7}$$

Then we define the distance between a tile t and a set of tiles S as $D_H(t, S) \coloneqq \min_{s \in S} d_H(s, t)$. Note that $D_H(t_1, T_2) < \infty$ for any $t_1 \in T_1$, and vice versa, because both sets of tiles have the same sets of glues.

Now the cost-to-go-function between $C_1 = (B, T_1)$ and $C_2 = (B, T_2)$ is defined equivalently to the Hausdorff distance [14] between the two sets of tiles.

$$D_H(C_1, C_2) \coloneqq \max\{\max_{t_1 \in T_1} d(t_1, T_2), \max_{t_2 \in T_2} d(t_2, T_1)\} \tag{3.8}$$

Furthermore, if a polyomino exists in C_1 that does not fit into any polyomino of C_2 we set the cost-to-go to ∞.

Since the number of times the cost-to-go function is evaluated in each expansion step grows with the number of nodes in the RRT, we try to further mitigate the computational

cost of repeatedly evaluating the cost-to-go function by keeping a sparser tree. This is achieved through a more expensive expansion step, which expands the closest node by multiple steps in the direction of the randomly selected configuration. For that purpose, a greedy best-first search with a limited number of iterations is used to minimize the cost-to-go.

A downside of both discussed cost-to-go functions is that they require knowledge about the pairwise distances between positions of the board. A potential solution is to use the taxicab distance instead of the length of a shortest path as the underlying distance metric. However, this would further decrease the accuracy of the estimated distance.

To estimate the distance to the goal, i.e., any configuration containing the target shape, a different metric must be used because the order of tiles within the target shape and the position of possible left-over tiles are not known. Therefore we choose the greatest distance among the n nearest available tiles, as defined by the GD approach, as an estimate for the distance to the goal. The RRT nodes are kept in a list ordered by the distance to the goal. We bias the RRT search to expand the closest viable node directly towards the goal in 5% of the expansion steps. For these expansion steps, the GGD approach with a limited number of iterations is used. If the distance from the selected node to the goal cannot be reduced in this way, the node is marked as dead and the next expansion towards the goal will use the next closest node.

3.4 Solution Shortening

Given two configurations C_1 and C_2 in which the tiles have the same positions relative to each other. Using a breadth-first search we can efficiently determine whether a sequence of inputs exists that transforms C_1 into C_2 and that preserves the relative positions of tiles after every step. This can be used as a way to shorten a solution found by a motion planner as follows.

Start at the first position of the original sequence and find the last configuration that contains the same relative positions of tiles as the current configuration. If a sequence of steps between the two configurations shorter than the original sequence can be found using the above method, replace this part of the solution with the shorter sequence. Continue this process for the rest of the sequence.

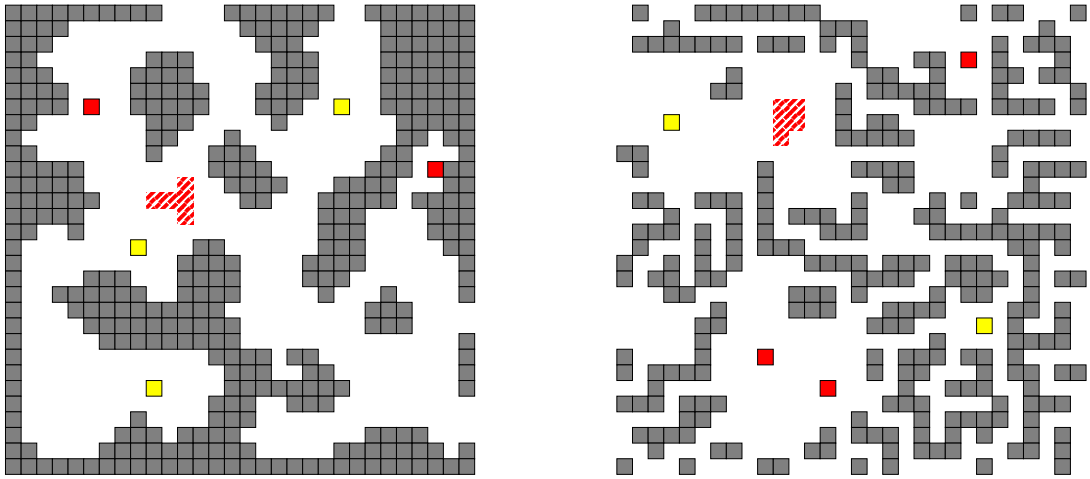
4 Experiments

In this chapter, the presented algorithmic approaches are experimentally evaluated on sets of procedurally generated instances with different features, including instances of both the problem with and without fixed seed tile. Furthermore, we investigate the impact of several parameters, such as the number of tiles, the size of the board, the placement of obstacles, etc., on the difficulty of the problem. The main focus of the analysis is on the comparison of runtimes and the quality (length) of the found solutions. Additionally, the memory requirements of our implementations and the effectiveness of the proposed solution shortening method are briefly evaluated.

4.1 Method for Instance Creation

To create a sufficient number of instances with a range of different features for experimentation, we implemented a Python function, that procedurally generates such instances based on six input parameters.

1. **Board type:** This parameter decides which randomized algorithm is used to determine the blocked positions on the board. Two different board types were used for the experiments. The first type is called *maze* and boards of this type are created in a two-step process. Firstly, starting on a board filled with blocked positions, a randomized recursive backtracking algorithm creates a tree of open spaces. Secondly, a number of rectangular open regions proportional to the size of the board is added uniformly at random. The second type is called *cave*. Boards of this type are created using cellular automata, with the following rules: A dead cell becomes alive in the next generation if it has 5 or more living neighbors. A living cell becomes dead in the next generation if it has less than 4 living neighbors. After applying two generations of these rules to a grid where each cell starts alive with a probability of 0.45, the positions of the living cells correspond to the blocked positions on the cave board. Additionally, to ensure connectedness, the largest connected component of open spaces is determined and all other open spaces are filled. The idea is to create boards with different properties in order to evaluate the influence of obstacle placement on the motion planning algorithms. Maze boards contain open areas, that are connected through narrow corridors; whereas cave boards contain fewer narrow sections.
2. **Target shape size:** The number of tiles that are required to assemble the target shape.
3. **Board size:** For the experiments, only square-shaped boards were used. The board size determines the edge length of the board.



(a) Example cave board instance

(b) Example maze board instance

Figure 4.1: Example instances for the two different board types. The grey squares represent blocked positions and the colored squares are tiles. The red-striped area marks the target shape. The border is implicitly considered as blocked.

4. **Number of extra tiles:** The number of tiles added to the initial configuration which are not needed to create the target shape. The total number of tiles is the sum of the target shape size and the number of extra tiles.

5. **Number of glues:** The number of different glues. The glues on the edges of each tile are selected uniformly at random from the set of available glues. To make it more likely for instances to have a solution, a subset of the tiles is arranged into the target shape and a set of rules is added such that they bond. Finally, additional rules are selected at random, until at least half of all possible combinations of glues stick together.

6. **Problem type:** The problem type determines whether a fixed seed tile exists or not. If a fixed seed tile exists, tiles can only bond, if the resulting polyomino contains the fixed tile.

To determine the initial tile positions, tiles are successively placed on legal positions uniformly at random. A position is considered legal, if it is open, does not contain a tile, and is not adjacent to a position containing a tile. The last condition ensures that the initial configuration only contains 1×1 polyominoes. If a fixed seed tile is required, it is placed first on a suitable position within the target shape. On all generated boards, the open positions form a connected region. Although some measures were taken to increase the probability that the created instances are solvable, instances are not guaranteed to have a solution.

4.2 Experimental Setup

This section explains the methods and tools we used for the experiments and provides an overview of the evaluated algorithmic approaches.

4.2.1 Simulation Environment

For the experiments, all discussed motion planning algorithms were implemented in a self-developed simulator, which is based on TumbleTiles [1]. The original TumbleTiles software was upgraded from Python 2 to Python 3 and heavily modified. It is capable of simulating step transformations on configurations with and without fixed seed tile. Furthermore, we implemented support for custom glue functions. Additionally, the software features a GUI that allows the user to view and edit a configuration, animate step sequences, and execute the various motion planning algorithms.

The experiments were conducted on multiple computers, each with the same specifications (**Intel(R) Core(TM) i7-6700K CPU @ 4x4.00GHz, 64GB RAM**) running Ubuntu 20.04 LTS. Multiple motion planning algorithms were evaluated at the same time using different CPU cores.

4.2.2 Evaluated Approaches

This subsection provides an overview of the 10 evaluated motion planning algorithms and the abbreviations that will be used to refer to them in the following. Furthermore, details about the used parameters are listed if required.

1. Breadth-first-search (BFS)
2. Greatest Distance heuristic (GD)
3. Average Distance heuristic (AD)
4. Greedy Greatest Distance heuristic (GGD)
5. Greedy Average Distance heuristic (GAD)
6. Weighted Sum of Distances heuristic (WSD): $e = 2$.
7. Minimum Moves to Polyomino heuristic (MMP)
8. Minimum Moves to Polyomino or Target area heuristic (MMPT): $d_t = 4$.
9. Distance to Fixed Position heuristic (DFP)
10. Rapidly-exploring Random Tree (RRT): The Hausdorf-distance based on the shortest path length is used as the cost-to-go function. Each expansion step consists of 40 iterations of a greedy best-first search. The bias towards the goal is 5%. Below a threshold distance of 7 or less to the goal, a best-first search limited to 500 iterations attempts to find a solution.

[1] TumbleTiles on Github: `https://github.com/asarg/TumbleTiles`

For each solver, all applicable pruning methods discussed in Chapter 3 were used. Additionally, in combination with the all-tiles-at-the-same-time heuristic approaches (GD, AD, GGD, GAD, WSD) the alternative stop condition from Section 3.2.1 was used, if the configuration contained no extra tiles.

4.2.3 Experimental Procedure

The motion planning algorithms were evaluated on two sets of instances, which were created in advance and contain the same instances for every experiment.

`InstanceSet1` consists of relatively easy to solve instances, that allow a comparison of all solvers, including the breadth-first search solver and other (near-)optimal solvers. It contains 5 procedurally generated instances for each possible combination of parameters from the following table.

Board type	cave, maze
Target shape size	$3, 4, 5, 6$
Board size	$20, 30, 40$
Extra tiles	$0, 1, 3$
Number of glues	$1, 2, 3$
Problem type	seed tile, no seed tile

`InstanceSet2` consists of instances with a wider range of difficulties. It contains 5 procedurally generated instances or each possible combination of parameters from the following table.

Board type	cave, maze
Target shape size	$5, 10, 13, 15$
Board size	$40, 80, 120$
Extra tiles	$0, 3, 5$
Number of glues	$1, 3, 5$
Problem type	seed tile, no seed tile

Both sets contain a total of 2160 instances each. The instances from `InstanceSet1` were attempted to solve with all 10 evaluated algorithms, whereas only selected solvers (GAD, WSD, MMPT, DFP) were used for InstanceSet2. DFP was only used on suitable instances, i.e. instances with a fixed seed tile.

All experiments were conducted with a timeout of 600 seconds per instance, after which the motion planning algorithm was stopped if it did not find a solution or proved that no solution exists.

Along with the problem instance, the solution sequence (if solved), the runtime, and the peak memory usage were recorded. Additionally, the number of nodes in the search tree, or in the case of one-tile-at-a-time algorithms the sum of nodes in all search trees, was recorded.

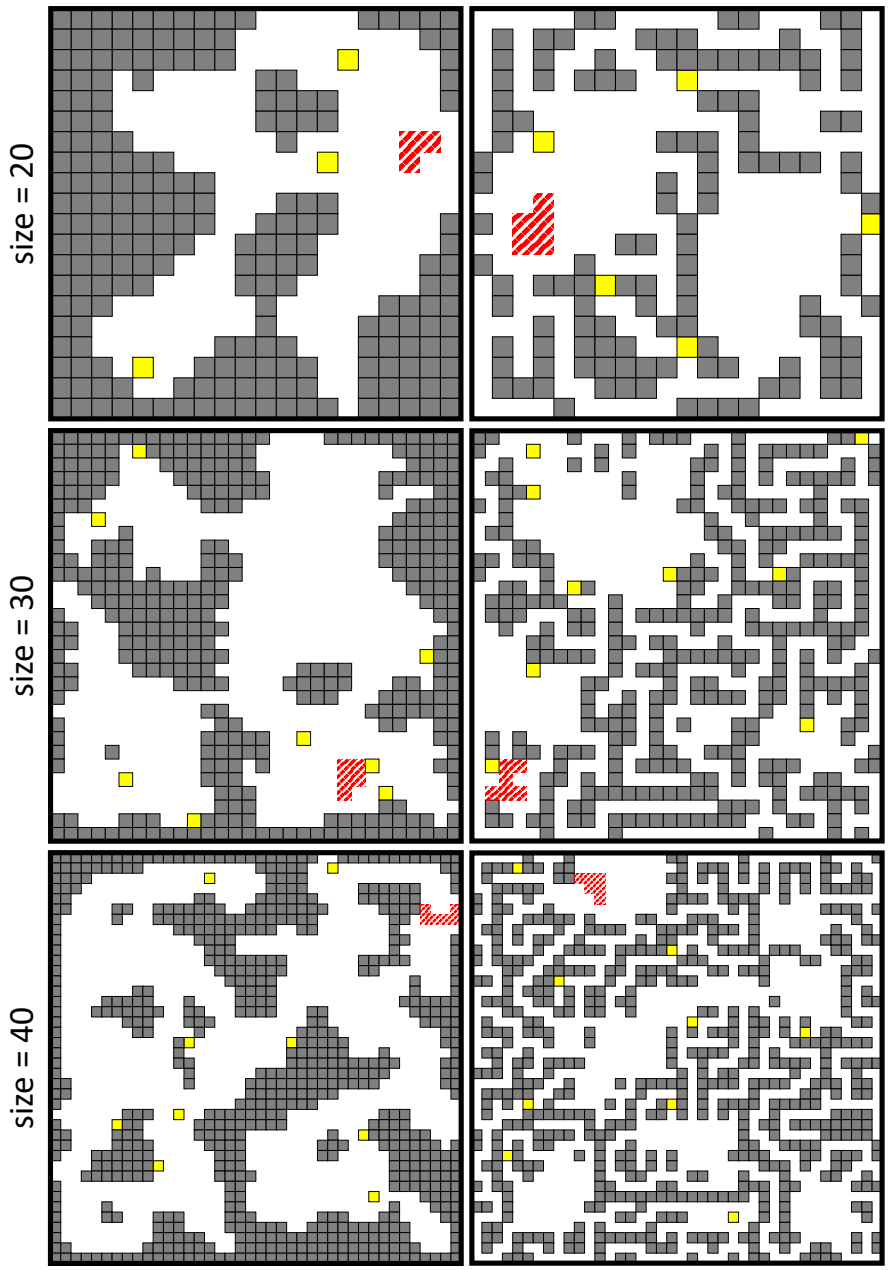


Figure 4.2: Six randomly generated instances of different board sizes from `InstanceSet1`, three cave boards on the left and three maze boards on the right. The yellow squares are tiles and the red-stripped polyominoes are the target shapes. Glues are not shown.

4.3 Results

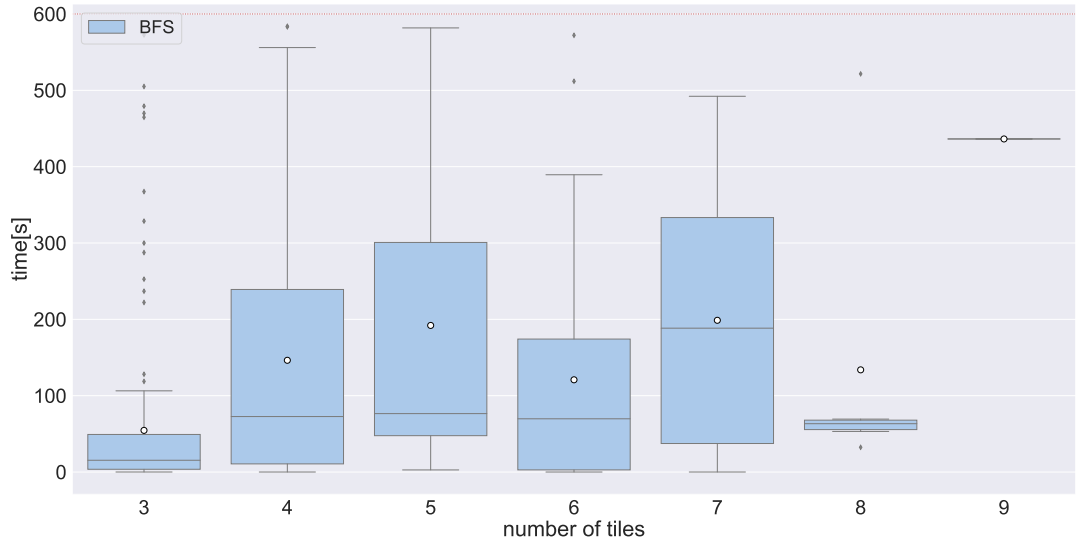
In this section, our algorithmic approaches are evaluated using the experiment results from both sets of instances.

