Toward Understanding the Hardness of Multi-Agent Path Finding

Ofir Gordon

Toward Understanding the Hardness of Multi-Agent Path Finding

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

Ofir Gordon

Submitted to the Senate
of the Technion — Israel Institute of Technology
Tishrei 5782 Haifa October 2021

This research was carried out under the supervision of Dr. Oren Salzman, in the Faculty of Computer Science.

Some results in this thesis have been published as an article by the author and research collaborators in a conference (SoCS 2021) during the course of the author's masters research period, the most up-to-date version of which being:

> Gordon, O., Filmus, Y., and Salzman, O. (2021). Revisiting the complexity analysis of conflict-based search: New computational techniques and improved bounds. In *Proceedings of the Fourteenth International Symposium on Combinatorial Search, SOCS 2021, Virtual Conference [Jinan, China], July 26-30, 2021*, pages 64–72. AAAI Press.

Acknowledgements

First and foremost, I would like to thank my advisor, Dr. Oren Salzman, for being an incredible supervisor and mentor, for inspiring me along the course of my studies and for helping and guiding me with endless patience and passion for the research world.

Additionally, I would like to thank Prof. Yuval Filmus from the faculty of computer science for his helpful insights and guidance that contributed a lot to my work.

I would also like to thank my friends from the Computational Robotics Lab in the computer science faculty for the helpful discussions and collaborations, and to my beloved wife which stands with me in every step I take.

Lastly, I would like to devote this work to my parents—everything I achieve is thanks to them.

The generous financial help of the Technion is gratefully acknowledged.

Contents

List of Figures

Abstract

The problem of Multi-Agent Path Finding (MAPF) calls for finding a set of conflict-free paths for a fleet of agents operating in a given environment. This problem has attracted high interest among the artificial-intelligence and robotics community in the recent years. There exist different real-world application which can be modeled and developed based on the theory behind the MAPF problem.

Even though planning an optimal path for a single agent can be done efficiently, solving the MAPF problem optimally is computationally intractable in the general case. Nevertheless, there are several algorithmic approaches for solving the MAPF problem optimally that are able to solve non-trivial problem instances effectively.

Arguably, the state-of-the-art approach to computing optimal solutions is Conflict-Based Search (CBS). Although this approach and its many improvements allow to solve non-trivial instances of the problem in practice, CBS sometime fails to solve MAPF instances even when allowed long periods of run-time. Interestingly, such instances do no exhibit notable differences from easy ones. That is, there is still a significant lack of understanding in the community regarding the causes for the hardness of the problem and the ability of different solvers to cope with it.

In this work we address this issue, and try to shed light on the main aspects that affect the problem's hardness. We do so by revisiting the complexity analysis of CBS to provide tighter bounds on the algorithm's run-time in the worst-case. Our analysis paves the way to better pinpoint the parameters that govern (in the worst case) the algorithm's computational complexity.

Our analysis is based on two complementary approaches: In the first approach we bound the run-time using the size of a Multi-valued Decision Diagram (MDD)—a layered graph which compactly contains all possible single-agent paths between two given vertices for a specific path length. In the second approach we express the run-time by a novel recurrence relation which bounds the algorithm's complexity. We use a generating functions-based analysis in order to tightly bound the recurrence.

This analysis provides new upper-bounds on CBS's complexity. The results allow us to improve the existing bound on the run-time of CBS for many cases. For example, on a set of common benchmarks we improve the upper-bound by a factor of at least 2^{10^6}. We also provide an extended discussion, which hopefully will allow to extend this line of research, toward improving our understanding of the MAPF problem's hardness.

Chapter 1

Introduction

The *Multi-Agent Path Finding problem* (*MAPF*) is a well-studied problem, which attracts high interest among the robotics and AI community. It can be used to model many real-life applications, from automated warehouses (Wurman et al., 2008), through computer games (Sturtevant, 2012) and to autonomous vehicles (Pallottino et al., 2007). Therefore, much effort is invested in order to solve the problem as efficiently as possible, under different models and for different scenarios.

In the general version of MAPF (Stern et al., 2019) we are given a graph $G = (V, E)$ with n vertices and a set of k agents $A = \{a_1, a_2, ..., a_k\}$. Each agent a_i is provided with a start and a goal location, (s_i, g_i) s.t. $s_i, g_i \in V$. Time is discretized and at every time-step an agent can either *wait* in its current location or *move* across an edge to an adjacent vertex. A *feasible* solution is a paths set $\mathcal{P} = \{p_1, p_2, ..., p_k\}$ such that p_i is a path for agent a_i from s_i to g_i, and there is no conflict between any two paths in $\mathcal{P}$. We consider two types of conflicts—a *vertex-conflict*, in which two agents occupy the same vertex at the same time-step, and an *edge-conflict*, in which two agents traverse the same edge from opposite sides at the same time-step. An *optimal* solution is a paths set $\mathcal{P}$ which also optimizes some objective function. Arguably, the most common objective functions used for MAPF are:

1. *Makespan*—where we want to minimize the time in which the last agent arrive at its goal.

2. *Sum-of-Costs (SoC)*—where we want to minimize the combined time it took for all agents to arrive at their goals.

The task of finding an optimal solution is known to be NP-hard (Yu, 2016) for both aforementioned objectives. The problem remains NP-hard even when G is a subgraph of a planar grid graph (Banfi et al., 2017). Nevertheless, state-of-the-art optimal algorithms are able to effectively solve many non-trivial instances.

The approaches for solving the MAPF problem can be divided into two algorithmic families:

1. **Search-based algorithms**—in which the algorithm searches through the space of possible solutions in order to find an optimal solution.

2. **Compilation-based algorithms**—in which the MAPF instance is being translated into an instance of a different problem, and is solved using a known solver for the different problem.

In our work we focus on search-based algorithms and mainly discuss Conflict-Based Search (which we present shortly). Nevertheless, compilation-based algorithms, and mainly the SAT-based approach, constitute a large and important portion of recent work on the MAPF problem. In addition, we later argue that our work can also be adapted to exploit ideas from compilation-based algorithms. Therefore, we address both algorithmic techniques in this introduction and explain the main ideas of SAT-based solvers for MAPF as well. State-of-the-art algorithms from both disciplines have achieved impressive results dealing with non-trivial MAPF instances. There is no clear evidence for one algorithm or technique that stands above all others, and which algorithm is better usually depends on the structure of the instance (Kaduri et al., 2021).

Search-Based Algorithms for MAPF

Solving the MAPF problem with a simple A* approach is usually ineffective. Such an algorithm would require to search in the joint space resulting in a graph whose size and branching factor are exponential in the number of agents. Therefore, other search-based approaches were suggested, in which the main idea is to start from a possibly infeasible solution and work iteratively to make it feasible. Arguably, the two main algorithms under this category are *Increasing-Cost Tree Search* (ICTS) (Sharon et al., 2013) and *Conflict-Based Search* (CBS) (Sharon et al., 2015).

Increasing-cost tree search. ICTS first finds a shortest path for each agent independently and constructs a solution with a given cost (induced by the found paths). It then systematically increases the cost of the solution and searches through the solutions space for each cost, until it finds a feasible solution. The search in each level (for any given cost) is not done exhaustively, but rather with dedicated techniques that make the search more efficient. Such techniques include fast pruning mechanism that allow to quickly eliminate irrelevant solutions. The search for possible solutions in each layer is done using dedicated data structures that capture a large portion of all possible solutions, to allow fast identification of feasible solutions.

Conflict-based search. CBS, which is one of the most commonly used algorithms for solving MAPF problems optimally, starts the same way as ICTS and constructs a solution by finding a shortest path for each agent. It then works in two levels to construct a feasible solution—in the *high-level search* it systematically identifies and

resolves conflicts in the solution until obtaining a feasible solution. After CBS finds a conflict it applies a *constraint* that prohibits a conflicted agent from being in the location of the conflict at that certain time-step. It then works in the *low-level search* to plan a new path for the newly constrained agents. We give a detailed explanation of CBS in Section 2.2.

A recent work combines CBS and ICTS, taking "the best of both worlds", in order to construct an improved algorithm which is less limited on instances that are difficult for each of the algorithms separately (Walker et al., 2021).

CBS improvements. CBS was found to be highly effective in coping with different types of MAPF scenarios, thus, many studies followed this approach and introduced different techniques that allow to empirically improve the algorithm's run-time. One of the earliest, most significant improvement for the algorithm is the *prioritize conflicts* technique which uses *conflict classification* (Boyarski et al., 2015). Identifying and prioritizing conflicts that have to be solved in order to find an optimal solution (called *cardinal conflicts*) can be done efficiently. Prioritizing conflicts decreases the run-time of CBS significantly.

Other improvements based on the technique of conflicts classification also exist. Developing dedicated heuristics for the algorithm's high-level search is a natural modification. The main heuristics which exist for CBS are based on finding and counting cardinal and other types of conflicts (Felner et al., 2018; Li et al., 2019a; Boyarski et al., 2021). *Mutex-propagation* (Zhang et al., 2020) is another improvement which considers conflict classification and is used to try and approximate the reachable states in the search-space efficiently, without the necessity of solving all conflicts explicitly.

Another recent work introduced the idea of CBS with *disjoint splitting* (Li et al., 2019b). In this version of the algorithm, splitting the high-level search due to a conflict is done by applying two constraints on the same agent—one that prohibits it from being at the location of the conflict and one that required it to be there. By doing that, the algorithm maintains its optimally guarantee while reducing the size of the search-space much faster on average. A detailed description of the disjoint splitting method is given in Section 2.2.

A popular line of research tries to identify different symmetries in the structure of the problem, and use techniques of *symmetry-breaking* in order to speed-up CBS's execution. Such techniques include triangle-reasoning (Li et al., 2019c), corridor-reasoning and target-reasoning (Li et al., 2020a). These ideas were shown empirically to significantly reduce the run-time of CBS on a variety of cases with different properties.

One last CBS-based approach is *Iterative-Deepening Conflict-Based Search* (IDCBS) (Boyarski et al., 2020), which uses the idea of an IDA*-like search in the high-level search of CBS. It allows to utilize different aspects of the search, such as similarities between consecutive expanded nodes, in order to speed-up computation time for many instances.

Since the problem of finding an optimal solution is computationally hard for the

general case, many works tried to developed bounded-suboptimal algorithms, that trade off solution quality with efficiency of computation. *Enhanced-CBS* (ECBS) is one such algorithm which is based on CBS (Barer et al., 2014; Li et al., 2020b). Other leading approaches include different compilation-based solvers (Cohen et al., 2016; Surynek et al., 2017a). Non-bounded algorithms also exists, such as *Priority-Based Search* (PBS) (Ma et al., 2019) which solves conflicts by prioritizing one agent's path before the other.

Compilation-Based Algorithms for MAPF

Compilation-based (or reduction-based) solvers call for reducing an instance of a problem, usually in polynomial time, into an instance of a different familiar problem, for which there exist known, effective, solvers. The challenge in this technique is often keeping the reduced instance small enough, so the solver could work on it in an efficient time. Optimization problems like MAPF raise another challenge—translating the necessity of optimizing a cost function to an instance of a decision problem is not trivial, and might require extending the algorithm (i.e., not just to work with an existing solver).

There exist several studies on compilation-based algorithms for the MAPF problem, most of them based on a reduction to the boolean satisfiability problem (or SAT). SAT is one of the most studied problems in computer science. It calls for deciding whether a given boolean formula, over a set of variables, can be satisfied. SAT is known to be NP-complete (Cook, 1971). Nevertheless, many studies developed techniques, often called "SAT-solvers", for coping with the SAT problem, allowing to solve many different instances in a very effective way (Kilani et al., 2013).

Reducing an NP-hard problem to SAT is a common approach for dealing with NP-hard problems. The main idea of this concept is reducing an instance of the original problem into a boolean satisfiability problem instance, i.e., a boolean formula, using some polynomial-time construction, and then using any SAT-solver in order to find if the formula can be satisfied.

In order to exploit the abilities of SAT-solvers for the purpose of solving a MAPF instance, previous works show how to translate the set of all possible arrangements of agents throughout an interval of a given cost into a boolean formula. The formula represents the question "whether a solution exists within this given cost?". The algorithm then works iteratively, going over increasing costs, until the solver returns a positive answer. This guarantees an optimal solution.

The first studies on the topic handled MAPF under the makespan objective (Surynek, 2012, 2014). More recent work has shown how to exploit dependencies between a solution's makesapn and sum-of-costs, in order to handle MAPF under the sum-of-costs objective as well (Surynek et al., 2016). Recent works extended this idea to create improved SAT-based MAPF solver (Surynek et al., 2017b). Other works use the SAT-

based approach to solve different formulations of the MAPF problem (Surynek, 2021; Barták et al., 2021).

Motivation and Contribution

The different aforementioned approaches for solving the MAPF problem allow to cope with a variety of complicated instances effectively. Nonetheless, each one of those algorithms has limitations, and there are many cases where CBS and other approaches cannot solve the problem even when allowed very long running times (Kaduri et al., 2020). Interestingly, many such empirically-hard instances do not exhibit notable differences from easy ones, and identifying the exact source of (theoretical and empirical) hardness is an open question (Salzman and Stern, 2020).

In this work we try to address this gap in the understanding of the hardness of the problem. In the long term, we hope that this work would be a starting point for identifying and classifying measurable factors that affect the hardness of the MAPF problem and the ability of the different algorithms to solve them effectively. Such results have the potential to significantly improve our ability to handle the problem. Such improvements can include adjustment of algorithms to an instance specification (e.g., by choosing a certain heuristic), better designing the environment in a way that would facilitate with the algorithm (this approach is highly motivated by the warehouses management domain), develop an algorithm selection mechanism, etc.

Our Contribution

As explained, we hope that this work would lay the ground for more extended research on the aspects that affect the hardness of the MAPF problem. As a first step, in this work we study the computational complexity of CBS, which is one of the most commonly-used algorithm for solving the MAPF problem. We provide a modified analysis for the algorithm's worst-case complexity, while presenting new observations and tools that can be used to further improve the understanding on the problem's difficulty. The original exposition of CBS (Sharon et al., 2015) presented a (loose) upper-bound on the algorithm's complexity that is exponential both in the problem's parameters (number of agents k and number of vertices n in the graph G) and in the cost of an optimal solution. In this work we tighten this upper bound and provide a new point of view on the analysis of a worst-case scenario for the algorithm. We believe that this is a first step toward improving our understanding of the problem's hardness which, in turn, will allow to design algorithms that can solve a wider range of instances.

We suggest two novel approaches to improve the algorithm's worst-case complexity analysis. In the first approach we improve the (existing) upper-bound on the number of possible constraints that CBS might need to apply in order to find a solution. We do this by bounding the size of a *Multi-valued Decision Diagram* (MDD)—a layered

graph which compactly contains all possible paths between two vertices for a specific path length (Sharon et al., 2013). In the second approach we express the run-time of an advanced variant of CBS using a novel recurrence relation which bounds the algorithm's complexity. We compute the *generating function* (Wilf, 2006) of the recurrence and use it to tightly bound its value in order to obtain a tighter bound on the complexity of the CBS variant.

Combining the results from both approaches provides us with new tighter upper-bounds for the worst-case complexity of CBS. Beyond the new bounds, we anticipate that the computational tools we introduce will allow to obtain future improvements to the upper-bound. For example, this can be obtained via tighter bounds on the recurrence relation, or by improving the analysis of an MDD size.

In addition to the main results on CBS's bounds, we give an extended discussion that can be a starting point for future work. We discuss several other directions for further improving the algorithm's analysis in particular, and trying to utilize our results for practical purposes.

Chapter 2

Preliminaries

2.1 MAPF Formulation

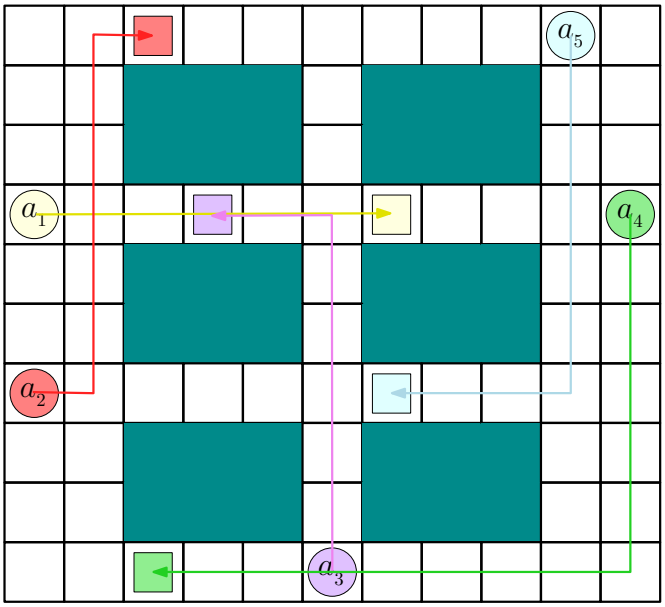


Figure 2.1: An illustration of a MAPF instance on a grid with five agents. The agents $a_1, ..., a_5$ are represented as colored circles and originate in their start location. The matching goal for each agent is marked with a square of the same color. The colored grid areas represent blocked vertices in the environment. The colored arrows represent an optimal solution.

We assume the standard setting of the MAPF problem, where we are given a graph G with n vertices and a set of k agents with start location s_i and goal location g_i for each agent a_i. The task is to plan a path for each agent, such that the total paths set, denoted by $\mathcal{P}$, is feasible, i.e., paths do not collide with each other. We say that $\mathcal{P}$ is an optimal solution if it optimizes some predefined objective function.

Given a solution $\mathcal{P}$, denote by $\mathcal{T}(p)$ the time that a single agent's path $p \in \mathcal{P}$ terminates (note that wait moves are counted as a time-step in a path p). Now, set C to be

the latest time that a single-agent's path $p \in \mathcal{P}$ terminates. Namely, $C = \max_i \{\mathcal{T}(p_i)\}$.

Under the *Makespan* objective, C constitutes the cost of the optimal solution $\mathcal{P}$. Thus, for the rest of this work we will consider C as the cost of an optimal solution of the problem, to be used in the complexity analysis and discussions. Notice that kC constitutes an upper bound on the cost of an optimal solution for the *Sum-of-Costs (SoC)* objective, where the cost is determined by the total sum of $\mathcal{T}(p_i)$ for each $1 \leq i \leq k$.

In most parts of this work we mainly focus on the makespan objective, since it provides a tight bound on the time of the total execution, which is important when analyzing the time complexity of an algorithm. We further discuss the applicability of our results and techniques for the sum-of-costs objective in Section 5.2.

Figure 2.1 presents an example of a MAPF scenario with five agents on a partially blocked 2D-grid graph, and an optimal solution to this scenario under the sum-of-costs objective.

2.2 The CBS Algorithm and its Complexity Analysis

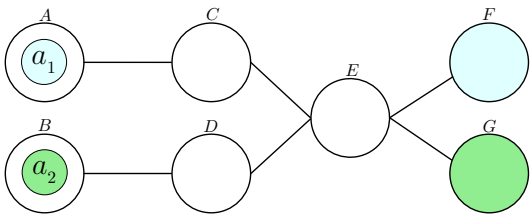


Figure 2.2: A MAPF scenario example with two agents.

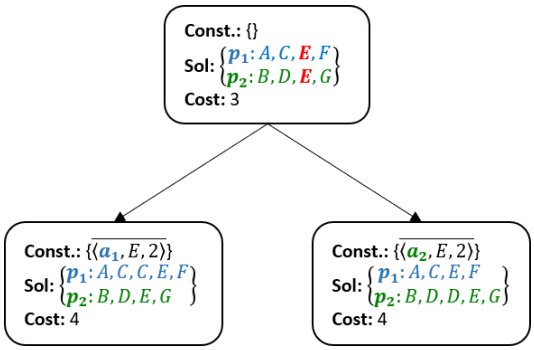


Figure 2.3: CBS's high-level search illustration, for the scenario in Figure 2.2.

Conflict-Based Search (CBS) (Sharon et al., 2015) is a two-level search algorithm which works as follows—first it finds an optimal path for each agent independently using some single-agent search algorithm like space-time A* (Hart et al., 1968; Silver, 2005). CBS then works to resolve conflicts that occur in the solution: in the high-level search it preforms a best-first search upon a constructed conflicts-tree (CT). Each node in the CT consists of a *solution*, the solution's *cost* and a set of *constraints* imposed on the agents. A constraint is either a *vertex-constraint* of the form $\overline{\langle a, v, t \rangle}$, which prohibits agent a from being at vertex v at time-step t, or an *edge-constraint* of the form $\overline{\langle a, u, v, t \rangle}$, which prohibits agent a from crossing the edge (u, v) at time-step t. We refer to such constraints as *negative constraints*.

In each iteration, CBS selects an unexpanded CT node with lowest cost. It then finds a conflict that occurs between two agents in the node's solution. It splits the CT node into two child-nodes, each with a constraint on one of the agents that were

involved in the conflict. It then runs the low-level search to construct a new solution in each child node, that does not violate the new constraint, by running a single-agent search algorithm like A*.

Figure 2.3 presents an illustration of the CT that CBS builds for the scenario presented in Figure 2.2. The original solutions (in the root node) have a vertex-conflict at vertex E at time-step 2. CBS splits the root and create two child nodes—one (left) with a constraint on agent a_1 and one (right) with a constraint on agent a_2. The solutions in each child node are feasible (no conflicts) and optimal with cost 4.

The basic version that we have just presented was recently improved using *positive constraints* (Li et al., 2019b). In a positive constraint $\langle a, v, t \rangle$, agent a is required to be at vertex v at time-step t. When using positive constraint with CBS, a CT node is split into two child nodes using a positive and a negative constraint, forcing and forbidding the conflicted agent to be at a vertex or edge at a certain time-step, respectively.

CBS's Complexity Analysis

An analysis of the worst-case time-complexity of CBS was originally presented by Sharon et al. (2015). They show that CBS's complexity can be decomposed to bounding the size of the CT and the complexity of the single-agent search in each low-level iteration. We refer to the size of the CT in a worst-case scenario as the *high-level search complexity*. The low-level search complexity corresponds to running A* for a single agent. In the main part of this work we focus on analyzing the high-level search complexity. An overall upper bound on CBS's complexity can obtained by simply multiplying any of the following results with the complexity of a single agent's A*-search.

The original analysis uses the following assumption for a worst-case scenario for the algorithm:

Assumption 2.2.1. In a worst-case scenario for CBS, each agent is constrained to avoid every vertex except one at every time-step in an optimal solution. [1]

Assumption 2.2.1 implies that each agent can potentially be in every vertex at every time-step, which can cause conflicts that CBS would need to resolve in order to find a solution. This bounds the number of (negative) constraints that CBS might need to apply by $\mathcal{O}(nkC)$. At each CT node, exactly one constraint is added. The result deduced from the above is summarized in the following observation:

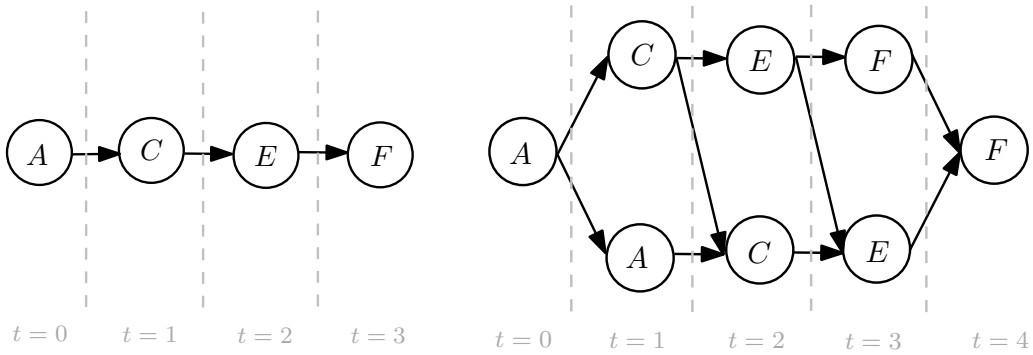
Observation 2.2.2. The number of possible constraints that CBS might need to apply bounds the depth of the CT. Therefore, $\mathcal{O}(2^{knC})$ is a bound on the maximal size of the algorithm's CT, and $\mathcal{O}(nC \cdot 2^{knC})$ is an overall bound on CBS's running time (where

[1] The original paper contains a minor error in the calculation of the upper bound. The bound presented here is the new bound whose validity was verified with one of the authors. Similarly, the oversight regarding not accounting for edge constraints (explained shortly) was also discussed and verified with one of the authors in the original CBS paper.

the nC factor corresponds to the complexity of each low-level search iteration, in the worst-case). We refer to this analysis and bounds as the *original analysis* and *original upper bounds*, respectively.

It is important to note that the original analysis does not account for the possibility that CBS would apply edge-constraints (in order to resolve conflicts). It is possible that in a worst-case scenario the algorithm would require not only to prevent any agent from occupying any vertex in the graph at each time-step, but also from crossing each edge of the graph, in order to find the optimal solution. Therefore, accounting for edge constraints should further increase the theoretical upper bound. For clarity of exposition, we present our tools for analysing CBS's complexity with the same assumption, i.e., that only vertex-constraints are considered. Nevertheless, we address this issue in detail in our discussion (Section 5.2) and show how to incorporate edge-constraints in the complexity analysis.

2.3 Multi-valued Decision Diagrams (MDD)



(a) MDD_1^3 (b) MDD_1^4

Figure 2.4: Examples of MDD graphs for agent a_1 from the scenario presented in Figure 2.2, for path lengths 3 and 4. t is the layer index in the MDD.