4.3.1 Evaluation of `InstanceSet1`

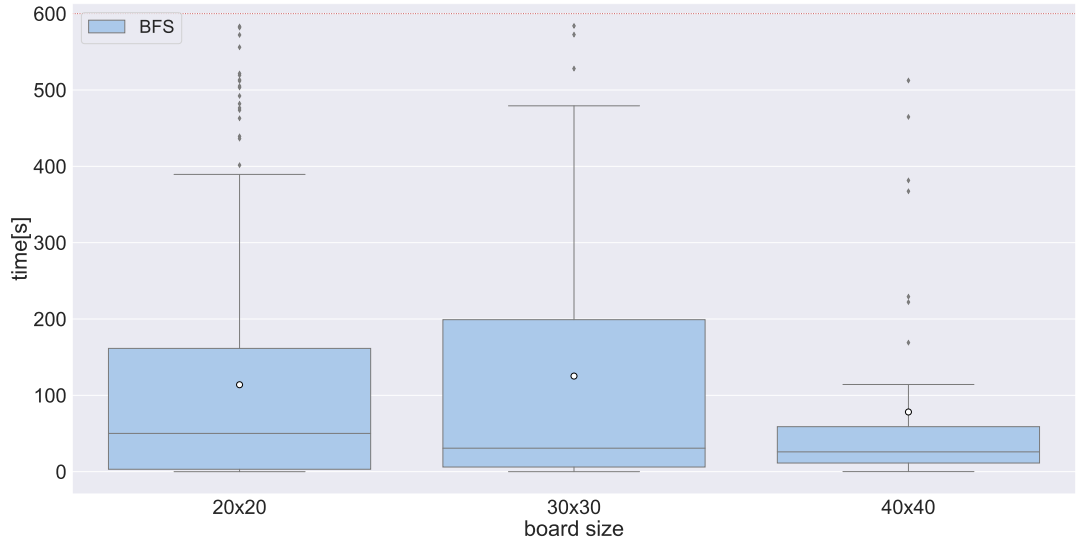
The experiment results from `InstanceSet1` are used to measure the impact of various parameters on the performance of different motion planning algorithms. The relatively small size of the instances allows every solver including the breadth-first search approach and other near-optimal motion planners to solve a large fraction of instances within the timeout of 10 minutes. This provides a baseline for the performance of the other solvers. Additionally, this data is used to evaluate the impact of several configuration parameters on the efficiency of the different motion planning algorithms and to compare the memory requirements of the different approaches as well as the effectiveness of our solution shortening method.

Performance of breadth-first search

Figures 4.3, 4.4 show the runtime distribution and the fraction of instances that were successfully solved within the timeout for instances with a small number of tiles. Note that the performance decreases sharply with an increased number of tiles. At 3 tiles, more than 70% were successfully solved or shown to have no solution, whereas at 5 tiles less than 10% of instances were finished before the timeout. Only one instance with 9 tiles was successfully solved. The size of the board also had a significant impact on the performance of the solver. Furthermore, a majority of the solved instances were solved long before the timeout.
The length of the solutions found by the BFS solver seems to decrease when the number of tiles on the board increases, as shown in Figure 4.5. There are two possible reasons for this. Firstly, an increased density of tiles on the board allows shorter solution sequences for target shapes of the same size. Secondly, for a larger number of tiles, the solver only found a solution before the timeout if a relatively short solution existed.
In contrast, the solution length significantly increases with increasing board size.

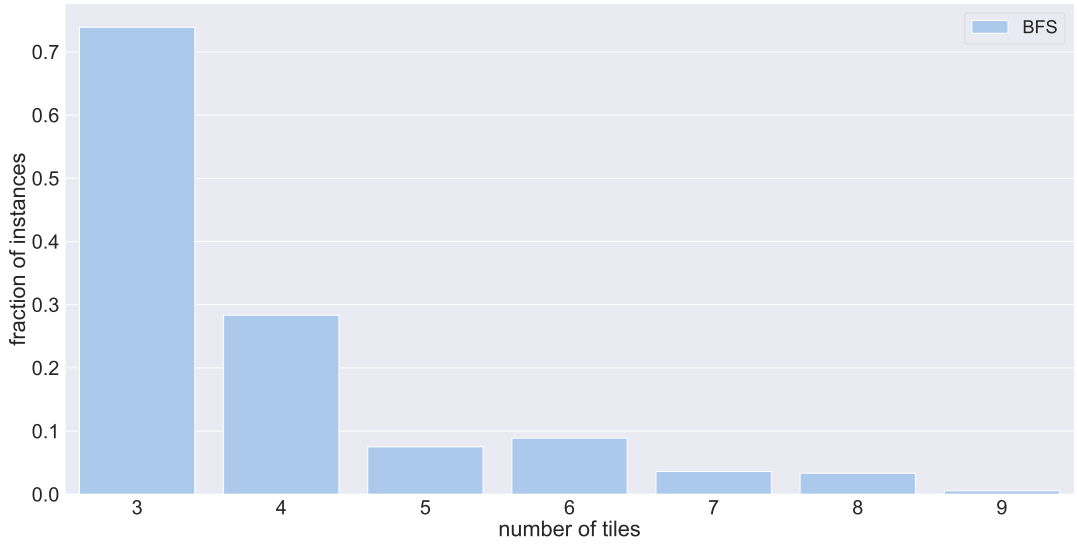


(a) Runtime distributions by number of tiles as box plot. Only successful solutions are shown.

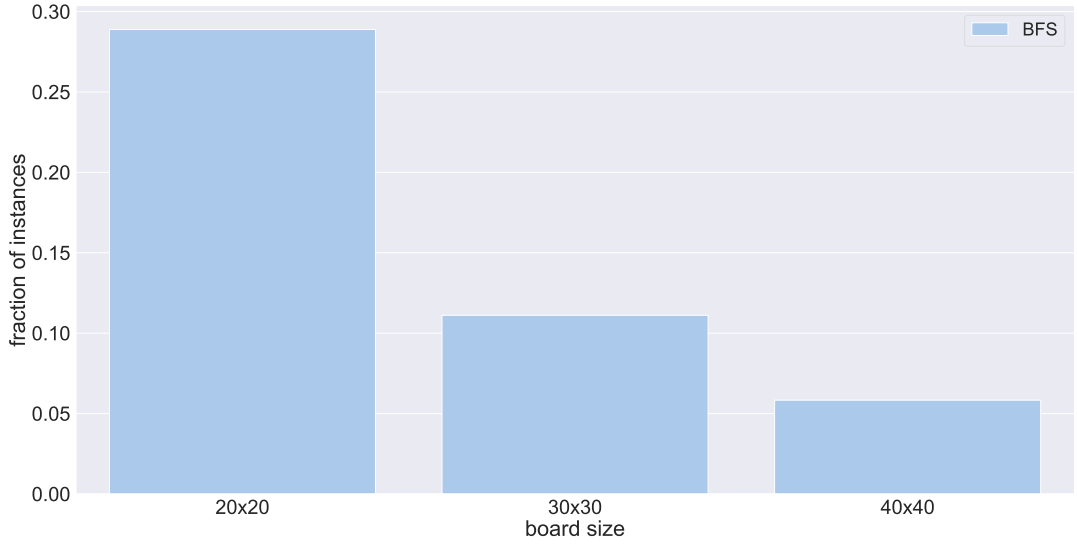


(b) Runtime distributions by board size as box plot. Only successful solutions are shown.

Figure 4.3: Runtime of the Breadth-first-search (BFS) motion planner on `InstanceSet1` by number of tiles and board size. The boxes show upper and lower quartile, the line the median, the whiskers extend to the most extreme value that is within a proportion of 1.5 of the interquartile range. The white dot indicates the arithmetic mean and the other dots outliers. All experiments were conducted with a 600 seconds timeout per instance, as shown by the dotted red line.

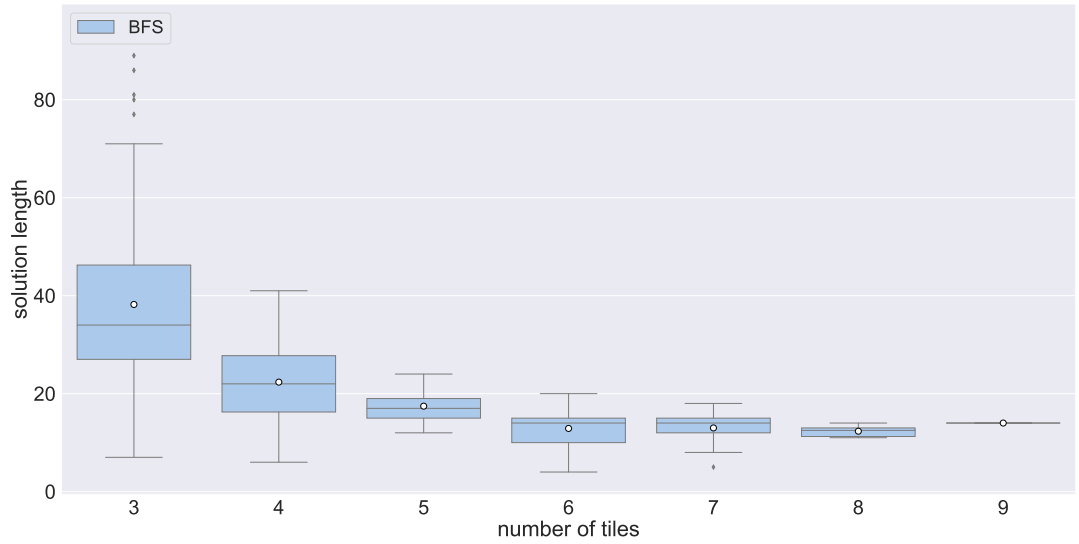


(a) Fraction of solved instances by number of tiles.

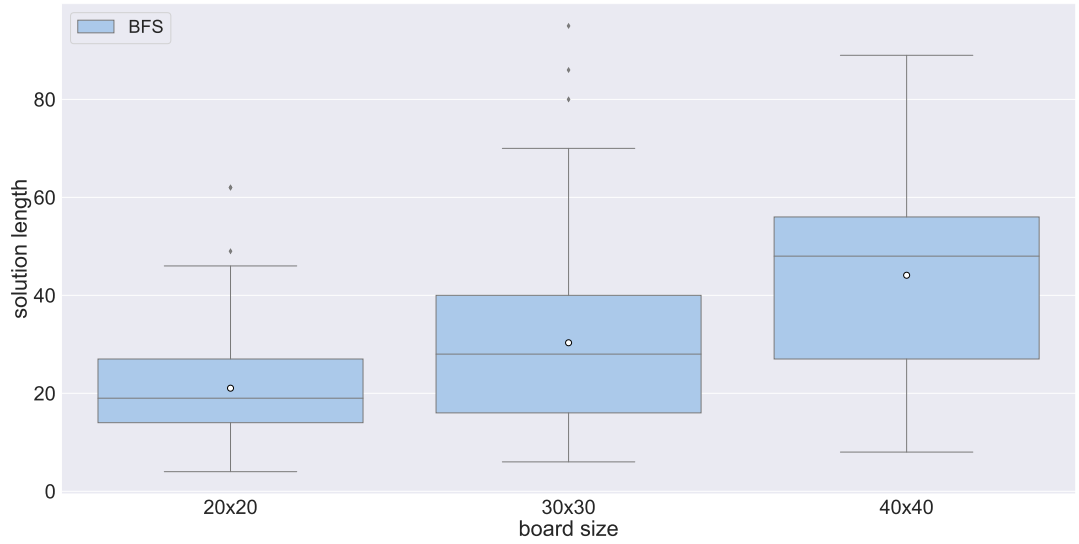


(b) Fraction of solved instances by board size.

Figure 4.4: Fraction of instances from `InstanceSet1` that were solved by the BFS motion planner or for which the solver determined that no solution exists by number of tiles and board size.



(a) Distribution of solution length by number of tiles as box plot.



(b) Distribution of solution length by board size as box plot.

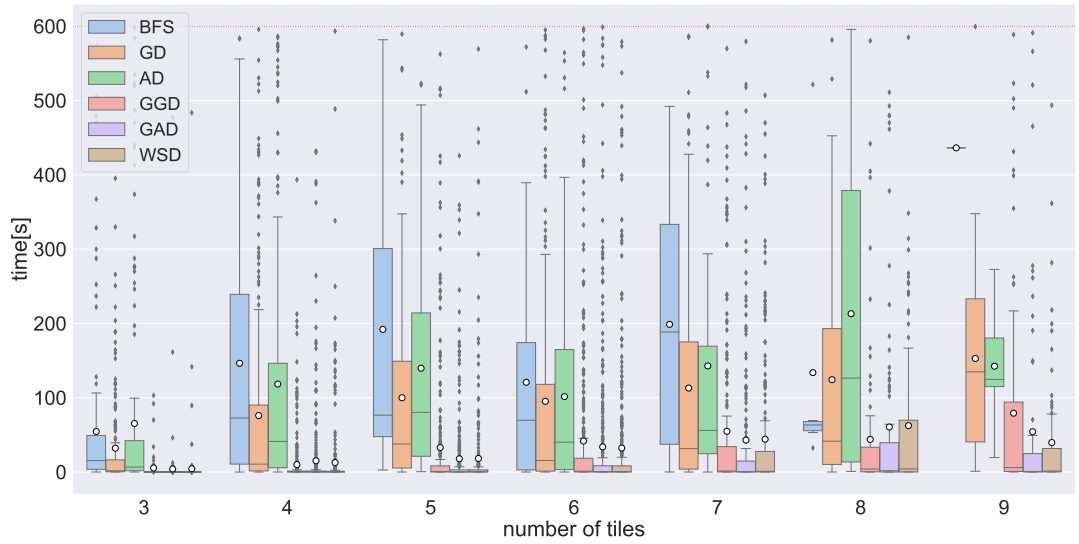
Figure 4.5: Solution length for the BFS motion planner on `InstanceSet1` by tiles and board size. Only instances for which a solution was found are taken into account.

Comparision of the all-tiles-at-the-same-time search-based motion planning algorithms

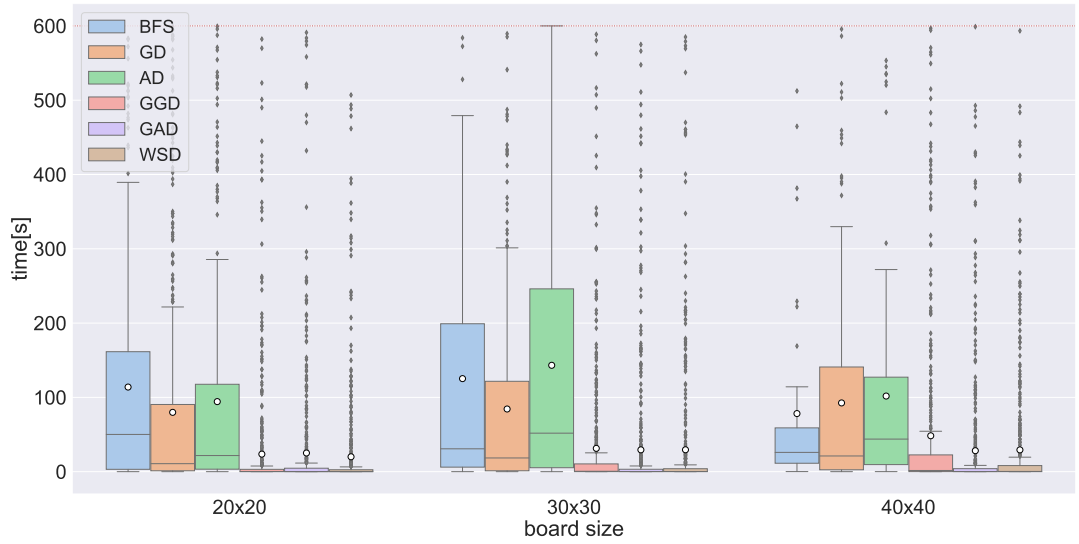
In this subsection, the performance of the BFS solver and the search-based motion planning algorithms GD, AD, GGD, GAD, and WSD are compared. All of these algorithms are best-first searches that try to minimize a function of the distance of multiple tiles to the target shape.
As expected, all heuristic searches show a better performance in terms of runtime and success rate than breadth-first search (see Figures 4.6, 4.7). The A* searches with consistent heuristic GD and AD take clearly longer to find a solution and have a lower success rate, especially on instances with greater numbers of tiles, than the greedy best-first search approaches. Interestingly, the GD algorithm seems to perform better than the AD algorithm, whereas the greedy algorithms GGD, GAD, and WSD all show a similar performance. All solvers show a clear decrease in success rate with an increasing number of tiles, whereas for the investigated board sizes GAD and WSD show low sensitivity to a change in board size. Overall, the number of tiles seems to have the greatest impact on the performance.
The near-optimal motion planning algorithms expectedly produce solutions of roughly the same length as shown in Figure 4.8. On the other hand, the solution length for the greedy algorithms was much longer on average and they show clear differences in the quality of the found solutions. The solutions found by GAD and WSD are usually shorter than solutions found by GGD. Again, the size of the board seems to have a greater impact than the number of tiles on the length of the found solution. However, the solution length of GGD seems to be more sensitive to the number of tiles than other approaches. Since the heuristic value of a configuration in GGD only represents the distance of a single tile to the target shape, the distance of a growing number of tiles to the target shape is ignored, when a large number of tiles is present, which may explain this trend.
On instances where both AD and GAD found a solution, the solution of the greedy approach was longer than the near-optimal solution of the A* approach by a factor of 2.96 on average. However, this factor decreases with an increase in board size as the following table indicates.

Board size	GAD/AD solution, mean ratio
20×20	3.24 ± 3.78
30×30	2.64 ± 2.31
40×40	2.49 ± 2.52

Table 4.1: Mean ratio of the length of a solution found by GAD and the solution to the same instance found by AD to 3 significant figures ±SD for different board sizes.

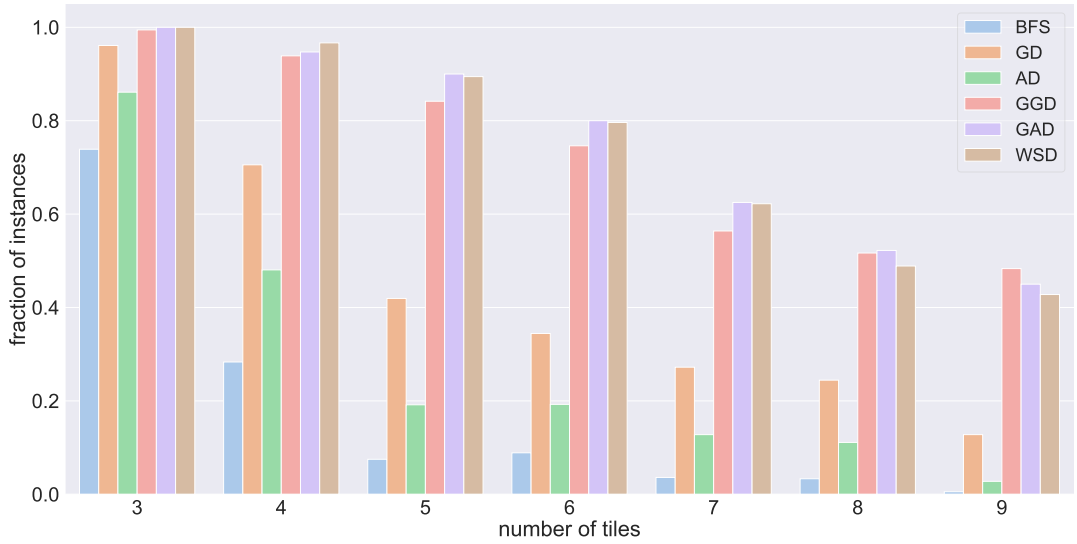


(a) Runtime distributions by number of tiles as box plot. Only successful solutions are shown.

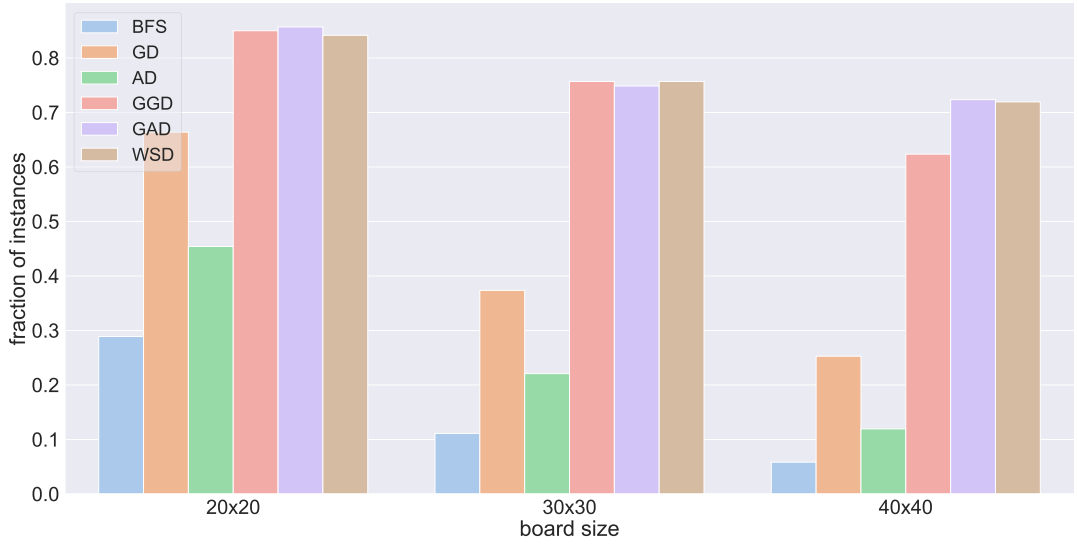


(b) Runtime distributions by board size as box plot. Only successful solutions are shown.

Figure 4.6: Comparison of the runtime of search-based heuristic motion planning algorithms on `InstanceSet1` by number of tiles and board size.

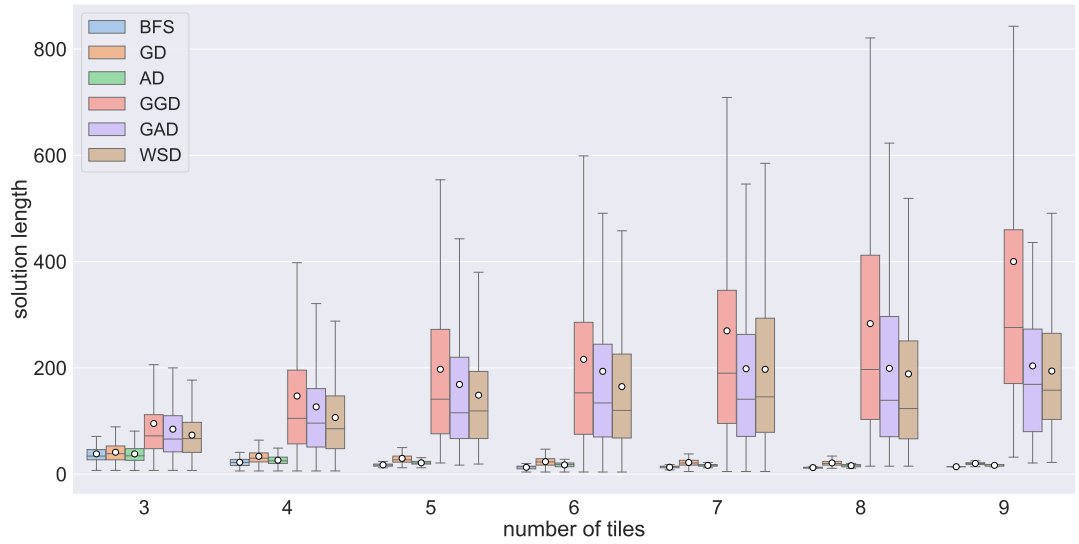


(a) Fraction of solved instances by number of tiles.

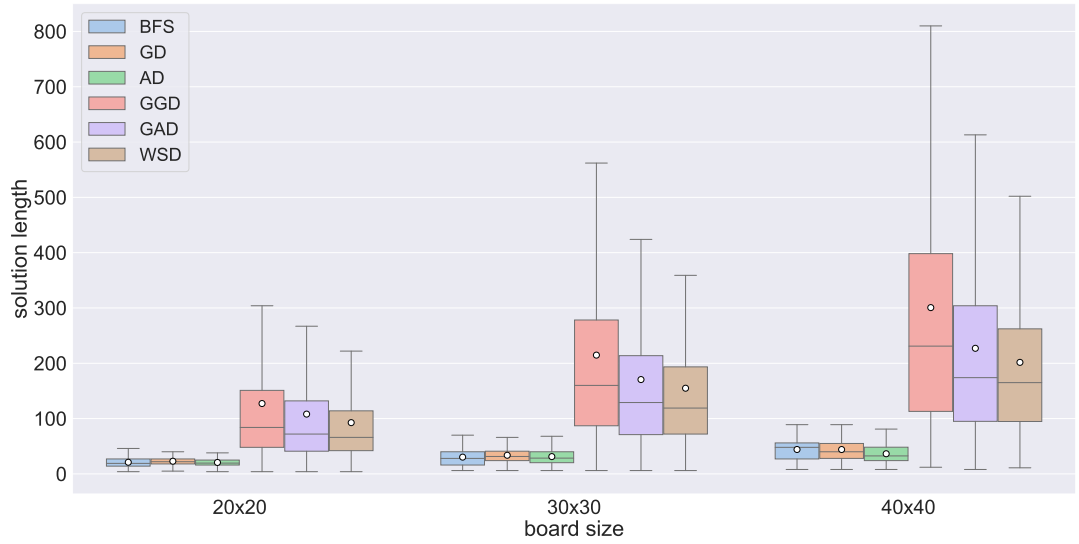


(b) Fraction of solved instances by board size.

Figure 4.7: Fraction of instances from `InstanceSet1` that were solved by search-based heuristic motion planning algorithms by number of tiles and board size.



(a) Distribution of solution length by number of tiles as box plot.



(b) Distribution of solution length by board size as box plot.

Figure 4.8: Comparison of the solution length of different search-based heuristic motion planning algorithms on `InstanceSet1` by tiles and board size. Only instances for which a solution was found are taken into account. Outliers are omitted for better readability.

Effects of the alternative stop condition

The alternative stop condition that was used only in the absence of extra tiles causes the near-optimal solvers to find solutions that are within the largest possible distance on the board of the optimal solution. In exchange, sometimes a solution can be found after fewer expanded nodes. Figure 4.9 shows the distributions of steps required to move the assembled final polyomino to the target shape. Note that the BFS algorithm on average assembles the polyomino much further away from the target shape position. The heuristics, which are based on the distance of tiles to the target shape, cause configurations with tiles closer to the target shape to be expanded first. Therefore, the target polyomino is more likely to be assembled close to its target position. In contrast, for the BFS solver, it does not matter where on the board the target polyomino is assembled as long as the target shape position is reachable. This suggests, that for the heuristic-based solvers the impact of the alternative stop condition on the solution length was negligible.

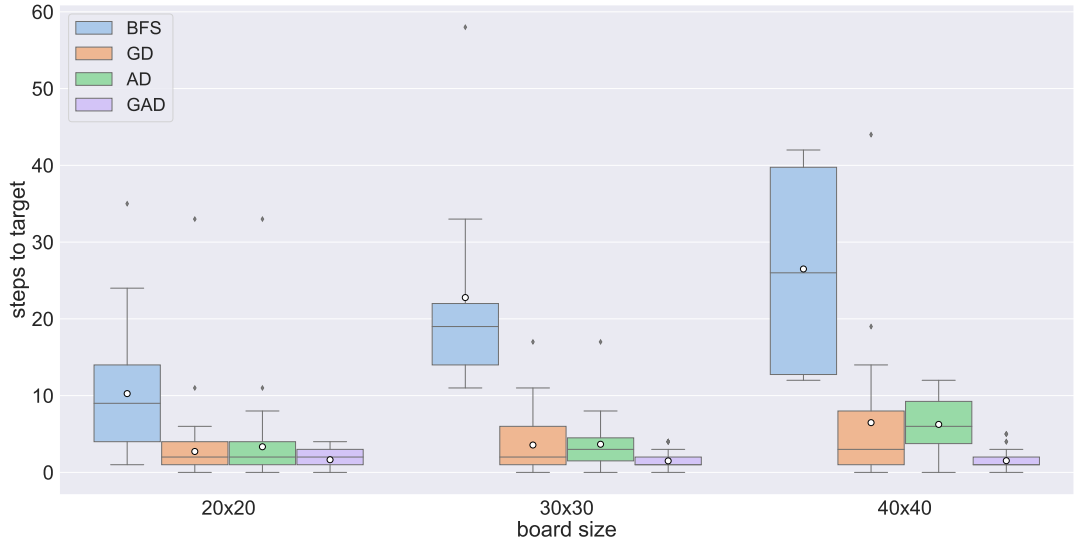
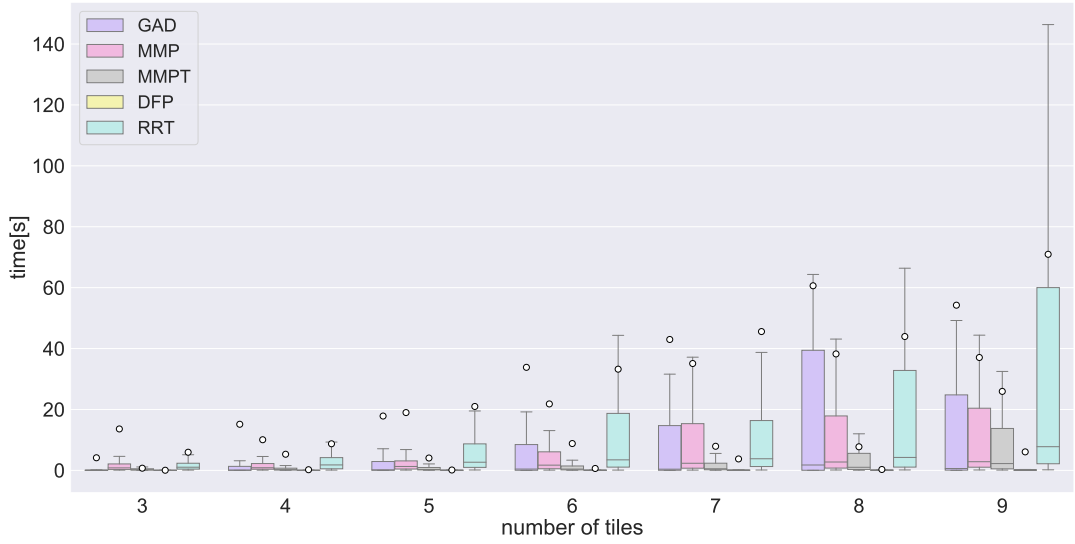


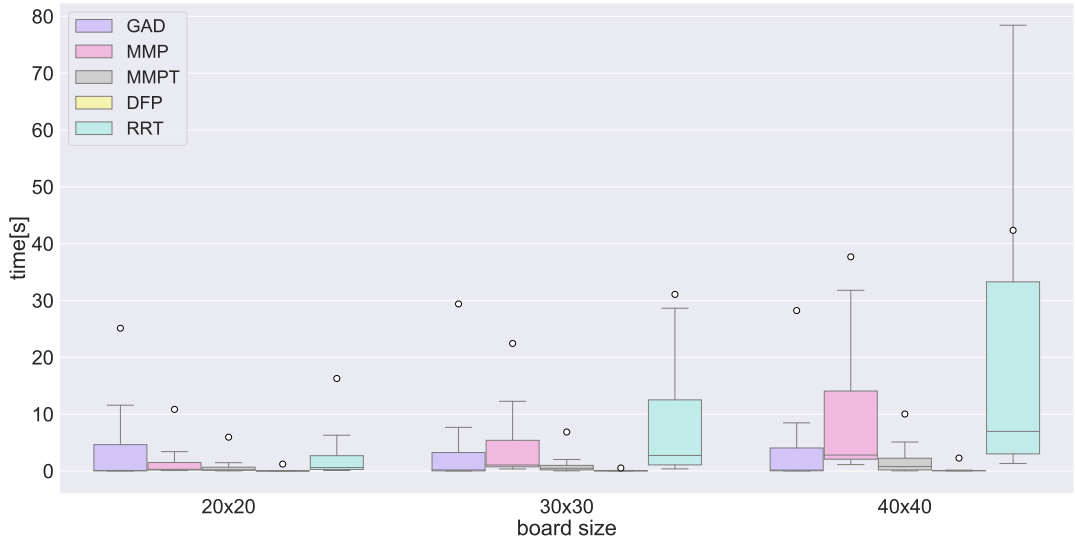
Figure 4.9: Boxplot comparison of the number of steps needed to move the assembled target polyomino to the target position for different heuristics, grouped by the size of the board. Only the instances where the alternative stop condition was used are shown.

Performance of the remaining motion planning algorithms

In this subsection, the performance of the remaining approaches, consisting of the one-tile-at-a-time motion planning algorithms (MMP, MMPT, DFP), as well as RRT, will be investigated. For comparison, GAD is included in the plots. As shown in Figures 4.10, 4.11 all one-tile-at-a-time algorithms display a better success rate than GAD on instances with a number of tiles greater than 5. The runtime on instances with many tiles is clearly better too. However, the runtime of MMP appears to be particularly sensitive to an increase in board size. MMPT does not have this downside and shows superior performance to MMP. In general, the performance of the one-tile-at-a-time approach seems to degrade much slower than the performance of other approaches when the number of tiles is increased.
The DFP motion planner can only be evaluated on instances of the Fixed Seed Tile Polyomino Assembly Problem. It has a high success rate of around 90% on these instances regardless of the number of tiles and the size of the board. Furthermore, a large majority of instances that are solved by DFP are solved in less than 1 second.
RRT displays a similar behavior as GAD but its runtime is more sensitive to increased board size.
Regarding the solution lengths, shown in Figure 4.12, the one-tile-at-a-time algorithms usually produce shorter solutions than other approaches. The length of solutions found by RRT are on average shorter and exhibit lower variance than those found by GAD. None of the solvers demonstrates a large effect of the number of tiles on the solution length. In contrast, an increased board size correlates with an increased solution length for all solvers.