The *multi-valued decision diagram* MDD_i^C is a layered graph that consists of C layers, which compactly contains all possible paths of agent a_i of cost at most C from s_i to g_i (Sharon et al., 2013). A vertex $v \in V$ appears at the t'th layer of MDD_i^C if it is reachable from s_i and g_i in t and $C - t$ steps, respectively. Finally, the *size* $\mathcal{M}$ of an MDD represents the total number of MDD nodes, and the size $\mathcal{M}_t$ of the t'th layer is the number of MDD nodes in that layer.

It is also possible to construct the cross-product of the MDDs for a set of agents. The cross-MDD contains in each layer the set of cross-products of vertices which appear at the layer in each agent's individual MDD. The cross-MDD is a very useful data structure which encapsulates a lot of information about the dependencies between agents. For instance, by searching through a cross-MDD of two agents, it is easy to detect conflicts

between the agents, which can be used for the high-level search of CBS or for path pruning in different algorithms, such as ICTS (Sharon et al., 2013).

MDD graphs are commonly used for different purposes in MAPF algorithms, since they can be constructed efficiently for a given cost, and their compact representation contains information that can help improve the identification and classification of conflicts (Li et al., 2019a; Zhang et al., 2020). In this work we use MDDs to bound the number of possible constraints that might need to be applied on a single agent during a CBS execution.

2.4 Generating Functions Approach for Bounding a Recurrence Relation

Generating functions are a well-known mathematical tool which, among other things, can be used to bound recurrence relations. Formally, a generating function of a sequence $a_0, a_1, a_2, \ldots$, with the general element denoted by a_r, is the function

$$F(x) = \sum_{r \geq 0} a_r x^r,$$

i.e., the sequence elements are the coefficients of the series expansion of $F(x)$. This notion can be extended to sequences (or recurrence relations) with multiple variables. For instance, given a recursion $T(r, s)$ which defines a sequence, a possible generating function for it will be of the form

$$F(x, y) = \sum_{r,s \geq 0} T(r, s)\, x^r y^s.$$

For further details on generating functions see, e.g., the book by Wilf (2006).

Given a generating function for a specific sequence, there are many methods that allow to utilize the function in order to bound the value of the sequence at a certain index. These different methods depend on the sequence and the obtained function and there is no guarantee that a certain method could always be applied for this purpose.

The work of Pemantle and Wilson (2008) provides one approach for dealing with recursions of multiple variables which we briefly describe (additional details are presented mainly in Section 3 of the aforementioned paper) as it will be a key technique used to obtain our complexity bounds. Assume that we are given a recursion $T(r, s)$ and a matching generating function for it $F(x, y) = \sum_{r,s \geq 0} T(r, s)\, x^r y^s$ that can be expressed in the following form:

$$F(x, y) = \frac{G(x, y)}{H(x, y)},$$

where G and H are polynomials.

Denote by H_z the partial derivative of H with respect to z (where z can be a sequence of x and y). The first step calls for finding *critical points*, which are given by the solutions in the positive quadrant (i.e., $x, y \geq 0$) of the following system:

$$\begin{cases} H = 0 \\ sxH_x = ryH_y. \end{cases} \tag{2.1}$$

Denote the critical points by $q_1, q_2, \ldots, q_m$. Each point $q_i = (x_i, y_i)$ *contributes* a certain factor to the approximation of $T(r, s)$, and this contribution can be calculated according to the point's multiplicity. The exact way each point contributes to the bound is detailed by Pemantle and Wilson (2008) and in Appendix. A.

Assume that the contribution of q_i is given by $T_i(r, s)$ for each $1 \leq i \leq m$, then the analysis suggests that the asymptotic growth of $T(r, s)$ can be tightly approximated by one of the factors which is given by the critical point's contribution.

Chapter 3

Complexity Analysis for CBS using an MDD Size Bound

3.1 Bounding the Number of Constraints using MDD Size

The original analysis of CBS's complexity, explained in Section 2.2, follows Assumption 2.2.1, which unnecessarily assumes that each agent can be at any vertex at any time-step. Therefore, the algorithm might need to apply a constraint on the agent at any vertex at any time-step. In practice, we observe that for the most part of an execution, an agent can only be located in a small local area of the graph. In this section we follow this observation and perform a refined analysis of the number of possible constraints that CBS may apply on a single agent. [1] We do that by bounding the size an agent's MDD graph.

We suggest a new approach to bound the number of possible constraints that CBS may apply, using the following observation:

Observation 3.1.1. Given an agent a_i and an optimal solution cost C, the maximal number of negative constraints that CBS may apply on a_i is bounded by the size of MDD_i^C.

Observation 3.1.1 holds as CBS may only apply a constraint on agent a_i at vertex v for time-step t if a_i can reach v from s_i within t time-steps and still reach g_i in $C - t$ time-steps, which is the exact definition of an MDD node.

By combining Observation 3.1.1 with Observation 2.2.2 regarding the connection between the number of constraints and the upper bound of CBS, we obtain the following corollary:

[1] Recall that, as explained in Section 2.2, the original analysis did not account for edge constraints as a possible means that can be used by the algorithm. We temporarily limit our analysis to account for vertex constraints only, and defer handling edge constraints to Section 5.2.

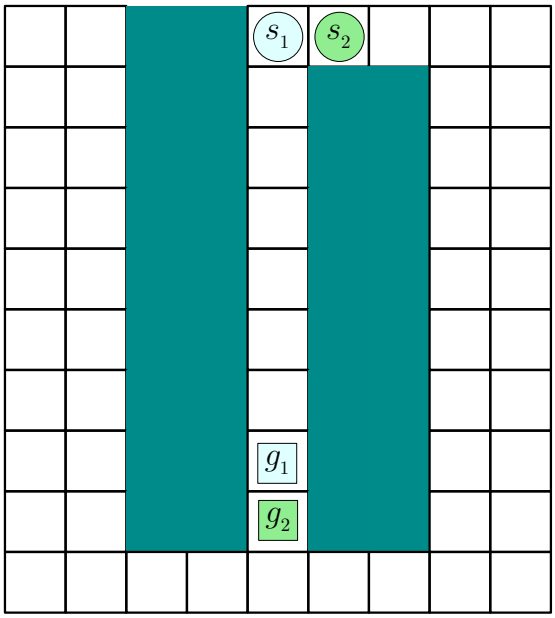


Figure 3.1: An empirically difficult scenario for CBS, where it is possible to increase the size of the graph without changing the difficulty.

Corollary 3.1. *Let $\mathcal{M}$ denote the maximal size of an agent's MDD in a given instance. The size of CBS's conflict-tree is bounded by $\mathcal{O}(2^{k\mathcal{M}})$ for any execution of the algorithm on this instance. This implies a similar bound on the algorithm's high-level search complexity.*

Corollary 3.1 can be used to recover the original analysis of CBS—a (loose) bound on $\mathcal{M}$ can be obtained by bounding the size of any MDD layer by n. Thus, $\mathcal{M} = nC$, which gives us the original bound of $\mathcal{O}(2^{knC})$. In general, Corollary. 3.1 allows us to improve the bound on CBS as long as we can tighten the bound on $\mathcal{M}$. In addition, we want to emphasize that expressing the bound using the size of the graph n may be problematic, since this size of the graph does not necessarily indicate that an instance is more difficult (or simpler). Moreover, it is easy to construct empirically complicated scenarios for CBS in which it is possible to add vertices to the graph without affecting the difficulty of the problem.

Figure 3.1 demonstrates such scenario. In the scenario depicted in the figure, an optimal solution is composed of the two agents crossing along the narrow corridor, switching order at the bottom row and entering back to reach their destinations. Finding this solution is difficult for CBS (even if the corridor were much shorter). In this case, it is straightforward to show that adding vertices at the left, right or bottom of the grid (that is, increasing the size of the graph) would not change the difficulty of the instance whatsoever (making it neither harder nor easier to solve).

For the simplicity of the exposition, we restrict the discussion in this chapter to MAPF instances on 4-connected grids (Banfi et al., 2017; Stern et al., 2019). That is, we consider the setting where an agent can move in four directions from any vertex

in the graph. Nonetheless, we emphasize that the technique we use to bound CBS's complexity can be used to bound the size of an MDD for any environment.

We assume that G is a complete grid with no blocked vertices. We also give the following observation:

Observation 3.1.2. The maximal MDD size is obtained for the case where the agent's start and goal locations are located at the same vertex, i.e., $s_i = g_i$.

It is not hard to see that when the start and the goal locations are not at the same vertex (namely, $s_i \neq g_i$), then getting them closer to each other can only increase the number of vertices that hold the necessary conditions to be included in the t'th layer of the agent's MDD (distance at most t from s_i and at most $C - t$ from g_i).

We follow Observation 3.1.2 in our analysis, that is, we assume that in the worst-case, s_i and g_i are located at the same vertex. Observe that this case serves as an upper bound on the size of MDD_i^C for any other instance. In our discussion we address the more general case, in which the start and goal locations have some distance between them, and suggest how to express the bound on the number of constraints using that distance.

Here, we present two approaches for calculating the bound on $\mathcal{M}$:

1. **General refined bound on MDD size**—the first analysis approach removes the number of environment vertices from the complexity analysis, eliminating the possibility to deem a problem computationally hard just by adding inconsequential vertices to the environment.

2. **MDD size based on graph radius**—the second approach accounts for the structure of G. In addition to the obtained bound, it demonstrates a complexity analysis restricted to a specific setting (the graph's radius which will be defined shortly). This allows to (potentially) obtain tighter bounds on the size of an MDD which, in turn, provides tighter bounds on CBS's complexity for a given setting of interest.

3.2 General Upper Bound on the Size of an MDD

To avoid accounting for the environment's boundary, we assume in this section that the graph is an infinite 4-connected grid, in addition to the assumptions that were mentioned in the previous section. For any optimal path, an agent a_i can't be located at any vertex within distance larger than $C/2$ from its start s_i or goal g_i. This implies a symmetry on the structure of MDD_i^C—the last $\lfloor C/2 \rfloor$ layers form a mirror-image of the first $\lfloor C/2 \rfloor$ layers. The number of vertices on a grid which are reachable from s_i within exactly t steps is $4t$ (see Figure 3.2). At time-step t, a_i can be located at any vertex within distance at most t from s_i. Therefore, we sum the number of reachable

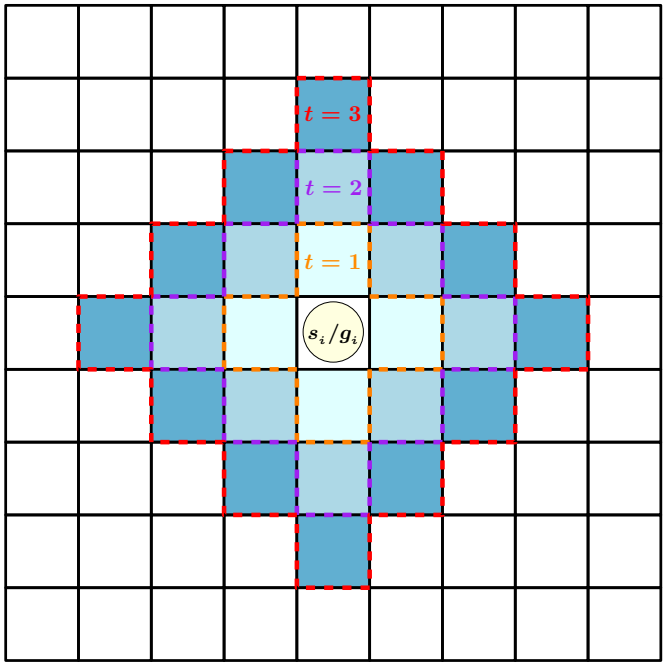


Figure 3.2: Illustration of vertices reachable for an agent with start and goal at s_i and g_i located at the same vertex, within distance of 1–3 time-steps, on a 4-connected grid. Note that the t'th MDD layer includes all the vertices in the matching surrounding dashed line.

vertices in the range from 1 to t (excluding s_i). For any $t \leq C/2$ the size of the t'th layer in MDD_i^C is:

$$\mathcal{M}_t \leq \sum_{i=1}^{t} 4i = 2t(t+1). \tag{3.1}$$

Given the aforementioned symmetry:

$$\mathcal{M} \leq 2 \cdot \sum_{t=1}^{C/2} 2t(t+1) = \frac{C^3 + 6C^2 + 8C}{6} = \mathcal{O}(C^3). \tag{3.2}$$

We assume for simplicity that C is even, if C is odd then the size of the middle layer ($\mathcal{O}(C^2)$ according to Equation 3.1) needs to be added to the result of Equation 3.2.

By placing the result from Equation 3.2 in Corollary 3.1, we obtain that:

Lemma 3.2.1. *The high-level search complexity of CBS on grid graphs is bounded by* $\mathcal{O}\left(2^{kC^3}\right)$.[2]

The bound in Lemma 3.2.1 provides a tighter estimate for the algorithm's complexity for any instance where: $C^3 < n \cdot C \Rightarrow C < \sqrt{n}$. In addition, this result removes n from the bound expression. As was explained earlier, eliminating the size of the graph from

[2]Note that in general it does not hold that $2^{\mathcal{O}(m)} = \mathcal{O}(2^m)$. The reason for which it does hold in this case is since the hidden constant in the Big-O notation in Equation 3.2 is smaller than 1.

the result gives a bound which can often reflect the behavior of the algorithm in a worst-case scenario more accurately.