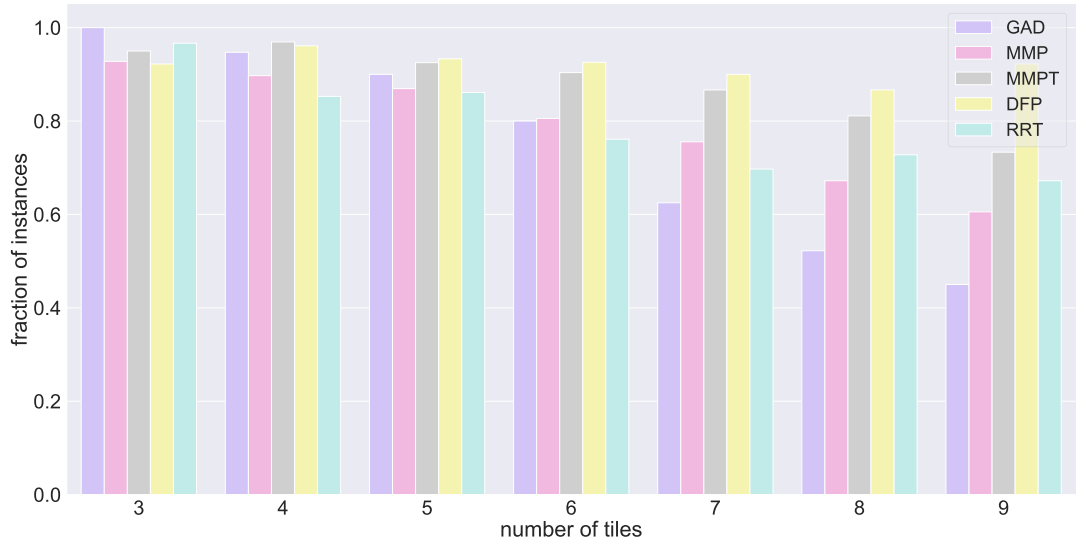


(a) Runtime distributions by number of tiles as box plot. Only successful solutions are shown.

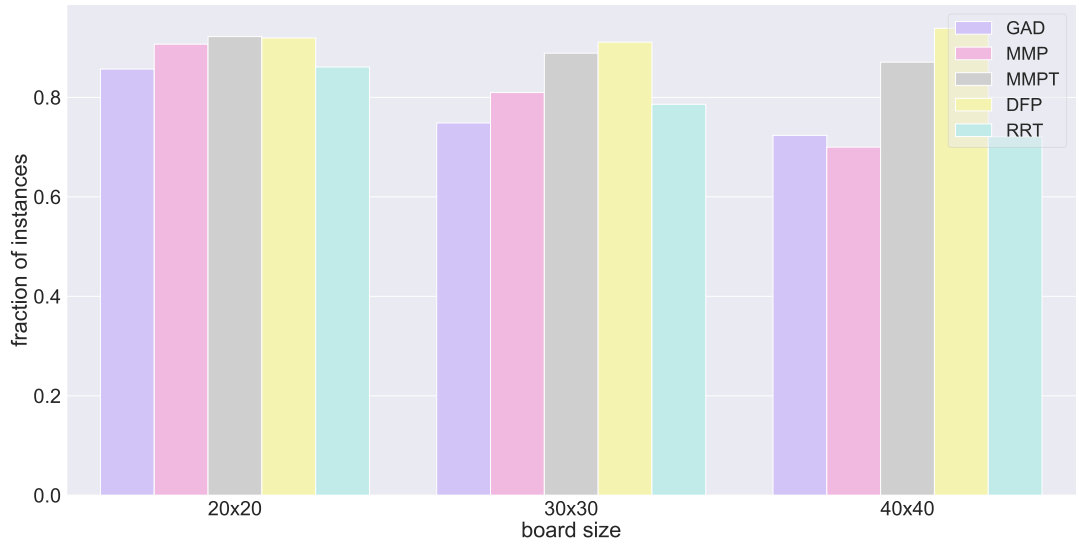


(b) Runtime distributions by board size as box plot. Only successful solutions are shown.

Figure 4.10: Comparison of the runtime of several motion planning algoritms on `InstanceSet1` by number of tiles and board size. Outliers are omitted.

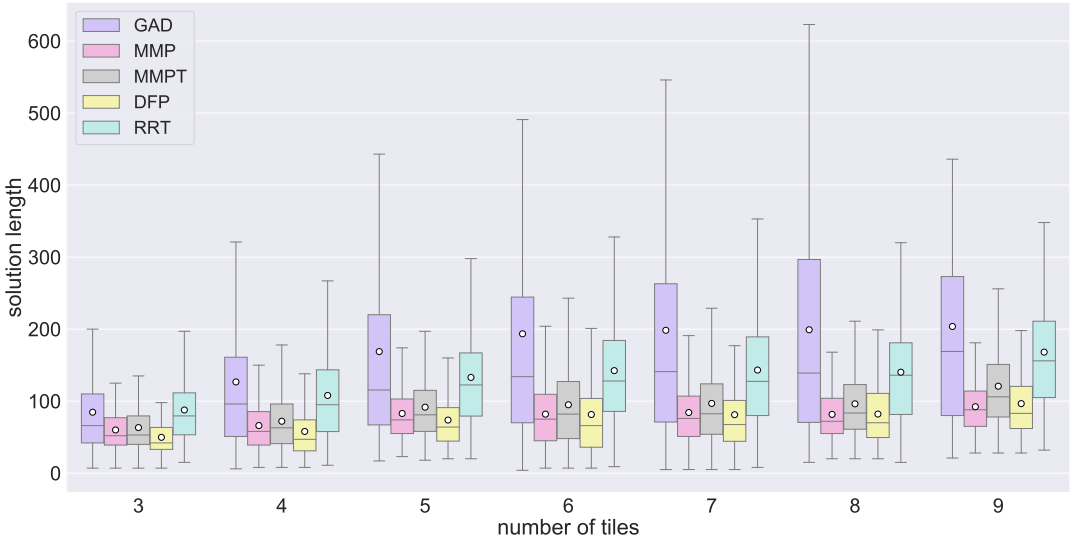


(a) Fraction of solved instances by number of tiles.

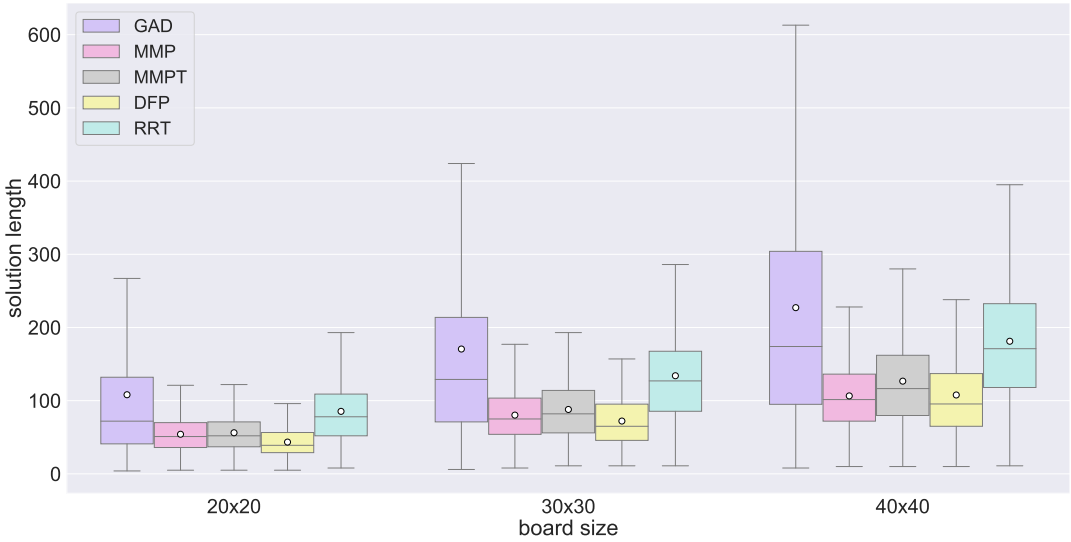


(b) Fraction of solved instances by board size.

Figure 4.11: Fraction of instances from `InstanceSet1` that were solved by different motion planning algorithms by number of tiles and board size.



(a) Distribution of solution length by number of tiles as box plot.



(b) Distribution of solution length by board size as box plot.

Figure 4.12: Comparison of the solution length of different motion planning algorithms on `InstanceSet1` by tiles and board size. Only instances for which a solution was found are taken into account.

Comparison of the difficulty of the two problem types

`InstanceSet1` contains instances of both the Polyomino Assembly Problem and the Fixed Seed Tile Polyomino Assembly Problem. We compare the performance of three fundamentally different motion planning approaches for both problems in order to evaluate which problem is harder to solve in practice. Figures 4.13, 4.14, 4.15 show the runtime performance of GAD, MMPT and RRT respectively. All three solvers demonstrate decisively better performance for the problem with a fixed seed tile. For less than 10 tiles, the time needed to solve an instance of the Fixed Seed Tile Polyomino Assembly Problem does not significantly increase when the number of tiles is increased. This is in stark contrast to instances of the Polyomino Assembly Problem where the runtime increases sharply. Moreover, DFP provides a very efficient and reliable way to solve instances with fixed seed tile.

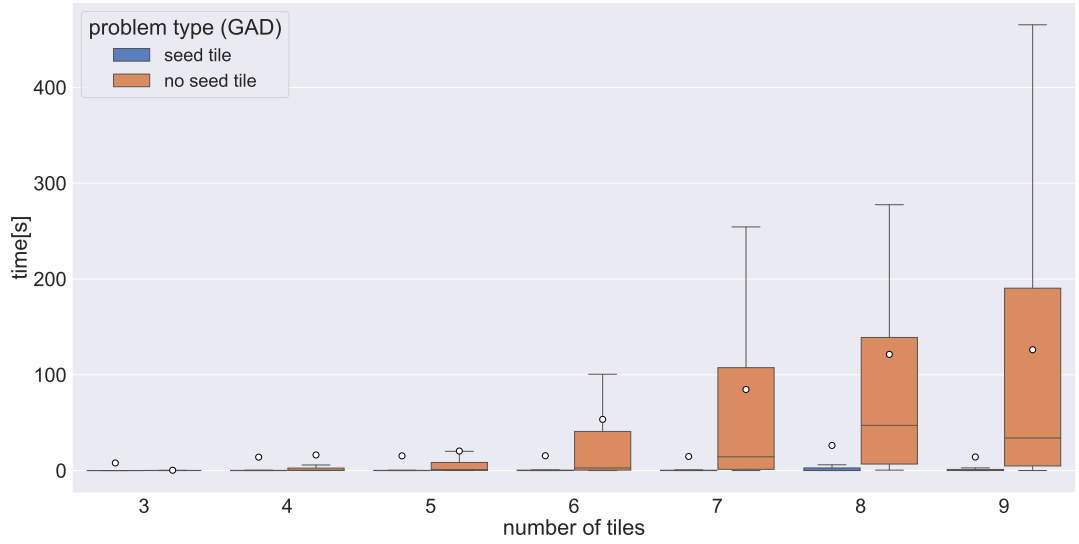


Figure 4.13: Comparison of the runtime of the Polyomino Assembly Problem with and without fixed seed tile for the GAD solver by number of tiles.

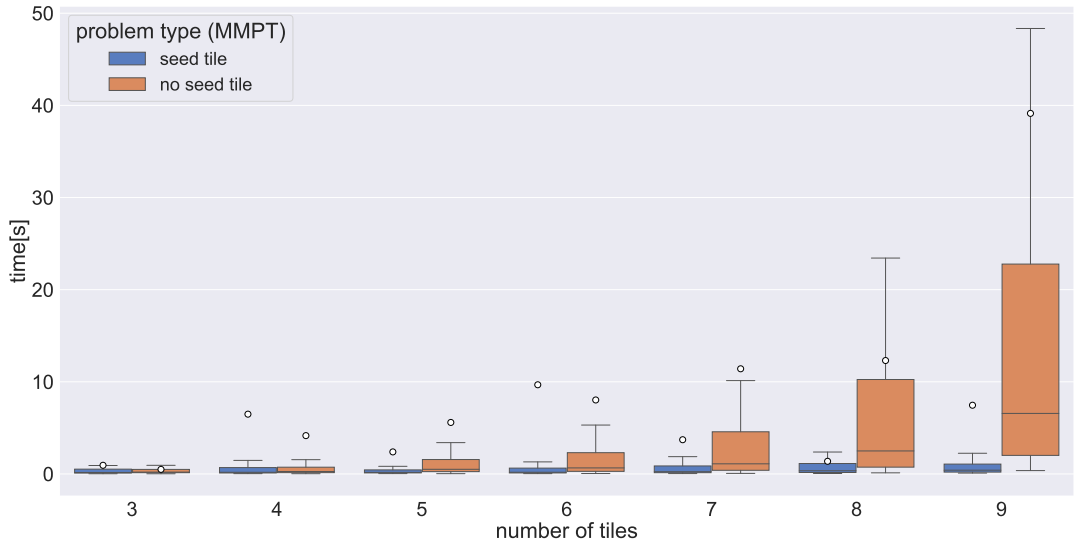


Figure 4.14: Comparison of the runtime of the Polyomino Assembly Problem with and without fixed seed tile for the MMPT solver by number of tiles.

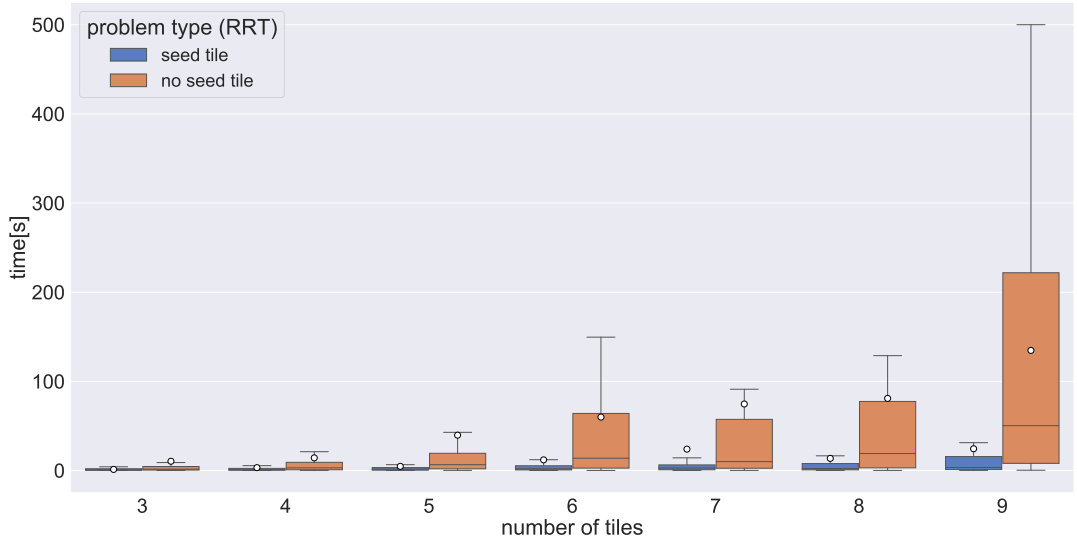
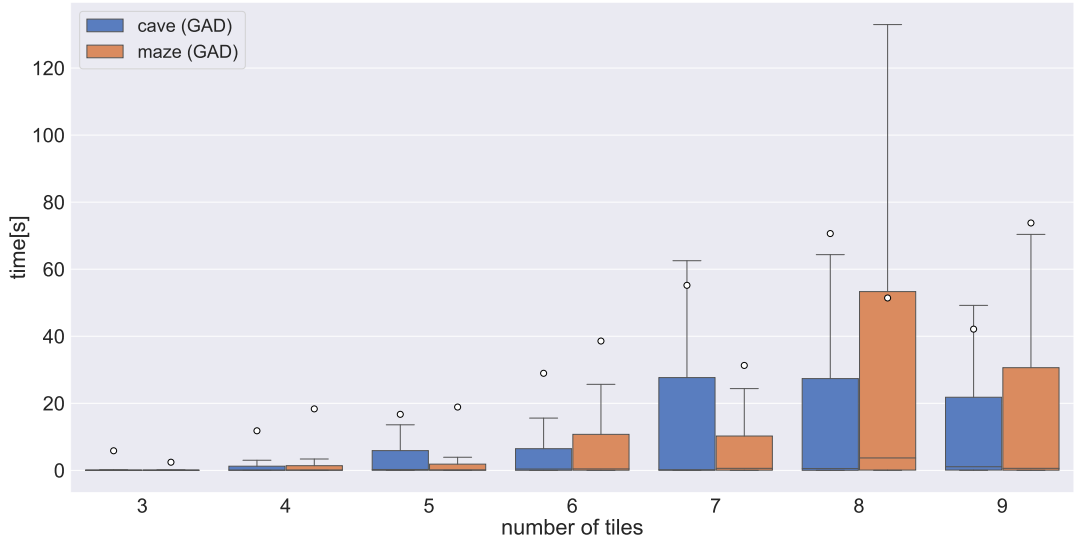