3.3 MDD Size Analysis Based on the Graph's Radius

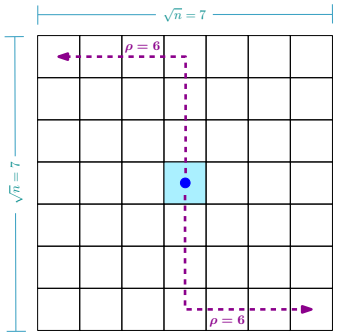


Figure 3.3: The radius $\rho = \sqrt{n} - 1$ on a grid graph with no obstacles.

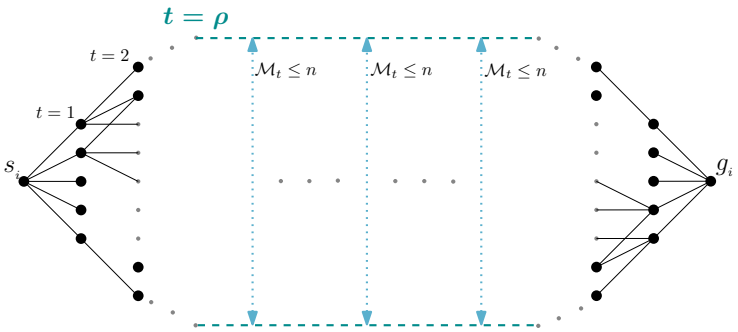


Figure 3.4: An illustration of an MDD graph for the case where $C = 2\rho + \delta n$. The first and last ρ layers are bounded according to Equation 3.1, while the δ middle layers can reach the size of the entire graph in the worst-case.

Definition 3.3.1. The **distance** $dist\,(u, v)$ between two vertices u, v in a graph is the number of edges on a shortest path between them.

Definition 3.3.2. The **radius** of a graph is $\rho = \min_{u \in V} \max_{v \in V} \{dist\,(u, v)\}$. A vertex u for which it holds that $\forall v \in V : dist(u, v) \leq \rho$ is called a **center** vertex.

In this section we express the bound on $\mathcal{M}$ using the graph's radius, which gives us an explicit value for the borders of the grid. Thus, in this case we don't need to assume an infinite grid.

We assume that G is a complete square grid with n vertices (i.e., a $\sqrt{n} \times \sqrt{n}$ grid). We have that $\rho = \sqrt{n} - 1$ (see an example in Figure 3.3), with the center in the $\lceil \frac{\sqrt{n}}{2} \rceil$'th row and column for an odd value of $\sqrt{n}$. When $\sqrt{n}$ is even, there is no single center vertex. For simplicity, we assume that $\sqrt{n}$ is odd. An illustration for the structure of the obtained MDD in the worst-case for this setting is depicted in Figure 3.4.

Observation 3.3.3. A layer of size n exists in MDD_i^C only if $C \geq 2\rho$ (note that a layer's size can't exceed n).

Observation 3.3.3 allows us to characterize settings for which it is possible to refine our analysis from Section 3.2. For the first and last ρ layers of MDD_i^C we bound a layer's size using Equation 3.1. Using Observation 3.3.3, the remaining layers are the only layers with size n. This gives us the following bound for $\mathcal{M}$, for cases where $C = 2\rho + \delta$ for some $\delta \in \mathbb{N}$:

$$\begin{aligned} \mathcal{M} &\leq \delta n + 2 \cdot \sum_{t=1}^{\rho} 2t(t+1) \\ &= \frac{4}{3} \cdot \rho(\rho+1)(\rho+2) + \delta n = \mathcal{O}(\rho^3 + \delta n). \end{aligned} \tag{3.3}$$

The expression $\frac{4}{3} \cdot \rho(\rho+1)(\rho+2)$ is smaller than $2 \cdot \rho^3$ for any $\rho \geq 7$. By placing the result from Equation 3.3 in Corollary 3.1, we obtain that:

Lemma 3.3.4. *The high-level search complexity of CBS on grid graphs with radius $\rho \geq 7$, where $C = 2\rho + \delta$ for some $\delta \in \mathbb{N}$, is bounded by $\mathcal{O}\left(2^{k \cdot (2\rho^3 + \delta n)}\right)$.*

Lemma 3.3.4 not only allows to express the bound in terms of a new (and arguably, more relevant) parameter (namely, the radius of a graph), it also provides a slightly tighter bound on the overall complexity for complete grid graphs, as long as the radius is not too small. The new bound is tighter than the original bound for cases where: $k \cdot (2\rho^3 + \delta n) < kn \cdot (2\rho + \delta) \Longrightarrow \rho < \sqrt{n}$ (which, indeed holds for complete grids). The hidden constant in the Big-O notation in both the new and the original bounds is small and does not affect the asymptotic comparison between them.

Chapter 4

Complexity Analysis for CBS using a Recurrence Relation

4.1 Recurrence Relation which Bounds CBS's Worst-Case Complexity

The structure of CBS's conflict-tree is inherently recursive. Naturally, the size of the maximal tree rooted at a certain CT node is composed of the maximal size of each of the child nodes' sub-trees. In addition, using a recursion to bound the size of the CT allows us to slightly address dependencies between different agents, i.e., a case where a constraint applied by CBS on one agent also eliminates possible future constraints of other agents. The original complexity analysis ignores such cases completely.

To this end, we introduce a novel recurrence relation which bounds the high-level search complexity of CBS. More precisely, it bounds the maximal number of CT nodes that might be generated during the high-level search. We provide an upper bound on this recurrence relation that allows to improve the original bound on the run-time of CBS for many cases.

Our improved bound incorporates the fact that recent CBS variants use *positive constraints* (Section 2.2), which were recently introduced by Li et al. (2019b). This is in contrast to the original analysis that only considers negative constraints. However, the method in which the recursion is defined is not tied to positive constraints. We choose to incorporate this improvement into the analysis since, unlike other improvements (such as conflicts-classification and different heuristics), it is independent of the structure of the specific instance, and its application in CBS is relatively simple and robust. Nonetheless, we believe that tighter bounds may be obtained in the future by defining a similar recursion for improvements of CBS, such as CBS with symmetry-breaking (Li et al., 2019c, 2020a).

Given a MAPF instance with k agents on a graph of size n where the optimal cost of a solution is C, our goal is to bound the maximal number of CT nodes that might be generated by CBS after applying a given number of positive and negative constraints.

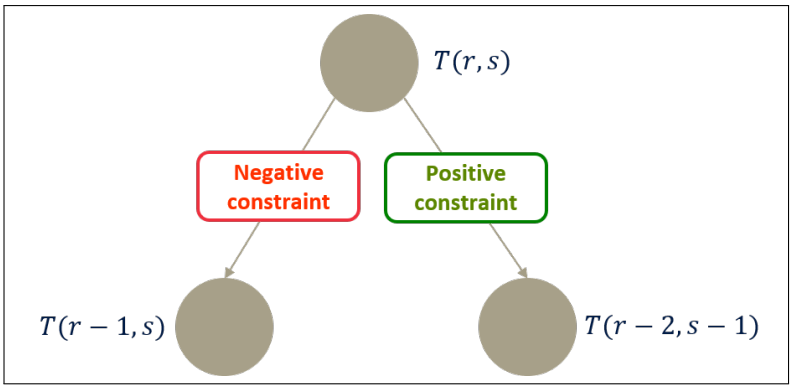


Figure 4.1: An illustration of the recursion presented in Equation 4.1.

CBS will terminate if:

1. All possible negative constraints were applied (this assumption is similar to the one used for the original bound).

2. The algorithm applied C positive constraints on each of the k agents.

Note that any (positive or negative) constraint applied to a CT node cannot be applied to any of its children in the CT. In addition, if agents a_i and a_j were found to be in a conflict at vertex v at time-step t, applying a positive constraint on agent a_i implies that the negative constraints $\overline{\langle a_i, v, t\rangle}$ and $\overline{\langle a_j, v, t\rangle}$ cannot appear in the sub-tree of the CT node.

From the above we get the following recurrence relation:

Lemma 4.1.1. *Let r and s denote the maximal number of negative and positive constraints that CBS may apply before it is bound to terminate, respectively. Then, the high-level complexity of CBS is bounded by:*

$$T(r,s) \leq \begin{cases} 1, & r = 0 \text{ or } s = 0 \\ 3, & r = 1 \text{ and } s > 0 \\ T(r-1,s) + \\ \quad T(r-2,s-1) + 1, & \text{else.} \end{cases} \tag{4.1}$$

For $r = 0$ or $s = 0$ we get that one of the aforementioned conditions for termination holds, therefore, the respected node is a leaf node in the CT. For $r = 1$ there is still a single negative constraint left to apply, so the node can be split only one more time, creating two additional leaves (and the node itself is also counted). For any other inner-node, the algorithm would split it according to a conflict by applying a negative constraint on one branch, and a positive constraint for the other branch (see illustration in Figure 4.1). Note that when applying a negative constraint (i.e., reducing r by one) it does not imply that a positive constraint has been applied (therefore, in the first component of the recurrence step s does not change).

We present two techniques for upper bounding the recursion presented in Equation 4.1, which in turn provide an upper bound on CBS's complexity:

1. **Induction-based analysis**—a common approach for solving a recurrence is to "guess" a solution (in an educated way) and prove by induction that it holds. This approach often requires additional relaxed assumptions which eventually give a loose result.

2. **Generating functions-based analysis**—a generating function is a common technique for analyzing recurrence relations. We explain the notion of generating functions and present a tight **empirical** upper bound for the recursion, that is obtained using the recursion generating function.

In both approaches we first bound the general form of the recurrence, i.e., we find a bound which holds for any given r and s. Recall that in our setting we have a loose upper bound on the maximal number of negative constraints, i.e., $r = knC$. The number of total positive constraints that can be applied is bounded by the cost of an optimal solution for each of the agents, i.e., $s = kC$. Therefore, we want to bound the value of $T(knC, kC)$. Notice that according to Corollary 3.1, in the general case we can replace the factor of nC in the value of r with a tighter bound on $\mathcal{M}$, if such a bound exists (for example, we can place $r = kC^3$ for grid graphs, according to the result in Equation 3.2).

We also give the following key observation, which later allows us to refine our analysis:

Observation 4.1.2. For any MAPF instance, there is a linear dependency between r, the maximal number of negative constraints and s, the maximal number of positive constraints that CBS can apply. Specifically, $r = n \cdot (kC) = n \cdot s$.

This key observation allows us to search for a bound only for the recursion nodes of the form $T(n \cdot s, s)$. We use that in our second analysis approach in order to obtain a tighter bound, which empirically holds for the relevant cases.

4.2 Induction-Based Bound

A common technique for bounding recurrence relations is by expanding the equation until reaching a boundary case and then deducing a closed-form equation. In our recursion, this approach is more complicated, because of the recursive step and the dependence on multiple variables. However, it does allow us to conjecture a possible bound that we can then prove by induction (the technique of "guessing" a bound and then proving it by induction is also a common technique for solving recursions). Note that one problem with this approach is that we do not know if the conjectured bound is tight. Nevertheless, the obtained general bound allows us to significantly reduce the bound on CBS.

Claim 4.2.1. *For any $(r,\ s)$ s.t. $r \geq 1$ and $s \geq 1$ it holds that:*

$$T(r,s) \leq 3 \cdot r^s. \tag{4.2}$$

Which implies that $T(r,s) = \mathcal{O}(r^s)$.

Proof. The proof is done by induction over pairs (r,s), assuming an order where $(r_1,s_1) \geq (r_2,s_2)$ if $r_1 \geq r_2$ and $s_1 \geq s_2$ (there exists such an order on pairs where $r,s \in \mathbb{N}$).
Base:

$$\begin{aligned}
T(1,s) &\leq 3 \leq 3 \cdot 1^s. \\
T(r,1) &\leq T(r-1,1) + T(r-2,0) + 1 \\
&\leq T(r-2,1) + T(r-3,0) + 1 + T(r-2,0) + 1 \\
&\leq \underbrace{\ldots}_{r-1 \text{ times}} \leq \underbrace{T(1,1)}_{3} + \sum_{i=1}^{r-1} (\underbrace{T(r-i-1,0)}_{1} + 1) \\
&\leq 3 + 2(r-1) = 2r + 1 \leq 3 \cdot r^1.
\end{aligned}$$

Step: We assume that the claim holds for all pairs smaller than (r,s) and prove it for (r,s):

$$\begin{aligned}
T(r,s) &\leq T(r-1,s) + T(r-2,s-1) + 1 \\
&\underset{\text{i.h.}}{\leq} 3(r-1)^s + 3(r-2)^{s-1} + 1 \\
&\underset{*}{\leq} 3(r-1)^s + 3(r-1)^{s-1} \\
&= 3[(r-1)^{s-1} \cdot (r-1) + (r-1)^{s-1}] \\
&= 3(r-1)^{s-1} \cdot (r-1+1) \\
&= 3r \cdot (r-1)^{s-1} \leq 3r \cdot r^{s-1} = 3r^s.
\end{aligned}$$

Where $*$ holds because $3(r-1)^{s-1} \geq 3(r-2)^{s-1} + 1$ for $r > 1$ and $s > 1$. ■

Recall that negative and positive constraints are bounded by $r = k\mathcal{M}$ and $s = kC$, respectively (where $\mathcal{M}$ is the size of an MDD graph of a single agent). Placing those values in Equation 4.2 gives the following result:

$$T(k\mathcal{M}, kC) \leq \mathcal{O}\Big((k\mathcal{M})^{kC}\Big). \tag{4.3}$$

From Equation 4.3 we obtain the following lemma:

Lemma 4.2.2. *Let $\mathcal{M}$ be an upper bound on the size of an agent's MDD graph in the worst-case. Then, the time-complexity of the high-level search of CBS is bounded by $\mathcal{O}\Big((k\mathcal{M})^{kC}\Big)$.*

By taking $\mathcal{M} = nC$ (i.e., the bound on an MDD size considered in the original analysis), we get an upper bound of $\mathcal{O}\Big((knC)^{kC}\Big)$, in contrast to the original bound

of $\mathcal{O}(2^{nkC})$. Here again, the hidden constants in the Big-O notation in both bounds are small, thus, we omit them in the upcoming comparison between the bounds (Section 4.4). The transition of n from the exponent to the basis of the expression, allows to significantly reduce the bound for most standard MAPF scenarios. In most scenarios the graph is large compared to the other parameters, and also does not necessarily contribute to the difficulty of the scenario (as explained in Section 3.1).

We can also apply the results we obtained in Sections 3.2 and 3.3 to obtain additional bounds expressed with different parameters of the problem, which are also tighter for different settings. By substituting the bound $\mathcal{M} = \mathcal{O}(C^3)$ from Lemma 3.2 in Lemma 4.2.2 we get a bound of $\mathcal{O}(kC)^{3kC}$. For the case where the solution's optimal cost is expressed using the graph radius, i.e., $C = 2\rho + \delta n$, we can substitute $\mathcal{M} = \mathcal{O}(\rho^3 + \delta n)$ (Equation 3.3) and get a bound of $\mathcal{O}(k \cdot \rho^3 + \delta n)^{k\cdot(2\rho+\delta n)}$ on CBS's high-level complexity.

4.3 Generating Functions-Based Bound

In our second approach, we present an alternative technique to bound the recursion (Equation 4.1) using *generating functions*. This, in turn, will allow us to obtain a tighter bound on CBS's complexity. A detailed description of the mathematical tools that are used throughout the analysis is given in Appendix. A.

We start by introducing the generating function for $T(r, s)$. We then continue to follow the steps outlined by Pemantle and Wilson (2008) to obtain a bound on $T(r, s)$. We stress that the suggested analysis method is not applicable for formally proving the bound's correctness, but we can use it to deduce an asymptotic upper bound which we then support empirically for a variety of different values.

The method by Pemantle and Wilson (2008), as described in Section 2.4, calls for finding the *contribution factor* for the bound obtained by each critical point of the solution for Equation 2.1. The contribution made by each critical point is an asymptotic estimation for the value of the recurrence. The resulted estimation is the sum of all contributions, but, since we only care about an asymptotic bound, only one of the factors governs the others, in terms of the asymptotic estimation of the recurrence.

We first find the contribution of each critical point. We then apply Observation 4.1.2, which allows us to deduce a suggested upper bound for CBS. By applying Observation 4.1.2, we get that one of the contribution factors obtained from the analysis gives a tight upper bound on $T(ns, s)$.

As explained, we start with presenting the generating function for this recursion, which is:

$$F(x, y) = \frac{1 - x + 2xy - x^2y}{(1 - x)(1 - y)(1 - x - x^2y)}. \tag{4.4}$$

We denote $F = G/H$, where

$$G(x, y) = 1 - x + 2xy - x^2y,$$
$$H(x, y) = (1 - x)(1 - y)(1 - x - x^2y),$$

and solve Equation 2.1 to find the critical points. In this setting there are three such points:

$$q_1 \coloneqq (x_1, y_1) = \left(\frac{-1 + \sqrt{5}}{2}, 1\right),$$
$$q_2 \coloneqq (x_2, y_2) = (1, 1),$$
$$q_3 \coloneqq (x_3, y_3) = \left(\frac{r - 2s}{r - s}, \frac{s(r - s)}{(r - 2s)^2}\right).$$

We denote the matching contribution factor of each point q_i by $T_i(r, s)$.

Computing the exact contribution for each point is done according to Pemantle and Wilson (2008). This involves basic (yet daunting) algebraic manipulations, and is summarized in Lemma 4.3.1 (proof omitted).

Lemma 4.3.1.

$$T_1(r, s) = 1,$$
$$T_2(r, s) = \mathcal{O}(1) \cdot \left(\frac{1 + \sqrt{5}}{2}\right)^r,$$
$$T_3(r, s) = \frac{(r - s)^{r-s}}{(r - 2s)^{r-2s} \cdot s^s} \cdot \frac{2s}{r - 2s} \cdot \sqrt{\frac{\alpha}{2\pi}},$$

where $\alpha = \mathcal{O}\left(\frac{r^2}{s}\right)$.

Following ideas from Pemantle and Wilson (2008), we can use Lemma 4.3.1 to estimate the asymptotic growth of Equation 4.1. Note that the generating function (Equation 4.4) doesn't satisfy some conditions from Pemantle and Wilson (2008), so the resulted estimation is not supported by the proof. Nonetheless, we use the results to deduce an asymptotic estimation of the recurrence value, which we then support with an empirical evaluation. Yet, using this estimation to deduce an upper bound for CBS's complexity is not straightforward.

Fortunately, by applying Observation 4.1.2 we can obtain an estimated approximation on Equation 4.1 that is tighter than the bound obtained using the induction-based analysis (Lemma 4.2.2). We do it by restricting the recurrence to values of r and s that can be attained in our MAPF setting. Specifically, using $r = n \cdot s$ in Lemma 4.3.1 we have that,

Proposition 4.3.2. *The high-level search complexity of CBS for instances with $n \geq 4$ vertices, k agents and an optimal solution cost C is bounded by $\mathcal{O}\left((en)^{kC}\right)$.*

We approximate the value of $T(ns, s)$ according to Lemma 4.3.1 and have that

$$\begin{aligned} T_1(ns, s) &= 1, \\ T_2(ns, s) &= \mathcal{O}(1) \cdot \left(\frac{1+\sqrt{5}}{2}\right)^{ns}, \\ T_3(ns, s) &= \left(\frac{(n-1)^{n-1}}{(n-2)^{n-2}}\right)^{s} \cdot \frac{2}{n-2} \cdot \sqrt{\frac{\beta}{2\pi s}}, \end{aligned} \tag{4.5}$$

where $\beta = \mathcal{O}(n^2)$.

The contribution to $T(ns, s)$ from q_1 and q_2 (given by $T_1(ns, s) + T_2(ns, s)$) is identical to the contribution from q_3 (given by $T_3(ns, s)$) at $n_0 = \frac{\sqrt{5}+2}{2} \approx 3.618033 < 4$. In Section 4.4 we empirically demonstrate that $T_3(ns, s)$ indeed constitutes a tight bound for any $n > n_0$.

Therefore, we continue with the simplification of the expression given by $T_3(ns, s)$. Since $\frac{2}{n-2} \cdot \sqrt{\frac{\beta}{2\pi}} = \mathcal{O}(1)$ and $\left(\frac{(n-1)^{n-1}}{(n-2)^{n-2}}\right) < en$, we get the following result:

$$T(ns, s) = \mathcal{O}((en)^s),$$

with a small hidden constant factor in the Big-O notation. By placing $s = kC$, which is the maximal number of positive constraints that needs to be applied by CBS in the worst case, we get the desired bound.

Proposition 4.3.2 improves the original known bound of $\mathcal{O}\left(2^{nkC}\right)$ for any set of values n, k and C. Moreover, it is also tighter than the already-improved bound presented in Lemma 4.2.2. Notice that it allows to replace the asymptotic factor of kC in the base of the exponent with a constant (e), while also still eliminating the exponential dependency in n.

New bounds can also be obtained by combining the results from Section 2.3 that bound the size of an MDD. For example, using Equation 3.2 we observe that there is a quadratic dependency between $r = kC^3$ and $s = kC$ on 4-connected grids. By simply substituting n with C^2 in the bound obtained by Proposition 4.3.2 we have that,

Corollary 4.1. *The high-level search complexity of CBS on 4-connected grids is bounded by* $\mathcal{O}\left((eC)^{2kC}\right)$.

4.4 Empirical Evaluation of Bounds

In this section we present an empirical evaluation of the obtained bounds on a variety of MAPF settings. There are three purposes for this evaluation: (1) to demonstrate the difference between the new bounds and the original one; (2) to highlight the significance of the improvement given by the new bounds and (3) to give an empirical evidence for the tightness of the estimated solution of the recurrence, given in Proposition 4.3.2.

Denote the original bound by ORG, the looser induction-based bound presented in

Benchmark	n	k	C	ORG	REC+IND	REC+GF	$\frac{\text{ORG}}{\text{REC+GF}}$
warehouse-10-20-10-2-2	9,776	8	185	2^{10^8}	2^{10^6}	2^{10^5}	$\mathbf{2^{10^7}}$
warehouse-10-20-10-2-2	9,776	64	200	2^{10^9}	2^{10^7}	2^{10^6}	$\mathbf{2^{10^8}}$
warehouse-10-20-10-2-2	9,776	128	206	2^{10^9}	2^{10^7}	2^{10^6}	$\mathbf{2^{10^8}}$
warehouse-20-40-10-2-2	38,756	64	420	$2^{10^{10}}$	2^{10^7}	2^{10^6}	$\mathbf{2^{10^9}}$
warehouse-20-40-10-2-2	38,756	128	440	$2^{10^{10}}$	2^{10^7}	2^{10^6}	$\mathbf{2^{10^9}}$
warehouse-20-40-10-2-2	38,756	256	450	$2^{10^{10}}$	2^{10^8}	2^{10^7}	$\mathbf{2^{10^9}}$
empty-48-48	2,304	64	70	2^{10^8}	2^{10^6}	2^{10^5}	$\mathbf{2^{10^7}}$
empty-48-48	2,304	128	74	2^{10^8}	2^{10^6}	2^{10^5}	$\mathbf{2^{10^7}}$
random-64-64-10	3,687	64	94	2^{10^8}	2^{10^6}	2^{10^5}	$\mathbf{2^{10^7}}$
random-64-64-10	3,687	128	102	2^{10^8}	2^{10^7}	2^{10^6}	$\mathbf{2^{10^7}}$
room-64-64-16	3,646	8	120	2^{10^7}	2^{10^5}	2^{10^4}	$\mathbf{2^{10^6}}$
room-64-64-16	3,646	128	180	2^{10^8}	2^{10^7}	2^{10^6}	$\mathbf{2^{10^7}}$
Berlin_1_256	47,540	16	324	2^{10^9}	2^{10^6}	2^{10^5}	$\mathbf{2^{10^8}}$
Berlin_1_256	47,540	256	414	$2^{10^{10}}$	2^{10^8}	2^{10^7}	$\mathbf{2^{10^9}}$
den520d	28,178	8	305	2^{10^8}	2^{10^6}	2^{10^5}	$\mathbf{2^{10^7}}$
den520d	28,178	16	333	2^{10^9}	2^{10^6}	2^{10^5}	$\mathbf{2^{10^8}}$
orz900d	96,603	16	2656	$2^{10^{10}}$	2^{10^7}	2^{10^7}	$\mathbf{2^{10^9}}$
orz900d	96,603	64	2990	$2^{10^{11}}$	2^{10^8}	2^{10^8}	$\mathbf{2^{10^{10}}}$

Table 4.1: A comparison between the different upper bounds obtained using the original analysis (ORG), Lemma 4.2.2 (REC+IND) and Proposition 4.3.2 (REC+GF), on standard benchmarks (Sturtevant, 2012; Stern et al., 2019). The last column presents a lower bound on the ratio between our improved bound and the original bound, which reflects the improvement. All bounds are calculated considering that $\mathcal{M} = nC$. The optimal cost is an averaged cost on multiple random instances for each benchmark. The exponent values are rounded such that the result constitutes an upper bound for each function's value and a lower bound on the ratio in the rightmost column. Note that all actual bounds include a small constant multiplicative factor, but the comparison in this table accounts only for the asymptotic factors.

Lemma 4.2.2 by REC+IND, and the tighter generating functions-based bound presented in Proposition 4.3.2 by REC+GF.