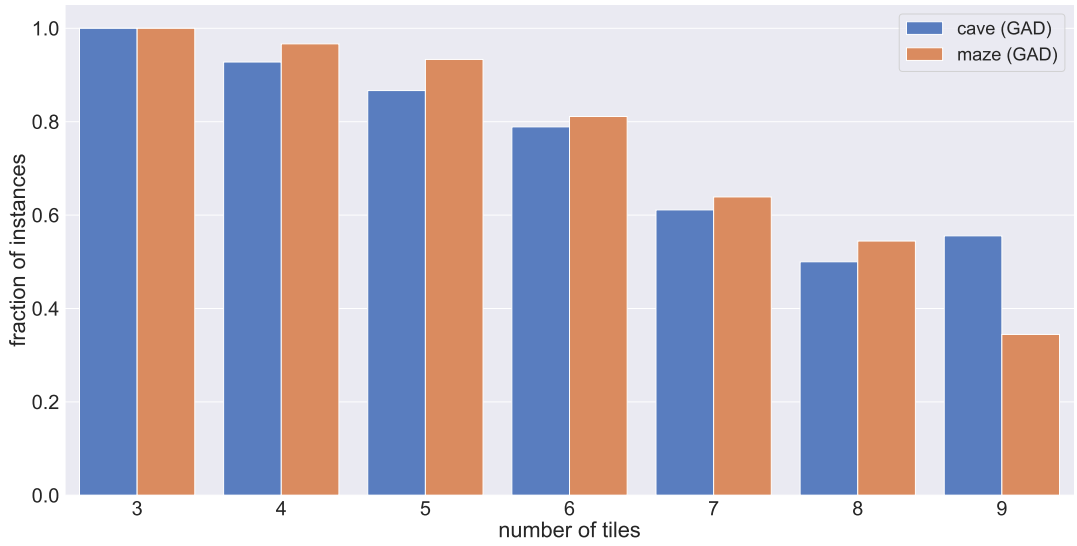
Figure 4.15: Comparison of the runtime of the Polyomino Assembly Problem with and without fixed seed tile for the RRT solver by number of tiles.

Effect of the board type on the problem difficulty

To investigate the effect of the obstacle placement, we compare the performance of motion planning algorithms on maze and cave boards (see Figures 4.16, 4.17, 4.18). GAD and RRT both display a similar success rate for both maze and cave boards, whereas MMPT solved slightly more cave boards than maze boards successfully. However, MMPT needed significantly more time for successful solves of cave boards compared to maze boards. This could indicate that, due to the different topography, undesired subassemblies are created more frequently on cave boards than on maze boards, which would cause the MMPT solver to spend more time trying to avoid the creation of subassemblies. Since GAD and RRT allow the creation of multiple subassemblies, they are less affected by this issue.
Across all solvers, the solution length on maze boards was higher than on cave boards of the same size, as shown in Figure 4.19. The reason for this could be the greater average pairwise distance between open spaces on maze boards (see Figure 4.20).

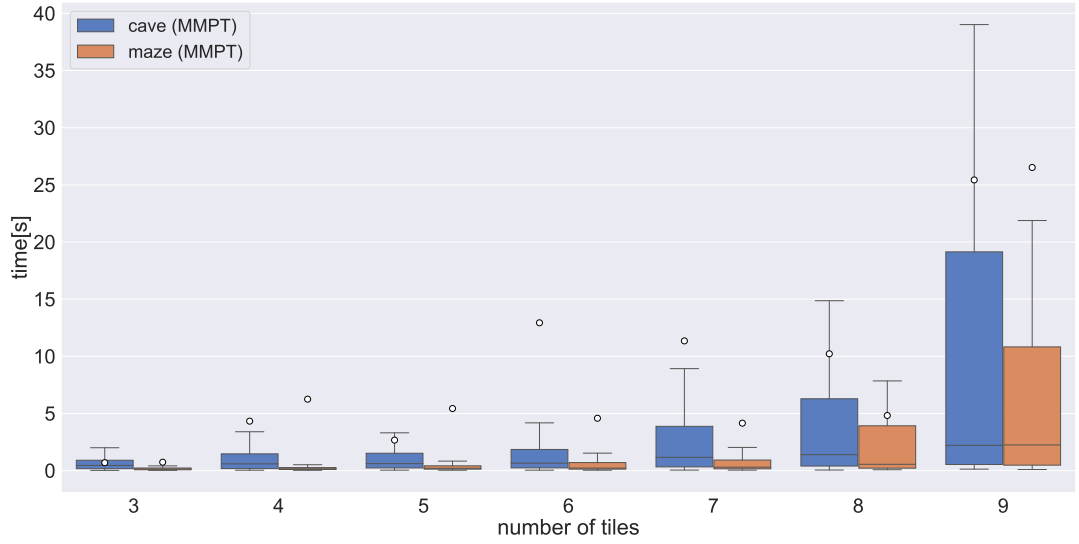


(a) Runtime distributions by number of tiles as box plot. Only successful solutions are shown.

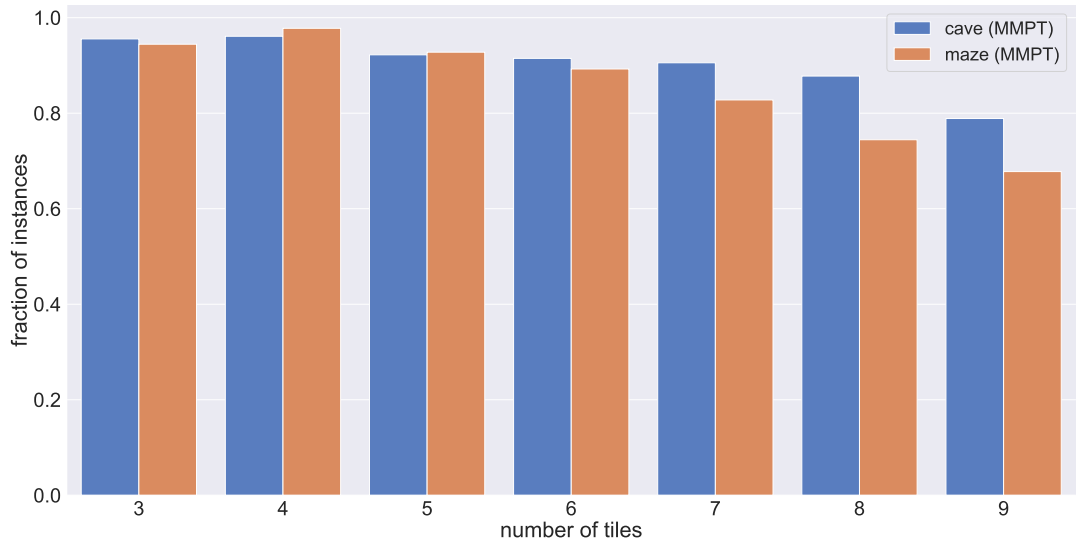


(b) Fraction of solved instances by number of tiles.

Figure 4.16: Comparison of the runtimes and success rates of the GAD motion planner on maze and cave boards by number of tiles.

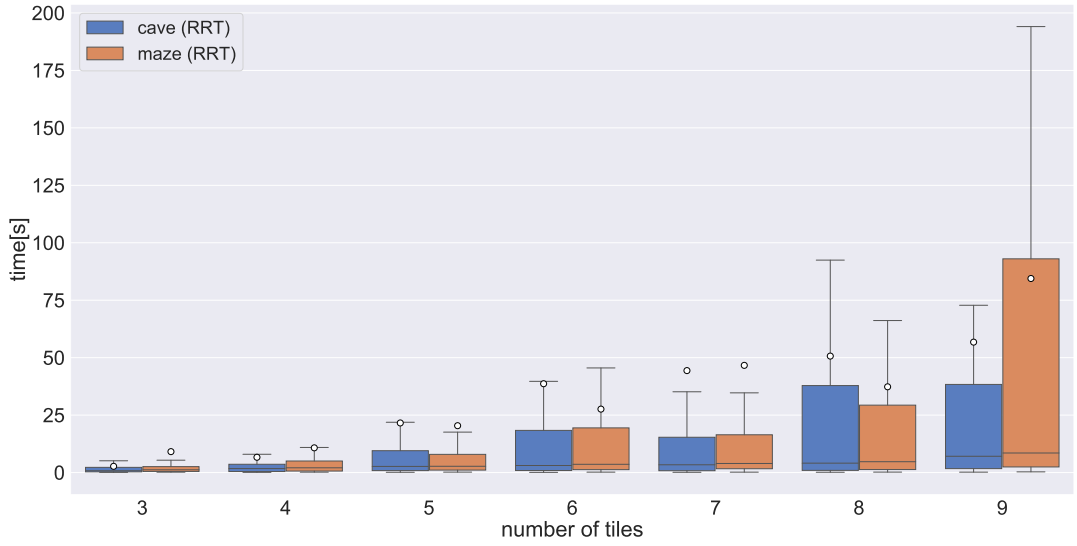


(a) Runtime distributions by number of tiles as box plot.

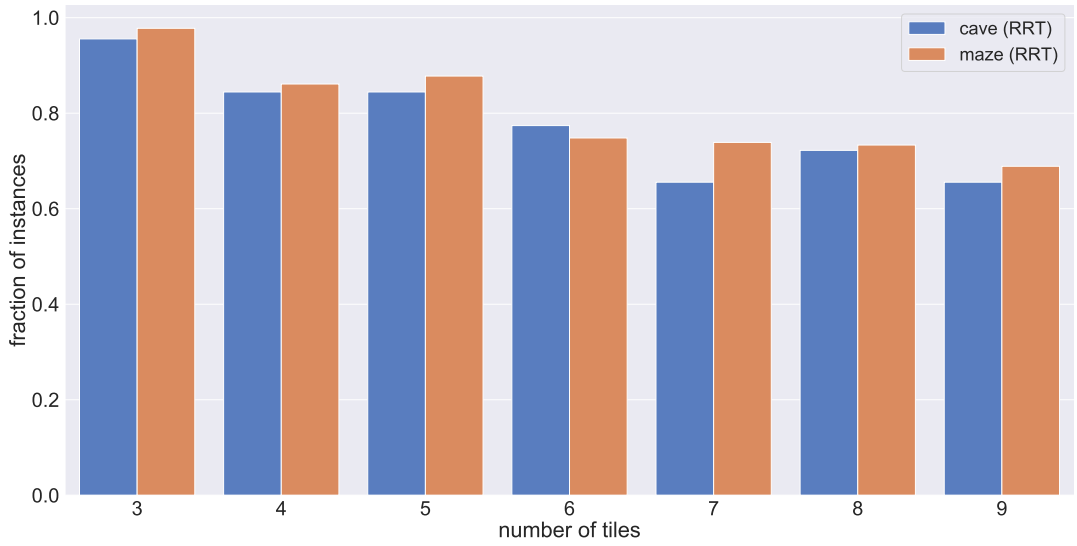


(b) Fraction of solved instances by number of tiles.

Figure 4.17: Comparison of the runtimes and success rates of the MMPT motion planner on maze and cave boards by number of tiles.



(a) Runtime distributions by number of tiles as box plot.



(b) Fraction of solved instances by number of tiles.

Figure 4.18: Comparison of the runtimes and success rates of the RRT motion planner on maze and cave boards by number of tiles.

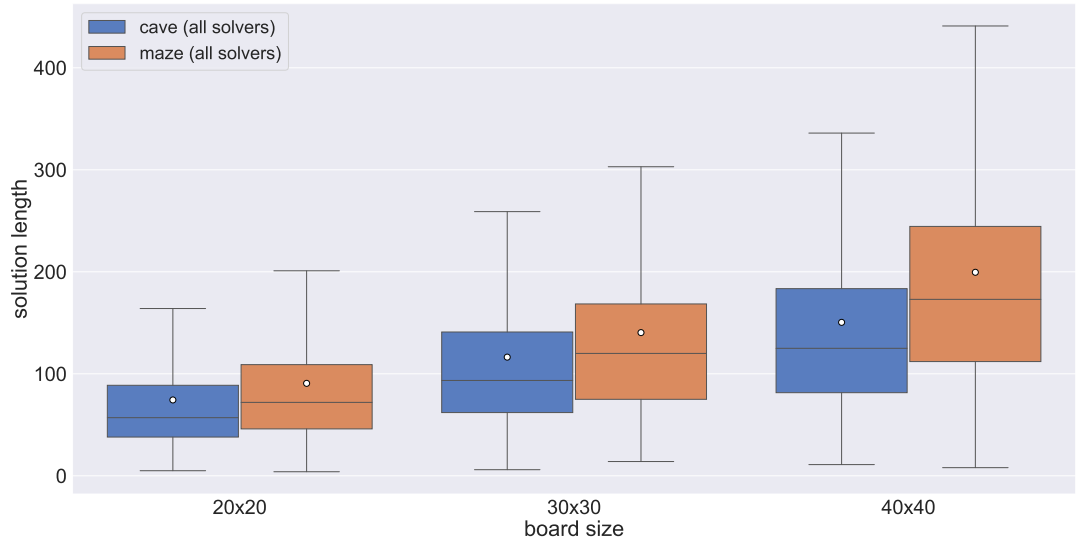


Figure 4.19: Length of found solutions compared for maze and cave boards across all solvers by board size.

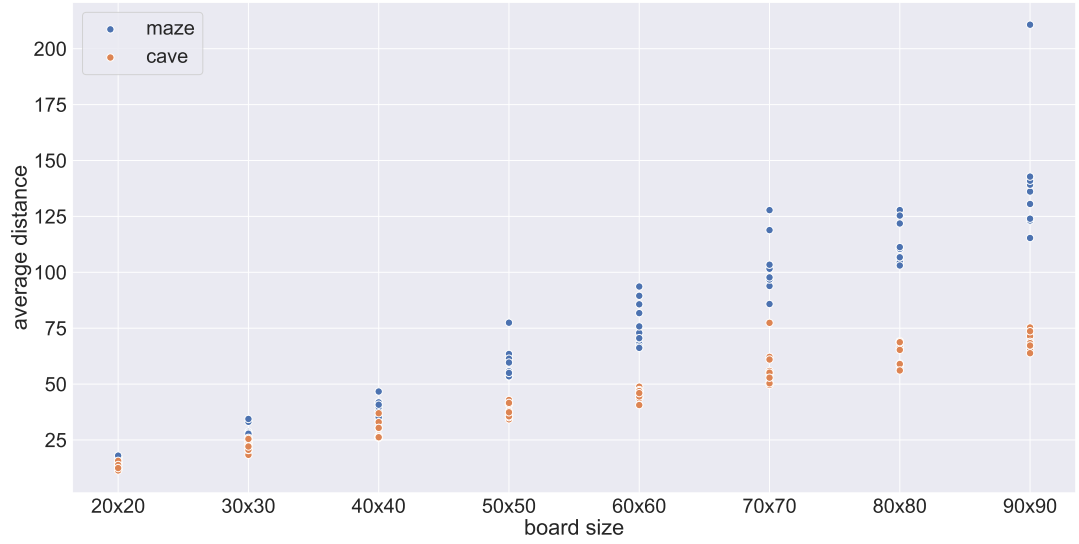
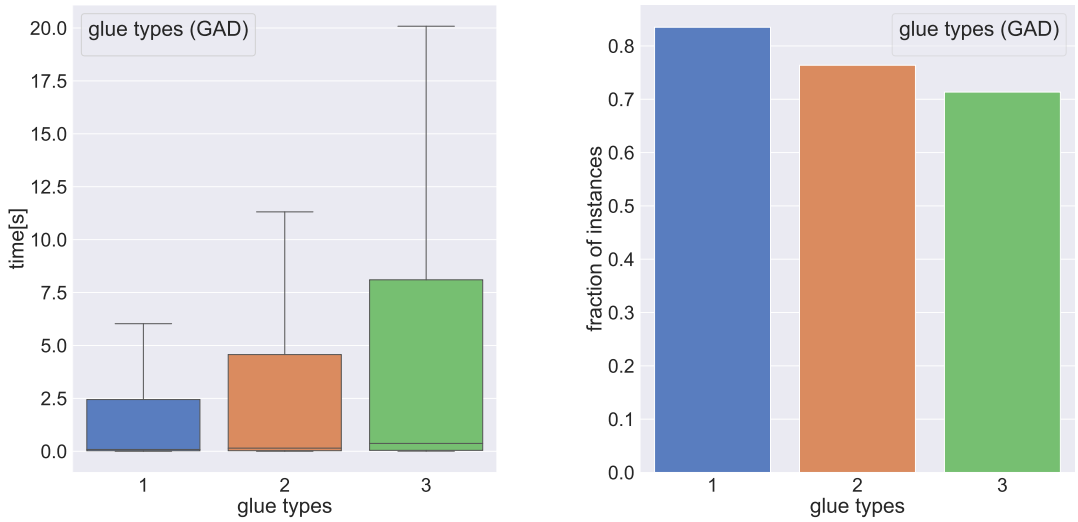


Figure 4.20: The arithmetic mean of the pairwise distances between two open positions on procedurally generated maze and cave boards of different sizes.

Effect of the number of unique glues on the problem difficulty

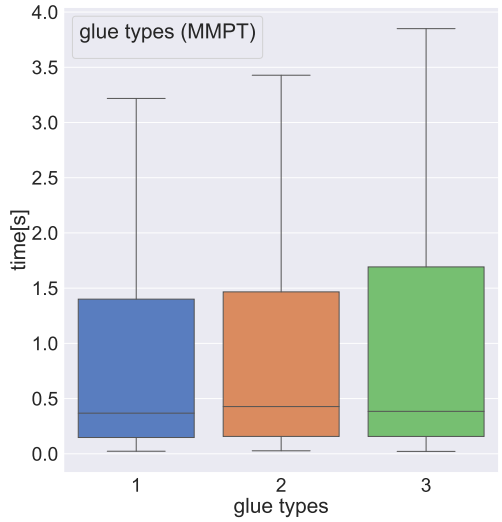
Figures 4.21, 4.22, 4.23 show the influence of different numbers of glue types on the distribution of runtimes and the success rate of GAD, MMPT and RRT. For all solvers, the runtime increases with an increase in the number of glue types. However, MMPT seems to be affected less by an increase in the number of tiles. Moreover, the success rate of MMPT does not decrease with an increased number of glues types. A reason for this is that once the building order of the polyomino is computed, which can be done quickly for the small sizes of target shapes in `InstanceSet1`, the performance of MMPT is no longer negatively influenced by a high number of glue types. On the contrary, a greater number of glue types causes tiles to create fewer undesired subassemblies.



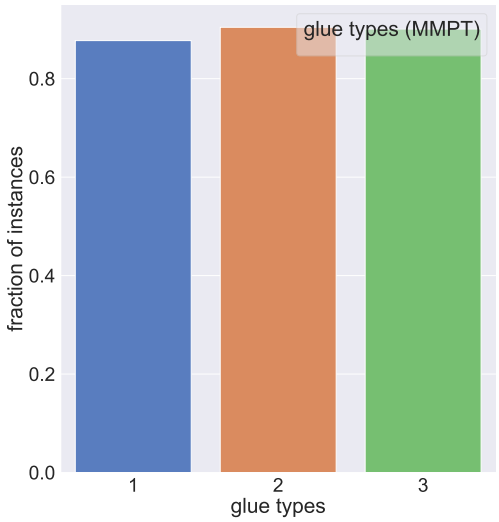
(a) Runtime distributions by number of tiles as box plot.

(b) Fraction of solved instances by number of tiles.

Figure 4.21: Comparison of the runtime and success rate of the GAD motion planning algorithm by the number of unique glue types.

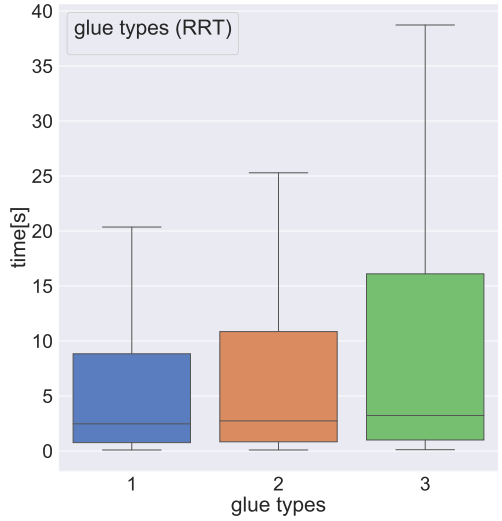


(a) Runtime distributions by number of tiles as box plot.

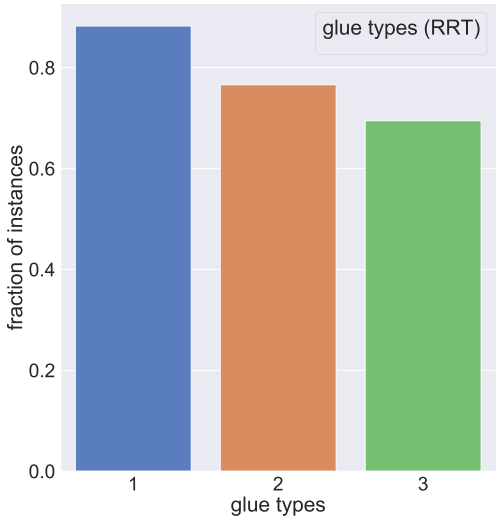


(b) Fraction of solved instances by number of tiles.

Figure 4.22: Comparison of the runtime and success rate of the MMPT motion planning algorithm by the number of unique glue types.



(a) Runtime distributions by number of tiles as box plot.



(b) Fraction of solved instances by number of tiles.

Figure 4.23: Comparison of the runtime and success rate of the RRT motion planning algorithm by the number of unique glue types.

Memory usage

Aside from the runtime and solution length, it is also interesting to compare the peak memory requirements of the different motion planning approaches. The peak memory requirement of three motion planning algorithms depending on the time needed to solve an instance are shown in Figures 4.24, 4.25, 4.26. For GAD and MMPT, the maximum of the peak memory usages implies a linear growth over time, because the speed at which nodes are expanded remains constant over time and the memory requirements per node remain constant. The memory usages of both algorithms cover a similar range with a maximum of around 8GB and 9GB respectively. In contrast, the peak memory usage of RRT is two orders of magnitude smaller, because only a sparse tree of configurations is stored. Furthermore, it does not increase linearly over time. Instead, it increases sharply in the first few seconds and then slows down significantly. An explanation for this behavior is that every iteration requires the computation of the Hausdorf distance from a random configuration to all configurations in the RRT. As the number of nodes grows over time, each iteration takes more time than the previous one. Furthermore, up to a certain number of iterations the memory requirements for a single expansion step, which includes a heuristic search, outweigh the memory requirements of the RRT.

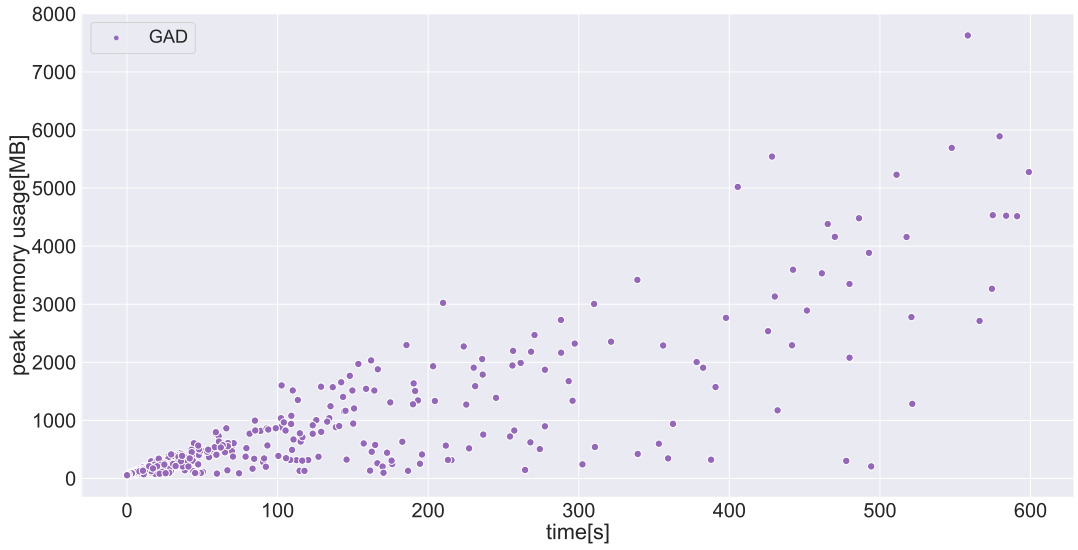


Figure 4.24: Peak memory usage of GAD in MB depending on the runtime of the solver as scatter plot.

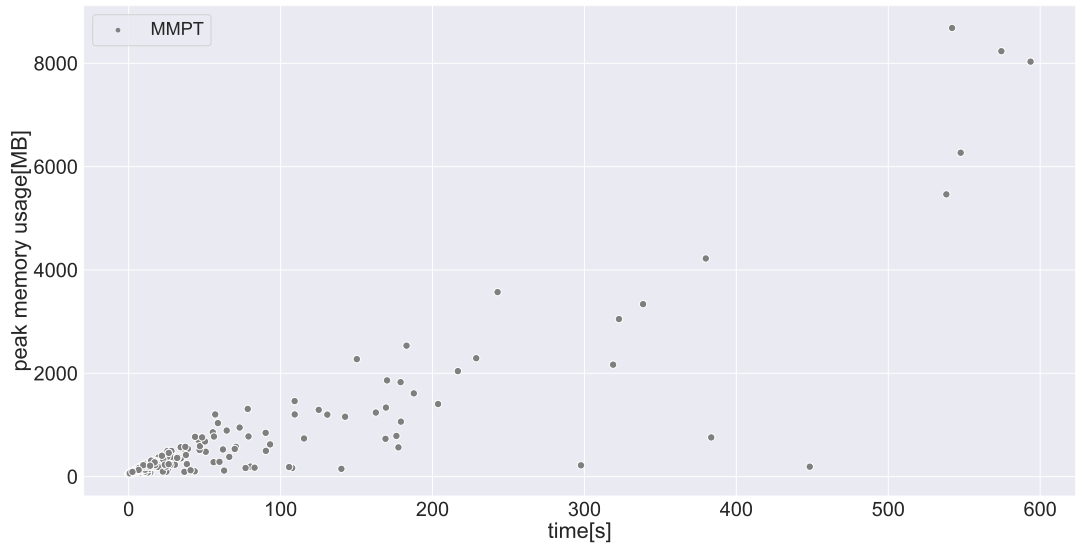


Figure 4.25: Peak memory usage of MMPT in MB depending on the runtime of the solver as scatter plot.

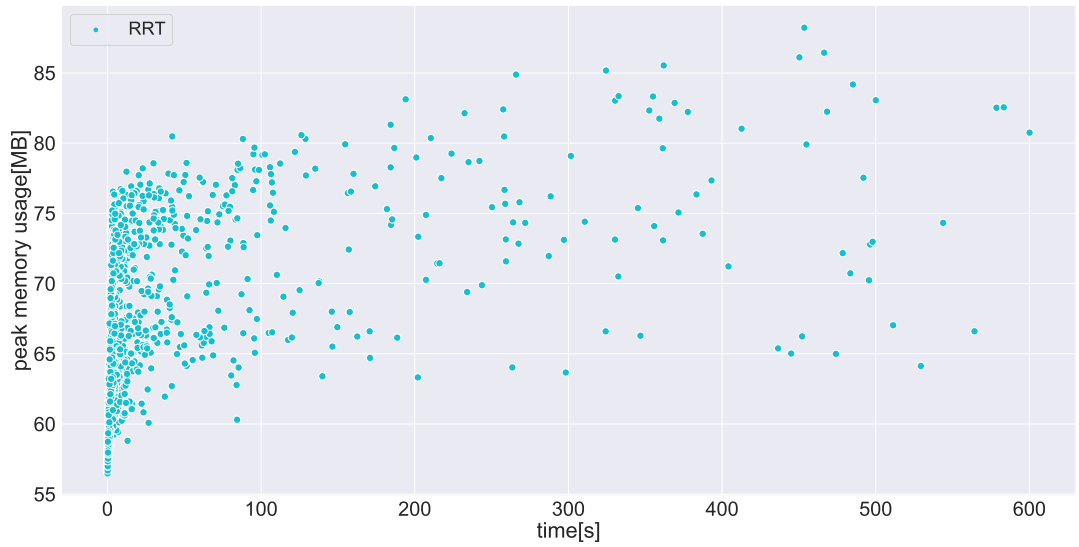


Figure 4.26: Peak memory usage of RRT in MB depending on the runtime of the solver as scatter plot.

Effectiveness of solution shortening methods

In this subsection, we evaluate the effectiveness of the solution shortening method discussed in Subsection 3.4.
We found this method to be most effective for solutions computed with the RRT algorithm (see Figure 4.27). Because of the randomized nature of the RRT, it often contains redundant steps that can be removed. The average percentage by which a solution to an instance from `InstanceSet1` could be shortened in this way is 4.83%. This includes only the instances without a fixed seed tile, for which we implemented the method.
Sometimes a significant portion of a sequence could be removed. For example, one sequence computed by GGD was shortened from 62 to 36 steps.

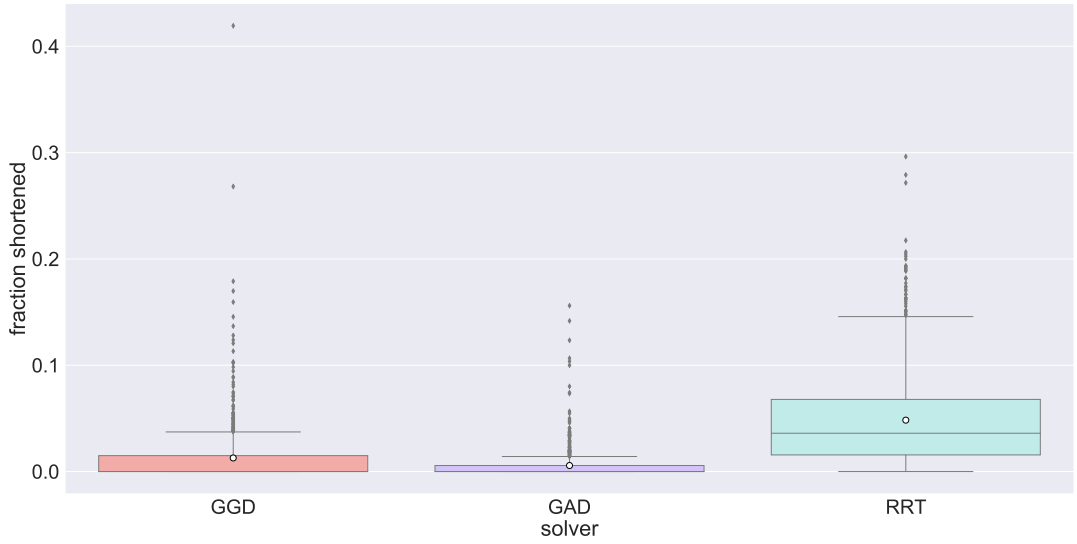


Figure 4.27: Length of shortened solutions devided by the length of the original solution for different solvers as box plot. The data consists of the results for instances without fixed seed tiles from `InstanceSet1`.

4.3.2 Evaluation of `InstanceSet2`

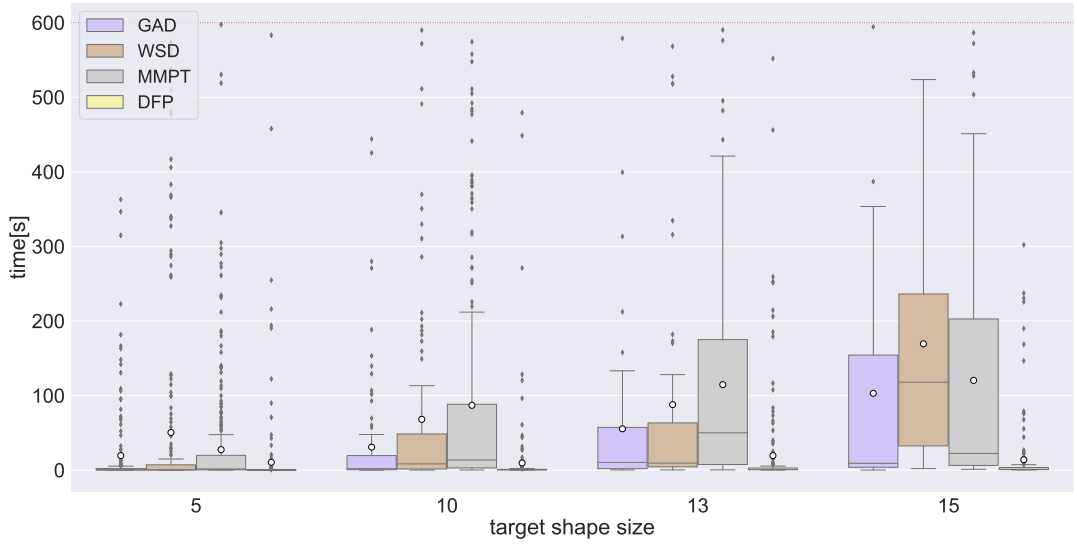
In this subsection, we evaluate selected motion planning algorithms that demonstrated a good performance in the previous experiments, on instances with a wider range of parameters. The results from `InstanceSet1` show that the near-optimal motion planning algorithms are usually unable to solve larger instances. With regards to the RRT solver, the calculation of the cost-to-go function based on the length of a shortest path is computationally infeasible for the larger boards in `InstanceSet2`. Therefore, we choose to compare GAD, WSD, MMPT, and DFP.