First, we evaluated the bound ratio, that is, the value $\frac{\text{ORG}}{\text{REC+GF}}$, for commonly-used benchmarks (Sturtevant, 2012; Stern et al., 2019). These benchmarks represent a large variety of MAPF instances (including real-life scenarios). For the second part of the evaluation, we examined the value of each bound (and the recursion from Equation 4.1) as a function of the graph's size, for different settings. A setting is composed of multiple parameters, i.e., k, C and n, with no guaranteed dependency between them. Therefore, we define several different ratios between the parameters in order to reflect as much as possible the relations between the parameters in actual MAPF settings. The behavior presented in the evaluations is consistent for all other examined ratios as well.

We emphasize that the original bound refers to the basic version of CBS, while the recurrence-based bounds leverage the structure of CBS with disjoint splitting, which is a slightly different algorithm. Nevertheless, this comparison allows us to show the improvement (in the worst-case) that is achieved by using positive constraints in CBS. It

also demonstrates the quality of the generating functions-based analysis, which allows to obtain a tight bound on the recurrence. [1]

Evaluation on Standard Benchmarks

We examined benchmarks of many types of grid-based environments. The diversity of environments represent different possible settings of the problem, with different ratios between the problem's parameters—n, k and C. The results are summarized in Table 4.1, which presents the various bounds on the worst-case complexity of CBS's high-level search. Beside the numerical evaluation of each bound, we also present the ratio between ORG and the best newly-obtained bound REC+GF, to demonstrate the improvement.

We can see that for all benchmarks, the new bound improves on the original one by a factor of at least 2^{10^6}. More precisely, for any instance we examined, we obtain a significantly tighter bound on the high-level complexity of CBS.

As expected, the bounds obtained by an analysis of the worst-case scenario of an NP-hard problem are still extremely large for standard instances. Yet, beside the significant improvement in the worst-case complexity compared to the original bound, we believe that the tools that we used in order to achieve these bounds can pave the way for further improving the evaluation of MAPF algorithms.

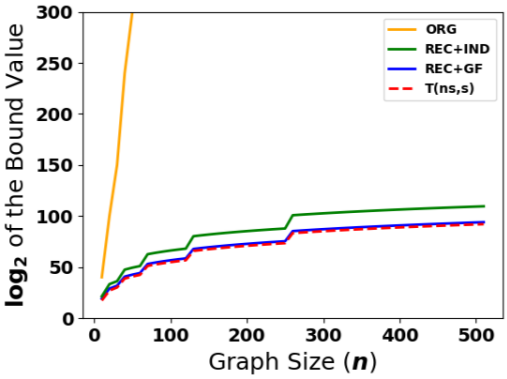
Empirical Comparison for Different Settings

One challenge that comes up when trying to evaluate the bounds is the fact that the problem's complexity depends on several parameters, with no clear relation between them. The ratios between the graph's size (n), the number of agents (k), and the optimal solution's cost (C) are not given, and diverges between different types of instances (meaning, there is no "correct" way to bound two parameters with the third one, beside maybe claiming that $k \leq n$, which is a trivial but not very interesting case).

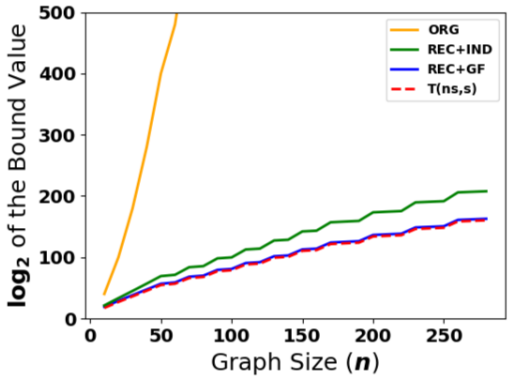
We deal with this challenge by examining the bounds for growing values of the graph's size, for different ratios between it and the product of k and C. The factor of kC appears in all bounds, and is also the value that we place in the second variable of the recursion, representing the maximal number of positive constraints ($s = kC$). The different settings we examine cover a wide range of MAPF instances, which help to strengthen the correctness of the presented results.

In addition, recall that the bound obtained in Proposition 4.3.2 is a chosen empirical estimate for the recursion, out of several options obtained from the generating functions-based analysis. To support the correctness of this estimate, we also compared its value to the recurrence's value, in all examined settings.

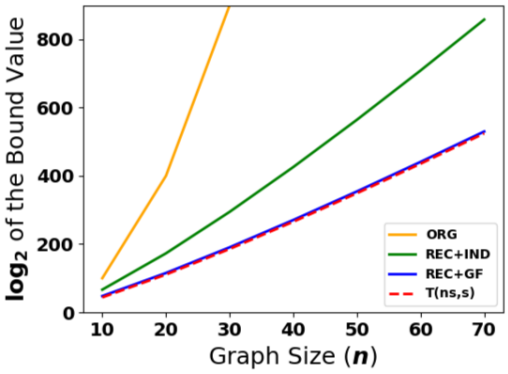
[1] According to one of the original CBS algorithm authors, using positive constraints in order to resolve conflicts is the correct and intuitive method that CBS should rely on. This strengthens the importance in providing a tight upper bound on the complexity of this version of the algorithm.



(a) $s = \log_2 n$



(b) $s = \sqrt{n}$



(c) $s = n$

Figure 4.2: The $\log_2$ of the bounds as a function of the graph size for different ratios between the graph's size (n) and the instance properties ($s = kC$). The two new bounds (REC+IND, REC+GF) are significantly lower than the original bound. Notice that the approximation obtained from the generating functions analysis (REC+GF) indeed tightly bounds the recurrence $T(ns, s)$.

Figure 4.2 presents a comparison (on a logarithmic scale) of the three bounds (ORG, REC+IND and REC+GF) and $T(r, s)$ for different graph sizes n for the setting $r = ns$ (Observation 4.1.2). Indeed, for all different cases the same trend can be observed, where the gap between the presented functions demonstrates the magnitude of the improvement obtained using our analysis. In addition, the figures serve as an empirical validation of the generating functions-based analysis—the asymptotic bound (REC+GF) tightly approximates the recurrence relation (Equation 4.1).

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this work we have studied the problem of Multi-Agent Path Finding, which calls for planning a path for a group of agents which move together in a shared environment. This problem and its different variants is relevant for many real-world application, therefore, being able to cope with a wide variety of the problem's instances effectively is an important task. In order to help improve the ability of coping with the problem, we studied the computational aspects of MAPF and one of its most common solvers, the Conflict-Based Search algorithm. Through an improved analysis of the algorithm's complexity we provided novel insights that we see as a first step toward improving our understanding of the aspects that affect the computational hardness of the MAPF problem.

We presented two novel approaches for analyzing the worst-case complexity of CBS's high-level search. In the first approach, by analyzing the size of an agent's MDD graph, we provided two new upper bounds for CBS's high-level search of: $\mathcal{O}(2^{kC^3})$ and $\mathcal{O}(2^{k\cdot(2\rho^3+\delta n)})$. The latter is a bound for a setting where a solution's optimal cost depends on the radius ρ of G. Our approach allows to seamlessly obtain tighter bounds on CBS's complexity given tighter bounds on the size of an MDD graph $\mathcal{M}$. Such bounds may be obtained either by (1) better analyzing the general structure of an MDD or (2) restricting the analysis to a specific instance of interest.

In the second approach, we presented the recurrence relation $T(r, s)$. An upper bound on $T(r, s)$ constitutes a bound on the size of the CT of CBS, therefore bounding the complexity of CBS's high-level search. Using this approach, we obtain a new general bound of $\mathcal{O}((k\mathcal{M})^{kC})$. When using $\mathcal{M} = nC$, we obtain the new induction-based bound of $\mathcal{O}((knC)^{kC})$.

Using a generating functions-based bound, we obtain a tighter bound on the recurrence which, in turn, provides a tighter bound for CBS. Observing that there exists a linear dependence between the number of negative and positive constraints, allows us to achieve a further improvement, and we eventually obtain a bound of $\mathcal{O}\Big((en)^{kC}\Big)$,

which improves the original bound on the algorithm by a significant factor for a wide range of standard benchmarks.

We believe that the recurrence relation can be further improved, in order to better express the real conditions of a worst-case scenario. An immediate step would be to try and account for tighter dependencies between the number of constraints that are being eliminated once a single constraint is applied. In addition, revisiting the conditions on the recursion's parameters may allow to tighten the upper-bound on the recursion, and in turn, on the complexity of CBS.

It is important to note that the new bounds are still somewhat loose and present a worst-case analysis. However, our analysis paves the way to better pinpoint the parameters that govern (in the worst case) the algorithm's computational complexity as well as to analyze the complexity when restricted to certain settings. Moreover, it provides a general methodology that can be used to analyze different variants of the MAPF problem. For example, in the next section (Section 5.2) we show how to seamlessly account for edge constraints as well as for settings that optimize the sum-of-costs objective.

5.2 Discussion and Future Work

In this section we present alternative analysis directions and possible applications for the result of this work, which haven't matured into completed formal results, but we hope that they can be of help to anyone working on this research topic. We address several issues and limitations that arose during the course of this work, and suggest how to handle them. In addition, we suggest several directions for using our results in order to improve the ability of dealing with the MAPF problem.

Accounting for Edge-Constraints in CBS's Analysis

Recall that the analysis we performed in Chapters 3 and 4 (as well as the original analysis from (Sharon et al., 2015)) accounted for vertex constraints only. We now show a simple approach to account for edge constraints using the existing analysis. Accounting for edge constraints directly is left for future work.

Counting edge constraints as vertex constraints Notice that an edge constraint implicitly defines two vertex constraints (forcing agent a_i to traverse (u, v) at time t corresponds to the constraint that a_i has to be at vertices u and v at times t and $t+1$, respectively). Thus, we can simply increase the number of vertex constraints by the number of possible edge constraints (which is twice the number of edges, as each edge can be traversed in both directions). Specifically, on 4-connected-grids, each node has at most four outgoing and incoming edges, therefore, using the original analysis, the number of negative constraints would increase to $nkC + 8nkC = 9nkC$ in the worst-

case. For the general case where $|E| \leq n^2$, the number of negative constraints would increase to $(2n^2 + n) \cdot kC$ in the worst-case. We emphasize that while the increase may seem negligible, the actual worst-case complexity is exponential in the number of negative constraints, and the additional constraints would appear in the exponent of the original bound. For example, the original bound of $\mathcal{O}(2^{nkC})$ would be $\mathcal{O}(2^{9 \cdot nkC})$.

A similar approach can be applied to the recursion-based bounds presented in Chapter 4. We can consider a total number of negative constraints of $r = 9nkC$, which also includes the negative edge-constraints. One can show that Equation 4.1 still upper-bounds the number of expanded nodes in CBS's high-level search. Thus, by substituting $r = 9nkC$ in Proposition 4.3.2 we obtain a bound of $\mathcal{O}\Big((9en)^{kC}\Big)$.

It is important to note that accounting for edge constraints increases the relative improvement of the bounds we present over the original bound. This is because in our bound, the additional work is reflected by increasing the base of the exponent rather than the exponent itself, as in the original bound.

Counting MDD edges In Corollary 3.1 the size of an MDD is defined as the number of MDD vertices. However, if we wish to account for edge constraints, the same result holds simply by changing the definition of the size of an MDD to be the number of MDD vertices *and* edges. For instance, on a 4-connected grid, the maximal (outgoing) degree of each MDD node is five. Therefore, the total number of MDD edges is bounded by five times the number of MDD vertices. Using Equation 3.2, the total number of vertex and edge constraints is bounded by $(1 + 5) \cdot \mathcal{O}\left(C^3\right)$. We then update Lemma 3.2.1 to incorporate edge constraints, and get a bound of $2^{\mathcal{O}\left(kC^3\right)}$ (with a small hidden constant factor slightly larger than 1).

Analysis for the Sum-of-Costs (SoC) Objective

The bounds we provided (Chapters 3 and 4) were obtained and expressed using C—an upper-bound on a single agent's path length in an optimal solution. In the setting where we seek to minimize the makespan, C is indeed an optimal solution's cost. Unfortunately, this is not the case for the SoC objective.

One way to use our results when using SoC as the optimization objective is to observe that kC is an upper bound on the optimal solution's cost. Namely, denote $C' = kC$ and use C' instead of kC in all the bounds presented. For example, our generating function-based analysis for the SoC objective yields the bound of $\mathcal{O}\Big((en)^{C'}\Big)$. Since C is an upper bound on a single-agent's solution's cost, the obtained bound on the algorithm's complexity in this case would be very loose.

MDD Size Analysis Based on the Distance between Start and Goal

In the analysis presented in Chapter 3, we provided several bounds on the size of an MDD graph on grids, all of them under a worst-case assumption that the start

and goal locations of an agent are located at the same vertex (Observation 3.1.2). Here, we suggest to remove this assumption, and assume a distance $dist(s_i, g_i) = d$ (Definition 3.3.2) between the agent's start and goal locations. This reflects a much more realistic scenario, and allows to address the bound on the number of constraints for each agent separately (which should result in a tighter total bound on the complexity). The idea is to use the ratio between an optimal solution's cost C and d in order to express the bound on $\mathcal{M}$.

The addition of a distance between the start and goal makes the analysis of the MDD size much more complicated, since it removes the symmetry over the graph's structure. Therefore, we only give a high-level idea for how to approach this analysis, and highlight possible outcomes and difficulties.

First, we observe that the case where $C < d$ is trivial, since there is no valid path for the agent which will transfer it from s_i to g_i in at most C steps. For $C \geq d$, the worst-case, which results in the largest possible MDD, is when s_i and g_i lay at the opposite corners of a square with a side of length $d/2$ (for simplicity assume d is even, otherwise bound any case where d is odd with $d + 1$).

We divide the remaining cases for (1) $C = d$ and (2) $C > d$. The first case is simple, since the agent would have to advance towards the goal at each time-step without the ability to get farther from g_i or wait at any vertex. The outcome is that the size of an MDD graph for length C between two vertices at distance d, where $C = d$, is bounded by $\mathcal{O}(C^2) = \mathcal{O}(d^2)$ (with a hidden constant smaller than 1). The explanation for this result is that in this case, the t'th MDD layer contains exactly the vertices on the t'th diagonal from the start location, which is of size of exactly $t + 1$ (for any $t \leq d/2$, and the other half is symmetric). The total MDD size is therefore:

$$\mathcal{M} \leq 2 \cdot \sum_{t=1}^{C/2-1} t + 1 + (C/2 + 1) = \frac{C^2 + 6C}{8} = \mathcal{O}\left(C^2\right). \tag{5.1}$$

By substituting the bound of Equation 5.1 in Corollary 3.1, we get a tighter bound on CBS's high-level search (for the case where $C = d$), of $\mathcal{O}\left(2^{kC^2}\right)$.

The case where $C > d$ is much more complicated to bound. We only outline the challenges that need to be solved in order to be able to give a bound for this general case. The task of finding a non-trivial bound for this case remains an open question. As a first step, we can assume that if $C \gg d$, then the asymptotic bound is reduced back to the scenario presented in Section 3.2, since the distance between s_i and g_i is negligible compare to C, and so C would govern the asymptotic bound.

So, assume that $C > d$, but is of the same order of magnitude, that is, C exceeds d by a small constant value. The main challenge in the analysis comes from the fact that in this case the agent might take steps that get it farther from the goal, but still have enough steps "left" to reach the goal within C time-steps. The number of these additional vertices that are added to the MDD can't be precisely bounded. It gets more

complicated, because the steps which take the agent farther from g_i can be taken at any point during the planning, which would increase each layer in the agent's MDD. We believe that in order to provide a non-trivial bound for this case, more assumptions are required (on the possible movement of the agent "outside of the $d/2 \times d/2$ square", on the location of the start and the goal, etc.).

Improve CBS in Practice

Throughout this work we present several observations, which we see as a first step to improve our understanding of the hardness of the MAPF problem. For instance, in Chapter 3 we discuss the dependence of the graph's size on the problem's complexity, and use it later to provide refined upper-bounds. We provide a new understanding that the worst-case complexity of CBS depends more on the graph's radius rather than on the graph's size. Another such observation is reflected in the incorporation of positive constraints in our recurrence-based analysis. It focuses on the importance of positive constraints by demonstrating the huge reduction in the search tree's size.

We hope that these observations would allow, in addition to improving the theoretical analysis of CBS, to be used to improve the algorithm in practice. One possible way to approach this task is to use these observations to develop heuristics. For instance, by favoring expansion of CT nodes with a large number of positive constraints or with a minimal amount of projected work that remains to be done by the algorithm.

Extended Complexity Analysis

The analysis techniques presented in this work utilize several observations regarding the MAPF problem in order to improve the complexity bound on CBS. However, there are additional aspects of the problem, as well as additional improvements of CBS, that may be incorporated into the analysis and would allow to further improve the upper-bounds.

For instance, tools such as prioritizing conflicts (Boyarski et al., 2015) may affect the maximal size of an MDD graph. They may also allow to further refine the recurrence relation such that it would capture the progress of CBS computation more accurately. Symmetry-breaking (Li et al., 2019c, 2020a) and different heuristics for the high-level search of CBS are additional improvements that could to be incorporated into the analysis framework that we presented.

Another aspect of this work that can be extended is the focus on CBS for the purpose of hardness analysis. As mentioned in Section 1, there are other successful approaches for dealing with the MAPF problem, both search-based (that extend CBS or are similar to it in the main idea) and compilation-based (such as SAT-based solvers). Applying the same analysis, even on algorithms that are similar to CBS (such as ID-CBS and ICTS), is not straightforward, and would require adjustments to the analysis framework. Nevertheless, we believe that the conceptual and mathematical tools that we presented for the analysis of CBS could be used to analyze the complexity of other

MAPF algorithms, which in turn, might allow to further improve the understanding of aspects that make the problem difficult.

Appendix A

Generating Functions Based Analysis for Recurrence Asymptotic Approximation

This appendix contains a detailed explanation regarding the technique for bounding the recurrence relation presented in Lemma 4.1.1. It includes a large part of the mathematical tools described by Pemantle and Wilson (2008) which are relevant to the specific analysis that was preformed in this work.

It is important to emphasize, as mentioned in Section 4.3, that this analysis does not provide a formal proof for the resulting upper bound, but rather helps to deduce potential candidates for possible tight upper-bound for the recurrence relation. In our work, we examined the resulting bound against the values of the recurrence, and found out that it does constitute an empirical upper-bound.

The Recurrence Relation Generating Function

Equation 4.1 presents a recurrence relation $T(r,s)$ which forms an upper-bound on the complexity of the high-level search of the CBS algorithm. In order to get an estimated upper-bound on T, we solve the recurrence corresponding to the case in which all inequalities in Equation 4.1 are tight.

The first step in our analysis requires finding the generating function of $T(r,s)$. This means that $T(r,s)$ is the coefficient of $x^r y^s$ in F:

$$F(x,y) = \sum_{r,s \geq 0} T(r,s)\, x^r y^s. \tag{A.1}$$

Obtaining the exact form of $F(x,y)$ is done using the following equation (for further reading about finding generating functions for recurrences with multiple variables, we

refer the readers to Chapter 1.5 in (Wilf, 2006)):

$$\sum_{r,s\geq 0} T\left(r+2,s+1\right)x^r y^s = \sum_{r,s\geq 0} T\left(r+1,s+1\right)x^r y^s + \sum_{r,s\geq 0} T\left(r,s\right)x^r y^s + \sum_{r,s\geq 0} x^r y^s.$$

The computation involves algebraic simplification and substitution of the sums with $F\left(x,y\right)$ according to Equation A.1:

$$\frac{1}{x^2 y}\left(F\left(x,y\right)-1-\frac{y}{1-y}-\frac{x}{1-x}-\frac{3xy}{1-y}\right) = F\left(x,y\right)+\frac{1}{xy}\cdot\left(F\left(x,y\right)-1-\frac{y}{1-y}-\frac{x}{1-x}\right)+\frac{1}{\left(1-x\right)\left(1-y\right)},$$

which, eventually gives the following generating function for the recurrence:

$$F\left(x,y\right)=\frac{1-x+2xy-x^2 y}{(1-x)(1-y)(1-x-x^2 y)}. \tag{A.2}$$

Approximation Analysis Method

Critical Points

In order to get a closed-form formula which approximates the value of $T\left(r,s\right)$, we follow the analysis by Pemantle and Wilson (2008). As stated in Section 4.3, this method does not allow to prove the bound's correctness for a recurrence relation of the form that we have, but it does allow us to obtain an estimated bound, which we empirically evaluate to be precise.

The first step is to express the function as a ratio $F = G/H$, that is:

$$G\left(x,y\right)=1-x+2xy-x^2 y,$$
$$H\left(x,y\right)=(1-x)(1-y)(1-x-x^2 y).$$

We use $H_x, H_y, H_{xx}, H_{yy}, H_{xy}$ for the partial derivatives of H with respect to the sub-

scripted variables:

$$\begin{aligned} H_x &= (1-y)(3x^2y - 2x(y-1) - 2), \\ H_y &= (1-x)(x^2(2y-1) + x - 1), \\ H_{xx} &= -2(y-1)((3x-1)y + 1), \\ H_{yy} &= -2x^2(x-1), \\ H_{xy} &= x^2(3-6y) + 4x(y-1) + 2. \end{aligned} \tag{A.3}$$

We need to find the critical points which are given by the solutions of the following system in the positive quadrant $(x, y > 0)$:

$$\begin{cases} H = 0 \\ sxH_x = ryH_y. \end{cases}$$

There are three solutions of the system in the positive quadrant:

$$\begin{aligned} (x_1, y_1) &= \left(\frac{-1+\sqrt{5}}{2}, 1 \right) \\ (x_2, y_2) &= (1, 1) \\ (x_3, y_3) &= \left(\frac{r-2s}{r-s}, \frac{s(r-s)}{(r-2s)^2} \right). \end{aligned} \tag{A.4}$$

The third solution is missing when $r = s$ or $r = 2s$. Suppose that $m < r/s < M$ for $m > 0, M < \infty$. Then according to the work of Pemantle and Wilson (2008), each critical point contributes some asymptotic factor to the possible upper bound of the recursion. We denote the matching contribution factor of each point (x_i, y_i) by $T_i(r, s)$.

The contribution by each point is given by a different formula, depending on the point's multiplicity.

Let $\mathcal{H} = H_{xx}H_{yy} - H_{xy}$. Then the contribution T_i of a *multiple point* (x_i, y_i) is given by:

$$T_i(r, s) = x_i^{-r} y_i^{-s} \frac{G(x_i, y_i)}{\sqrt{-x_i^2 y_i^2 \mathcal{H}(x_i, y_i)}}. \tag{A.5}$$

Let:

$$\begin{aligned} Q(x, y) = &- xH_x y^2 H_y^2 - yH_y x^2 H_x^2 - y^2 H_y^2 x^2 H_{xx} \\ &- x^2 H_x^2 y^2 H_{yy} + 2xH_x yH_y xyH_{xy}, \end{aligned} \tag{A.6}$$

Then the contribution T_i of a *single point* (x_i, y_i) is given by:

$$T_i(r, s) = \frac{G(x_i, y_i)}{\sqrt{2\pi}} x_i^{-r} y_i^{-s} \sqrt{\frac{-y_i H_y(x_i, y_i)}{sQ(x_i, y_i)}}. \tag{A.7}$$

When H doesn't contain quadratic factors, a point is considered "single" if the value

of the gradient of H at the point is non-zero. Otherwise, it is considered "multiple".

Multiple Points' Contribution

The points (x_1, y_1) and (x_2, y_2) from Equation A.4 are both multiple points. The corresponding values of $\mathcal{H}$ and G functions for those points are:

$$\begin{aligned}\mathcal{H}(x_1, y_1) &= \frac{15\sqrt{5}}{2} - \frac{35}{2}, G(x_1, y_1) = \sqrt{5} - 1,\\ \mathcal{H}(x_2, y_2) &= -1, G(x_2, y_2) = 1.\end{aligned}$$

The points' contributions, given by substituting the aforementioned values in Equation A.5, are:

$$\begin{aligned}T_1(r, s) &\sim \left(\frac{4}{3\sqrt{5} - 5}\right) \cdot \left(\frac{1 + \sqrt{5}}{2}\right)^r\\ T_2(r, s) &= \frac{G(1, 1)}{\sqrt{-\mathcal{H}}} = 1.\end{aligned} \tag{A.8}$$

Single Points' Contribution

The point (x_3, y_3) from Equation 4.3 is a single point. According to Equation A.6 and Equation A.7, its contribution is:

$$T_3(r, s) = \frac{(r - s)^{r-s}}{(r - 2s)^{r-2s} \cdot s^s} \cdot \frac{2s}{r - 2s} \cdot \sqrt{\frac{\alpha}{2\pi}}, \tag{A.9}$$

where $\alpha = \mathcal{O}(\frac{r^2}{s})$.

Obtained Recurrence Approximation

In Section 4.3, we presented the motivation for computing a bound for the specific case where there is a linear dependence between r and s, namely, $r = ns$ for a given $n \in \mathbb{N}$. For this case, by simply substituting r and reducing the fractions, we get the following factors for each critical point:

$$\begin{aligned}T_1(ns, s) &= 1,\\ T_2(ns, s) &= \left(\frac{4}{3\sqrt{5} - 5}\right) \cdot \left(\frac{1 + \sqrt{5}}{2}\right)^{ns},\\ T_3(ns, s) &= \left(\frac{(n - 1)^{n-1}}{(n - 2)^{n-2}}\right)^s \cdot \frac{2}{n - 2} \cdot \sqrt{\frac{\beta}{2\pi s}},\end{aligned} \tag{A.10}$$

where $\beta = \mathcal{O}(n^2)$.