Comparison of the motion planning algorithms

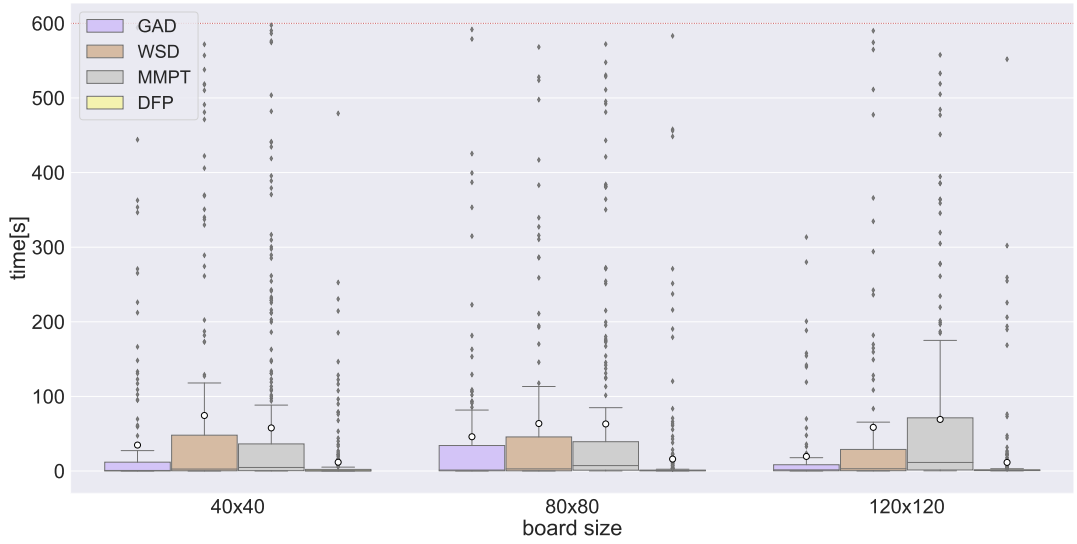
The time needed by the motion planning algorithms and their success rate can be seen in Figures 4.28, 4.29. This time the plots are grouped by the target shape size, instead of the total number of tiles on the board. Instances can have up to 5 additional tiles that are not needed to build the target shape. Generally speaking, the runtime of all algorithms increases with an increased size of the target shape. Again, all algorithms finish a majority of the solved instances long before the timeout, which indicates that there are certain instances that the solvers struggle with, whereas other instances can be solved easily. The performance of GAD and WSD in terms of the fraction of solved instances continues to decrease sharply with an increase in the size of the target shape. MMPT shows an overall higher success rate but the performance also falls off when the target shape gets is bigger. DFP performs best and shows a high success rate that only declines slowly with an increase in the target shape size and does not fall under 60%. Even when the success rate of all solvers is compared exclusively on instances with a fixed seed tile, DFP still has a far higher success rate than the other algorithms, as shown in Figure 4.31. Furthermore, DFP finds a solution much faster than all other algorithms. Since instances are not guaranteed to have a solution, it is possible that randomly generated instances with a larger target shape are less likely to be solvable, which could also be a factor for the fraction of solved instances. The greater range of board sizes in `InstanceSet2` confirms that the runtime increases much faster with an increased number of tiles, rather than increased board size. In particular, the success rate of DFP is not decreased when the board size is increased from 40×40 to 120×120.

Although the one-tile-at-a-time algorithms solve a larger fraction of instances within the timeout, GAD and WSD can sometimes find solutions faster than MMPT or find solutions to instances that MMPT is not able to solve because MMPT is incomplete. Particularly instances with simple glues and without extra tiles that the one-tile-at-a-time approach could not solve were sometimes efficiently solved by the all-tiles-at-the-same-time approach, as shown in Figure 4.32.

The solution lengths shown in Figure 4.30 grow with both the size of the board and target shape. The length of solutions found by GAD and WSD seems to be particularly sensitive to an increased target shape size. In comparison, WSD produced slightly shorter sequences than GAD. Overall the one-tile-at-a-time algorithms find shorter solutions than the all-tiles-at-the-same-time algorithms.
Note that due to the method through which the instances were created it was relatively easy for the one-tile-at-a-time algorithms to find a valid set of tiles and an order in which a given polyomino can be built because a majority of glue types was guaranteed to stick together. However, for larger target shapes or sparser glue functions, this becomes a challenging task that could limit the efficiency of the one-tile-at-a-time approach.

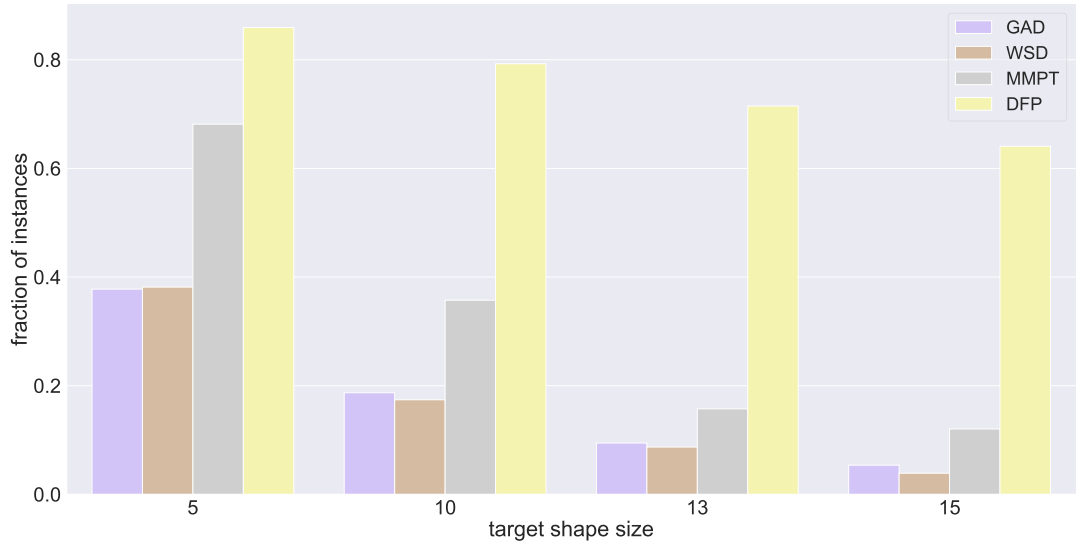


(a) Runtime distributions by target shape size as box plot. Only successful solutions are shown.

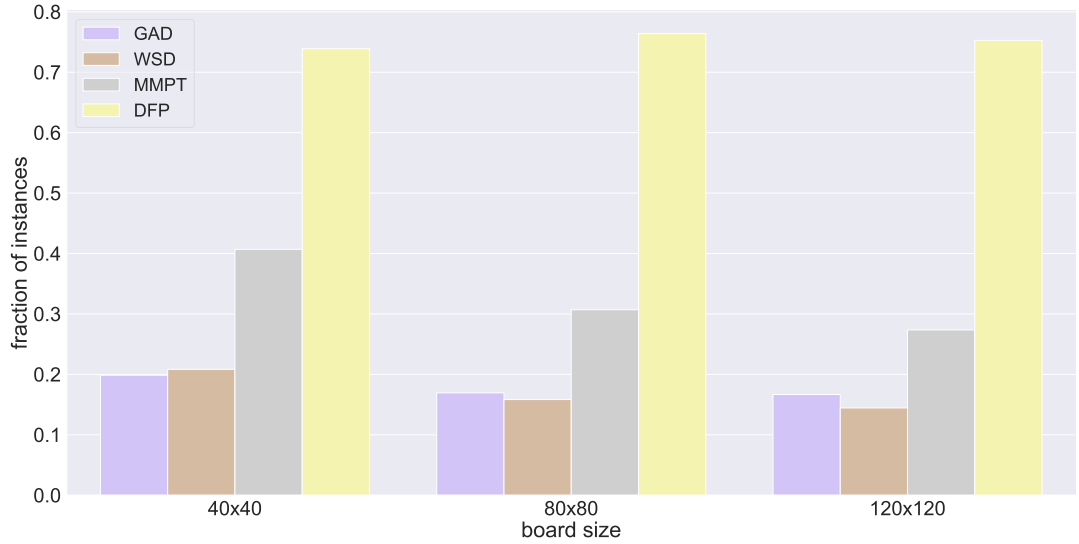


(b) Runtime distributions by board size as box plot. Only successful solutions are shown.

Figure 4.28: Comparison of the runtime of four motion planning algoritms on `InstanceSet2` by target shape size and board size.

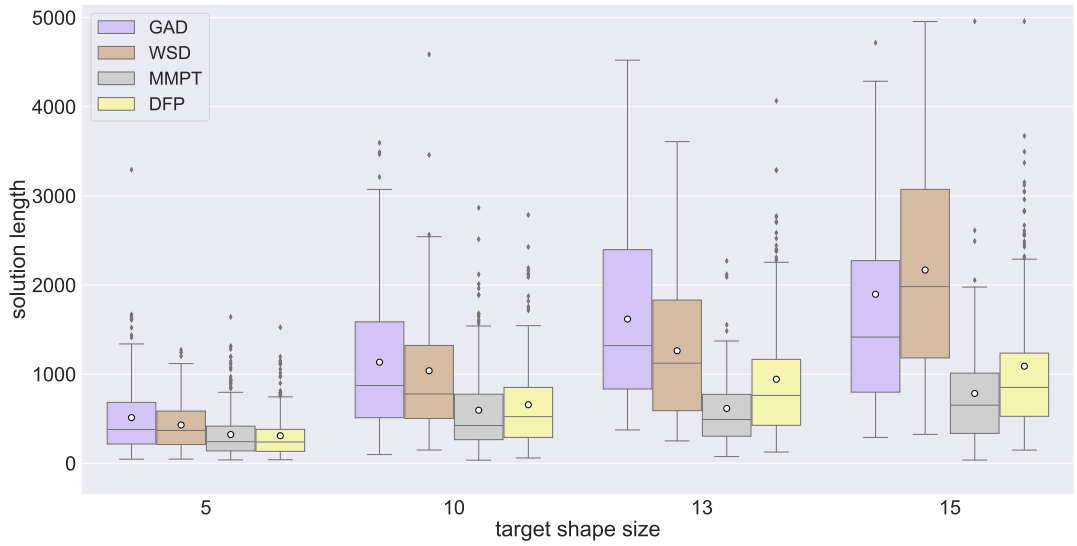


(a) Fraction of solved instances by target shape size.

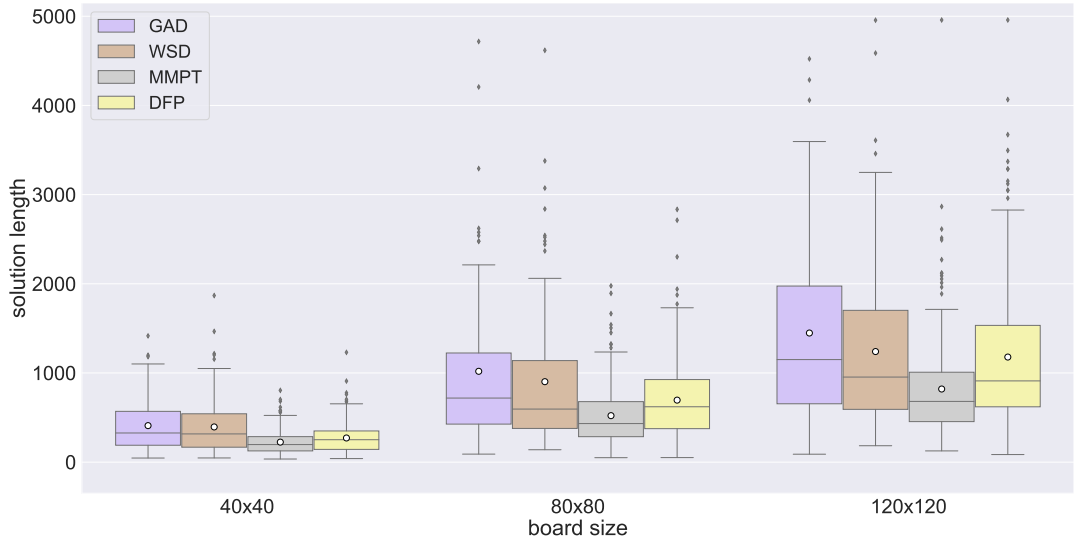


(b) Fraction of solved instances by board size.

Figure 4.29: Fraction of instances from `InstanceSet2` that where solved by four motion planning algorithms by target shape size and board size. For DFP the fraction of suitable instances with a fixed tile that were solved is shown.



(a) Distribution of solution length by target shape size as box plot.



(b) Distribution of solution length by board size as box plot.

Figure 4.30: Comparison of the solution length of four motion planning algorithms on `InstanceSet2` by target shape size and board size. Only instances for which a solution was found are taken into account.

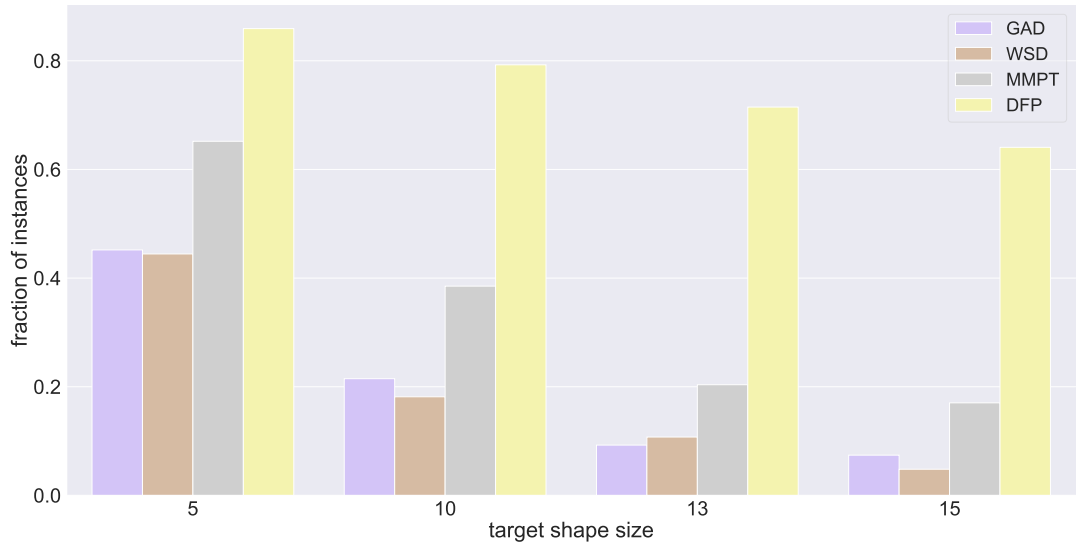
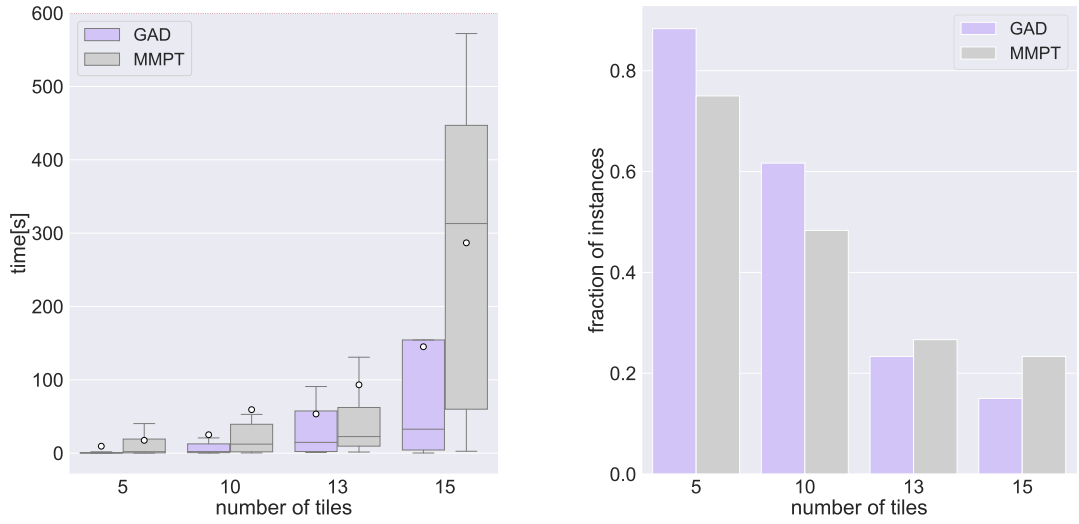


Figure 4.31: Fraction of instances **with fixed seed tiles** from `InstanceSet2` that were solved by four motion planning algorithms by target shape size and board size.



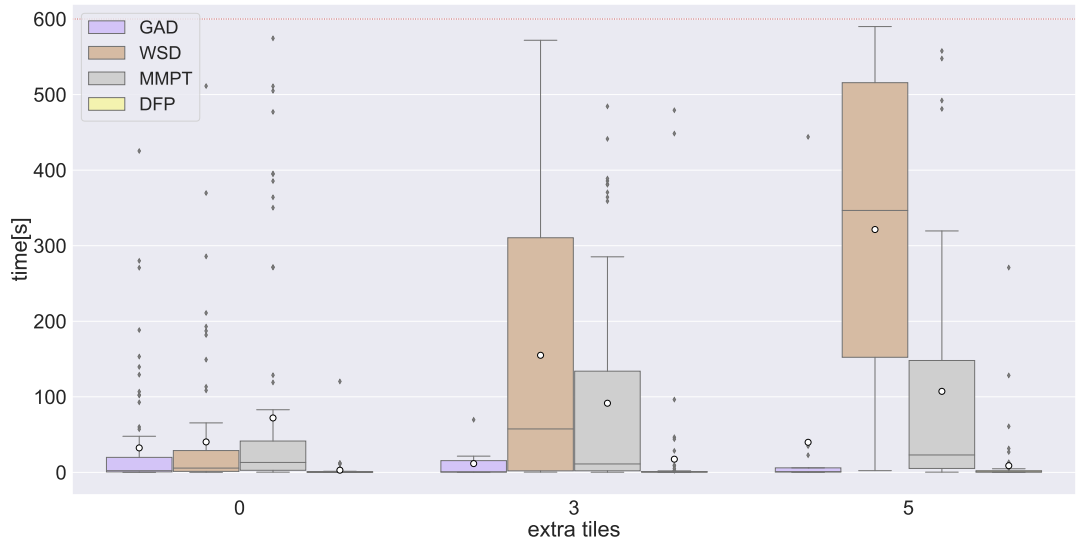
(a) Runtime distributions by number of tiles as box plot.

(b) Fraction of solved instances by number of tiles.

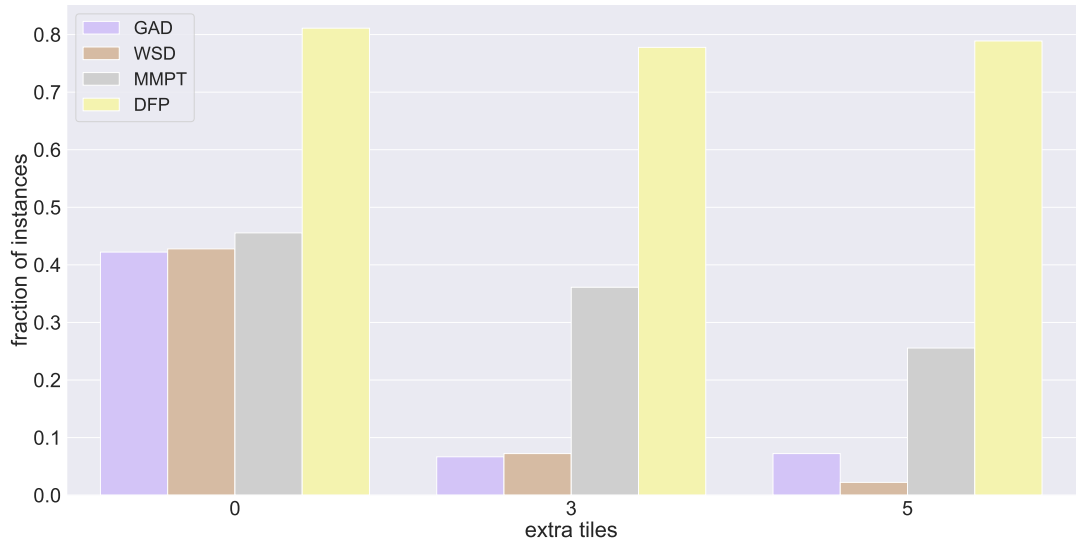
Figure 4.32: Comparison of the runtime and success rate of GAD and MMPT on instances from `InstanceSet2` with exactly one glue and no extra tiles.

Effect of extra tiles on the problem difficulty

In this subsection, the effect of extra tiles that are not needed to assemble the target polyomino on the performance of motion planning algorithms is investigated. As an example, the runtime and success rate of different solvers depending on the number of extra tiles for the instances from `InstanceSet2` that have a target shape of size 10 is shown in Figure 4.33. The negative effect of extra tiles on the success rate seems to be much stronger for GAD and WSD than for other solvers. The reason for this is that GAD and WSD only minimize the distance of the n nearest tiles to the target shape, where n is the target shape size. Hence, whenever it is impossible to build the target shape from the n tiles with the nearest starting positions, this approach is very inefficient. In contrast, DFP only shows a minor decrease in success rate when extra tiles are present. The results for the runtime of GAD and WSD are not meaningful due to the small fractions of solved instances with extra tiles. MMPT on the other hand displays an increased runtime when a small number of extra tiles are added. Presumably, the greater density of tiles on the board makes it more difficult to avoid undesired subassemblies.
Figure 4.34 shows that the solution length does not grow when a small number of extra tiles is present. The solution lengths of GAD and WSD appear to be shorter when extra tiles are added, which can be explained by the fact that on average a greater number of tiles start close to the target shape in this case.



(a) Distribution of runtime by number of extra tiles as box plot. Only successful solutions are shown.



(b) Fraction of solved instances by number of extra tiles.

Figure 4.33: Comparison of the runtimes and success rates of four motion panning algorithms by the number of extra tiles. Evaluated on the instances from `InstanceSet2` that have a target shape size of exactly 10.

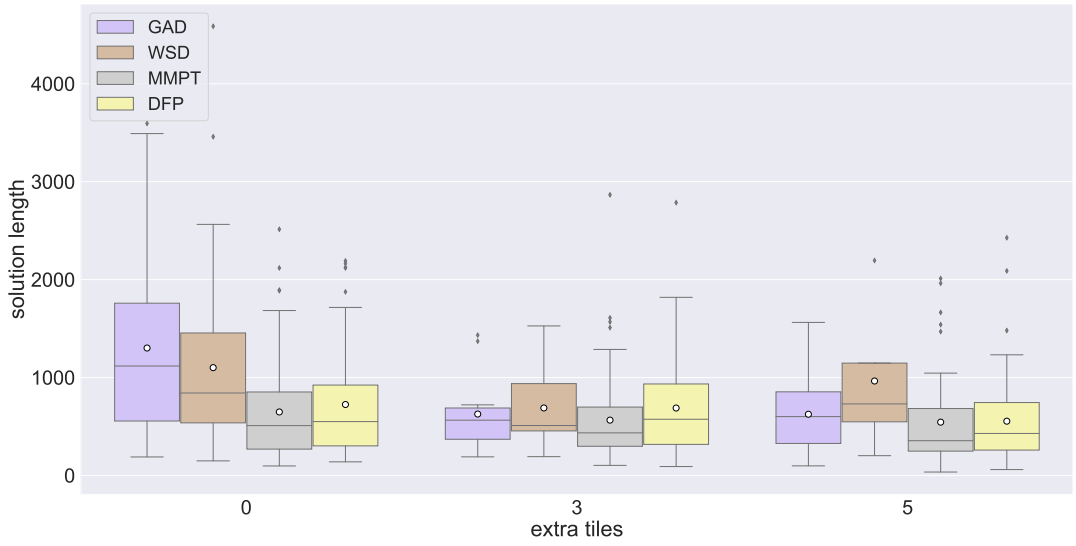


Figure 4.34: Comparison of the solution length of four motion panning algorithms by the number of extra tiles. Evaluated on the instances from `InstanceSet2` that have a target shape size of exactly 10.

5 Conclusion and Future Work

In this thesis, we designed motion planning algorithms for the assembly of shapes in the Tilt model based on multiple different approaches, including search-based and sampling-based motion planning algorithms. We evaluated the effectiveness of these approaches experimentally on procedurally created boards and investigated the parameters that are most significant for the performance of motion planning algorithms in general, as well as the specific strength and weaknesses of the different approaches. We found that the number of tiles on the board has the most significant impact on the runtime, which is an expected result since the difficulty of motion planning problems is known to largely depend on the dimensionality of the configuration space, which is proportional to the number of tiles in the case of Tilt.

Our first approach, an A* search in combination with a consistent heuristic based on the distance of tiles to the target shape, is able to find optimal solutions for small numbers of tiles, but quickly becomes computationally infeasible.

Greedy best-first search with heuristics based on the distances of tiles to the target shape is more efficient, however it often produces a solution with far from optimal length. On a positive note, the average factor by which a solution found by GAD was longer than a solution to the same instance found by AD was less than 3 for the evaluated instances and seems to decrease for larger boards. Unfortunately, the runtime of greedy best-first search increases sharply when the number of tiles is greater than 10 and is very sensitive to additional tiles that are not needed to construct the target shape, due to the heuristics indifference towards the glue types of tiles and their possible positions within the target shape. The three proposed greedy best-first search algorithms GGD, GAD, and WSD (with the exponent 2), which all use a heuristic based on the distance of tiles to the target shape, exhibit very similar performance. However, GGD seems to perform worse than the other algorithms as the number of tiles grows and WSD produced solutions of slightly shorter length. Overall, WSD seems to present the best tradeoff between solution length and runtime, which might be further improved by selecting a different exponent.

Avoiding subassemblies and adding one tile at a time to the target polyomino can often quickly yield a solution even on boards with more than 10 tiles. However, this approach is not a complete algorithm. Furthermore, it struggles with boards that are densely populated with tiles as well as boards containing few obstacles that separate tiles from each other, because in these cases it becomes hard or even impossible to avoid subassemblies. Another problem for one-tile-at-a-time motion planning algorithms is the necessity to compute a valid construction order for the target shape with regard to the glues on the edges of the available tiles. For complicated glues and large target shapes, this is a hard problem to solve. While a large number of glue types increases the difficulty

of assembling the target shape, it helps to avoid the accidental creation of undesired subassemblies and can therefore be advantageous for one-tile-at-a-time algorithms, once they found a valid building order. Whereas for the other evaluated classes of algorithms, more glue types had a negative effect on the performance.

A simple solution shortening technique can help to decrease the length of solutions found by non-optimal algorithms and is particularly effective for solutions found with the RRT algorithm.

Even though the Fixed Seed Tile Polyomino Assembly Problem is PSPACE-complete, it was much easier to solve at the same numbers of tiles than its counterpart without fixed seed tiles. DFP is a simple and fast motion planning algorithm for this problem, however, it is not complete. In the experiments, DFP solved a large fraction of instances including instances with larger numbers of tiles in a short time.

The evaluation of computational complexity for the Polyomino Assembly Problem without a fixed seed tile and without extra tiles is left to future research.

RRT is a promising method to solve Tilt motion planning problems. Combined with a computationally more expensive expansion step, it has the added benefit of requiring less memory than other approaches. A major challenge in this context is to find a cost-to-go function that is fast to compute and gives a good approximation of the actual distance between two configurations. Furthermore, the RRT algorithm we designed has a large number of adjustable parameters. Future work could find optimal settings for these parameters and evaluate the effectiveness of other cost-to-go functions.

Better best-first search algorithms could potentially be achieved with heuristics that not only depend on the distance of tiles to the target shape but instead consider, for example, the involved glue types and possible positions of tiles within the target polyomino.

Another possible direction for future work is to evaluate motion planning algorithms on configurations with parameters outside the scope of this thesis, such as boards that contain a high density of tiles or a small number of obstacles.

Bibliography

[1] J. Balanza-Martinez, D. Caballero, A. Cantu, T. Gomez, A. Luchsinger, R. Schweller, and T. Wylie. Relocation with uniform external control in limited directions. *The 22nd Japan Conference on Discrete and Computational Geometry, Graphs, and Games, JCDCGGG*, pages 39–40, 2019.

[2] J. Balanza-Martinez, D. Caballero, A. A. Cantu, M. Flores, T. Gomez, A. Luchsinger, R. Reyes, R. Schweller, and T. Wylie. Hierarchical shape construction and complexity for slidable polyominoes under uniform external forces. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, volume 2020-Janua, pages 2625–2641. Association for Computing Machinery, 2020.

[3] J. Balanza-Martinez, D. Caballero, A. A. Cantu, L. A. Garcia, A. Luchsinger, R. Reyes, R. Schweller, and T. Wylie. Full tilt: Universal constructors for general shapes with uniform external forces. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2689–2708. Association for Computing Machinery, 2019.

[4] A. Becker, E. D. Demaine, S. P. Fekete, G. Habibi, and J. McLurkin. Reconfiguring massive particle swarms with limited, global control. In *Proceedings of the International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS)*, pages 51–66, 2013.

[5] A. Becker, E. D. Demaine, S. P. Fekete, and J. McLurkin. Particle computation: Designing worlds to control robot swarms with only global signals. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 6751–6756. Institute of Electrical and Electronics Engineers Inc., 2014.

[6] A. T. Becker, S. P. Fekete, L. Huang, P. Keldenich, L. Kleist, D. Krupke, C. Rieck, and A. Schmidt. Targeted Drug Delivery: Algorithmic Methods for Collecting a Swarm of Particles with Uniform, External Forces. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2508–2514. Institute of Electrical and Electronics Engineers Inc., 2020.

[7] A. T. Becker, S. P. Fekete, P. Keldenich, D. Krupke, C. Rieck, C. Scheffer, and A. Schmidt. Tilt Assembly: Algorithms for Micro-Factories that Build Objects with Uniform External Forces. In Y. Okamoto and T. Tokuyama, editors, *28th International Symposium on Algorithms and Computation (ISAAC 2017)*, volume 92 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:13, 2017.

[8] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[9] D. Caballero, A. A. Cantu, T. Gomez, A. Luchsinger, R. Schweller, and T. Wylie. Hardness of reconfiguring robot swarms with uniform external control in limited directions. *Journal of Information Processing*, 28:782–790, 2020.

[10] D. Caballero, A. A. Cantu, T. Gomez, A. Luchsinger, R. Schweller, and T. Wylie. Relocating Units in Robot Swarms with Uniform Control Signals is PSPACE-Complete. In *Proc. 32th Canadian Conference on Computational Geometry*, pages 49–55, 2020.

[11] S. Cannon, E. D. Demaine, M. L. Demaine, S. Eisenstat, M. J. Patitz, R. Schweller, S. M. Summers, and A. Winslow. Two Hands Are Better Than One (up to constant factors). *arXiv preprint arXiv:1201.1650*, 2012.

[12] H. L. Chen and D. Doty. Parallelism and time in hierarchical self-assembly. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1163–1182. Society for Industrial and Applied Mathematics Publications, 2012.

[13] E. D. Demaine, M. L. Demaine, S. P. Fekete, M. Ishaque, E. Rafalin, R. T. Schweller, and D. L. Souvaine. Staged self-assembly: Nanomanufacture of arbitrary shapes with O(1) glues. In *Natural Computing*, volume 7, pages 347–370. Springer, 2008.

[14] European Mathematical Society. Hausdorff metric. In *Encyclopedia of Mathematics*. EMS Press, 2022.

[15] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the Complexity of Motion Planning for Multiple Independent Objects; PSPACE- Hardness of the "Warehouseman's Problem":. *The International Journal of Robotics Research*, 3(4):76–88, 1984.

[16] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[17] J. Keller, C. Rieck, C. Scheffer, and A. Schmidt. Particle-Based Assembly Using Precise Global Control. *arXiv preprint arXiv:2105.05784*, 2021.

[18] S. M. Lavalle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. Technical report, 1998.

[19] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, USA, 2006.

[20] L. Petrović. Motion planning in high-dimensional spaces. *arXiv preprint arXiv:1806.07457*, 2018.

[21] A. Schmidt, S. Manzoor, L. Huang, A. T. Becker, and S. P. Fekete. Efficient parallel Self-Assembly under uniform control inputs. *IEEE Robotics and Automation Letters*, 3(4):3521–3528, 2018.

[22] H. Wang. Proving Theorems by Pattern Recognition — II. *Bell System Technical Journal*, 40(1):1–41, 1961.

[23] E. Winfree. *Algorithmic Self-Assembly of DNA*. Phd thesis, Caltech, 1998.

[24] Y. Yang, J. Pan, and W. Wan. Survey of optimal motion planning. *IET Cyber-Systems and Robotics*, 1(1):13–19, 2019.

[25] X. Yun and K. C. Tan. Wall-following method for escaping local minima in potential field based motion planning. In *1997 8th International Conference on Advanced Robotics. Proceedings. ICAR'97*, pages 421–426. IEEE, 1997.