We care about finding the component in Equation A.10 which closely approximates the value of $T(ns, s)$. First, observe that $\frac{2}{n-2} \cdot \sqrt{\frac{\beta}{2\pi}}$ is asymptotically $\mathcal{O}(1)$. Now, let n_0

be the solution of

$$\frac{(n-1)^{n-1}}{(n-2)^{n-2}} = \left(\frac{1+\sqrt{5}}{2}\right)^n,$$

which is $n_0 = \frac{\sqrt{5}+2}{2} \approx 3.618033$. Note that we intentionally omit the multiplication by $1/\sqrt{s}$, which is insignificant asymptotically in this case.

Each of the critical points' contribution is used to approximate the recursion value for a certain range of values of n. Empirically, we get that if $n < n_0$ then the contribution of the multiple points gives a tight approximation:

$$T(ns, s) \sim 1 + \left(\frac{4}{3\sqrt{5}-5}\right) \cdot \left(\frac{1+\sqrt{5}}{2}\right)^{ns},$$

which is the same, asymptotically, as the ns'th Fibonacci number.

If $n \geq n_0$ then the approximation is given by the contribution of the single point, therefore:

$$T(ns, s) \sim \left(\frac{(n-1)^{n-1}}{(n-2)^{n-2}}\right)^s \cdot \frac{1}{\sqrt{s}},$$

where we assume that $\sim$ ignores a constant factor within the approximated expression.

We are interested in the asymptotic behavior of this approximation, therefore we focus on the formula obtained for $n \geq n_0$. We notice that $\frac{(n-1)^{n-1}}{(n-2)^{n-2}} < en$, thus, in conclusion we get the following upper bound on T:

$$T(ns, s) \sim \frac{(en)^s}{\sqrt{s}}. \tag{A.11}$$

Bibliography

Banfi, J., Basilico, N., and Amigoni, F. (2017). Intractability of Time-Optimal Multi-robot Path Planning on 2D Grid Graphs with Holes. *IEEE Robotics and Automation Letters*, 2:1941–1947.

Barer, M., Sharon, G., Stern, R., and Felner, A. (2014). Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. *Symp. on Combinatorial Search, SoCS*, pages 19–27.

Barták, R., Ivanová, M., and Švancara, J. (2021). From classical to colored multi-agent path finding. *Symp. on Combinatorial Search, SoCS*, pages 150–152.

Boyarski, E., Felner, A., Harabor, D., Stuckey, P. J., Cohen, L., Li, J., and Koenig, S. (2020). Iterative-deepening conflict-based search. In Bessiere, C., editor, *Int. Joint Conf. on Artificial Intelligence, IJCAI*, pages 4084–4090.

Boyarski, E., Felner, A., Le Bodic, P., Harabor, D. D., Stuckey, P. J., and Koenig, S. (2021). f-aware conflict prioritization & improved heuristics for conflict-based search. *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 12241–12248.

Boyarski, E., Felner, A., Stern, R., Sharon, G., Tolpin, D., Betzalel, O., and Shimony, E. (2015). ICBS: Improved conflict-based search algorithm for multi-agent pathfinding. *Int. Joint Conf. on Artificial Intelligence, IJCAI*, pages 740–746.

Cohen, L., Uras, T., Kumar, T. K. S., Xu, H., Ayanian, N., and Koenig, S. (2016). Improved solvers for bounded-suboptimal multi-agent path finding. In Kambhampati, S., editor, *Int. Joint Conf. on Artificial Intelligence, IJCAI*, pages 3067–3074.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, page 151–158. Association for Computing Machinery.

Felner, A., Li, J., Boyarski, E., Ma, H., Cohen, L., Kumar, T. K., and Koenig, S. (2018). Adding heuristics to conflict-based search for multi-agent path finding. In *International Conference on Automated Planning and Scheduling, ICAPS*, pages 83–87. AAAI press.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. on Systems Science and Cybernetics*, 4:100–107.

Kaduri, O., Boyarski, E., and Stern, R. (2020). Algorithm selection for optimal multi-agent pathfinding. In *International Conference on Automated Planning and Scheduling, ICAPS*, pages 161–165. AAAI press.

Kaduri, O., Boyarski, E., and Stern, R. (2021). Experimental evaluation of classical multi agent path finding algorithms. *Symp. on Combinatorial Search, SoCS*, pages 126–130.

Kilani, Y., Bsoul, M., Alsarhan, A., and Al-Khasawneh, A. (2013). A survey of the satisfiability-problems solving algorithms. *International Journal of Advanced Intelligence Paradigms*, 5:233–256.

Li, J., Felner, A., Boyarski, E., Ma, H., and Koenig, S. (2019a). Improved heuristics for multi-agent path finding with conflict-based search. *Int. Joint Conf. on Artificial Intelligence, IJCAI*, pages 442–449.

Li, J., Gange, G., Harabor, D., Stuckey, P. J., Ma, H., and Koenig, S. (2020a). New techniques for pairwise symmetry breaking in multi-agent path finding. *International Conference on Automated Planning and Scheduling, ICAPS*, 30:193–201.

Li, J., Harabor, D., Stuckey, P. J., Felner, A., Ma, H., and Koenig, S. (2019b). Disjoint splitting for multi-agent path finding with conflict-based search. In *International Conference on Automated Planning and Scheduling, ICAPS*, pages 279–283. AAAI press.

Li, J., Harabor, D., Stuckey, P. J., Ma, H., and Koenig, S. (2019c). Symmetry-breaking constraints for grid-based multi-agent path finding. *Symp. on Combinatorial Search, SoCS*, pages 184–185.

Li, J., Ruml, W., and Koenig, S. (2020b). EECBS: A bounded-suboptimal search for multi-agent path finding. *CoRR*, abs/2010.01367.

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

Pallottino, L., Scordio, V. G., Bicchi, A., and Frazzoli, E. (2007). Decentralized cooperative policy for conflict resolution in multivehicle systems. *IEEE Trans. Robotics*, 23(6):1170–1183.

Pemantle, R. and Wilson, M. C. (2008). Twenty combinatorial examples of asymptotics derived from multivariate generating functions. *SIAM Journal on Computing*, 50(2):199–272.

Salzman, O. and Stern, R. (2020). Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, pages 1711–1715. International Foundation for Autonomous Agents and Multiagent Systems.

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

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

Silver, D. (2005). Cooperative pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, page 117–122. AAAI Press.

Stern, R., Sturtevant, N. R., Felner, A., Koenig, S., Ma, H., Walker, T. T., Li, J., Atzmon, D., Cohen, L., Kumar, T. K. S., Boyarski, E., and Bartak, R. (2019). Multi-agent pathfinding: Definitions, variants, and benchmarks. *Symp. on Combinatorial Search, SoCS*, pages 151–158.

Sturtevant, N. (2012). Benchmarks for grid-based pathfinding. *IEEE Trans. on Computational Intelligence and AI in Games*, 4:144–148.

Surynek, P. (2012). Towards optimal cooperative path planning in hard setups through satisfiability solving. In Anthony, P., Ishizuka, M., and Lukose, D., editors, *PRICAI 2012: Trends in Artificial Intelligence - 12th Pacific Rim*, volume 7458 of *Lecture Notes in Computer Science*, pages 564–576. Springer.

Surynek, P. (2014). Compact representations of cooperative path-finding as SAT based on matchings in bipartite graphs. In *IEEE International Conference on Tools with Artificial Intelligence, ICTAI*, pages 875–882. IEEE Computer Society.

Surynek, P. (2021). Multi-goal multi-agent path finding via decoupled and integrated goal vertex ordering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 12409–12417. AAAI Press.

Surynek, P., Felner, A., Stern, R., and Boyarski, E. (2016). Efficient sat approach to multi-agent path finding under the sum of costs objective. In *Proceedings of the Twenty-Second European Conference on Artificial Intelligence*, page 810–818. IOS Press.

Surynek, P., Felner, A., Stern, R., and Boyarski, E. (2017a). Modifying optimal sat-based approach to multi-agent path-finding problem to suboptimal variants. In Fukunaga, A. and Kishimoto, A., editors, *Symp. on Combinatorial Search, SoCS*, pages 169–170. AAAI Press.

Surynek, P., Svancara, J., Felner, A., and Boyarski, E. (2017b). Integration of independence detection into sat-based optimal multi-agent path finding - A novel sat-based optimal MAPF solver. In van den Herik, H. J., Rocha, A. P., and Filipe, J., editors, *Proceedings of the 9th International Conference on Agents and Artificial*, pages 85–95. SciTePress.

Walker, T. T., Sturtevant, N. R., Felner, A., Zhang, H., Li, J., and Kumar, T. K. S. (2021). Conflict-based increasing cost search. *International Conference on Automated Planning and Scheduling, ICAPS*, pages 385–395.

Wilf, H. S. (2006). *Generatingfunctionology*. A. K. Peters, Ltd.

Wurman, P. R., D'Andrea, R., and Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *Artificial Intelligence*, 29(1):9–20.

Yu, J. (2016). Intractability of optimal multirobot path planning on planar graphs. *IEEE Robotics and Automation Letters*, 1:33–40.

Zhang, H., Li, J., Surynek, P., Koenig, S., and Satish Kumar, T. K. (2020). Multi-agent path finding with mutex propagation. In *International Conference on Automated Planning and Scheduling, ICAPS*, pages 323–332. AAAI press.

חשיבות גבוהה ליכולת של האלגוריתמים השונים של בעית MAPF למצוא פיתרון באופן יעיל, ולהיות מסוגלים להתמודד עם מגוון רחב של תרחישים.

נקודת המוצא לעבודה זו הינה הניתוח הקיים לחסם הסיבוכיות של אלגוריתם CBS, עבור המקרה הגרוע ביותר. בעבודה זו אנו מתבססים על האנליזה הקיימת, ומשפרים אותה באמצעות הפעלת כלים מתמטיים שונים ואבחנות חדשות אודות הגורמים המשפיעים על הקושי של הבעיה והיכולת של CBS להתמודד איתה באופן יעיל. האנליזה המוצגת בעבודה מבוססת על שתי שיטות משלימות: בשיטה הראשונה, אנו חוסמים את זמן הריצה של האלגוריתם באמצעות ניתוח הגודל המקסימלי של מבנה נתונים ייחודי הנקרא Multi-valued Decision Diagram (MDD) – גרף שכבות אשר מכיל ייצוג קומפקטי של כלל המסלולים האפשריים עבור סוכן בין שתי נקודות על הגרף, עבור אורך מסלול נתון. החסם על גודל גרף ה-MDD מאפשר לנו לשפר את החסם על גודל מרחב החיפוש של אלגוריתם CBS במקרה הגרוע.

בשיטה השניה, אנו מביעים חסם על סיבוכיות זמן הריצה של האלגוריתם באמצעות נוסחה רקורסיבית חדשה. אנחנו משתמשים בשיטת אנליזה המבוססת על פונקציות יוצרות בכדי לחסום באופן הדוק את הנוסחה הרקורסיבית.

באמצעות השיטות הללו אנחנו מציגים מספר חסמים עליונים חדשים ומשופרים עבור סיבוכיות זמן הריצה של אלגוריתם CBS. התוצאות מאפשרות לנו להדק את החסם הידוע על האלגוריתם עבור מגוון רחב מאוד של תרחישים. לדוגמא, על אוסף רחב של תרחישי-מבחן סטנדרטיים (benchmarks), אנו רואים שיפור של החסם העליון בפקטור של לפחות 2^{10^6}, לעומת החסם המוכר. בנוסף, אנו מציגים פרק מורחב של דיון בתוצאות ובצעדי המשך מחקריים אפשריים, אשר אנו מקווים כי יאפשרו להרחיב את כיוון המחקר הנ"ל, לקראת שיפור ההבנה אודות הגורמים המשפיעים על הקושי של בעית MAPF.

תקציר

בבעית "תכנון מסלול מרובת סוכנים" (MAPF) נדרש למצוא אוסף מסלולים עבור קבוצת סוכנים הנעים בסביבה משותפת נתונה, כך שהמסלולים אינם מתנגשים אחד עם השני. לבעיה זו עניין הולך וגובר בשנים האחרונות בקרב קהילת המחקר בתחומי הבינה המלאכותית והרובוטיקה, עקב היישומים הרבים מהעולם האמיתי אשר ניתנים למידול בהסתמך על התיאוריה מאחורי בעית MAPF.

על אף שתכנון מסלול עבור סוכן יחיד זו בעיה אשר ניתנת לפיתרון יעיל, מציאת פיתרון אופטימלי עבור בעית MAPF הינה משימה קשה חישובית. עם זאת, קיימות מספר גישות אלגוריתמיות אשר מאפשרות להתמודד עם תרחישים לא טריוויאליים של הבעיה באופן אפקטיבי.

אחד מהאלגוריתמים המובילים, אשר משמש למציאת פיתרון אופטימלי עבור הבעיה, הינו אלגוריתם Conflict-Based Search (CBS). אלגוריתם זה מתמודד עם מגוון רחב של תרחישים של הבעיה בפועל באופן יעיל, אך עם זאת, ישנם תרחישים רבים שבהם לא יצליח למצוא פיתרון אופטימלי, אפילו תחת זמן-ריצה ארוך מאוד. מעניין לראות שהתרחישים בהם האלגוריתם מתקשה אינם בעליי מאפיינים ייחודיים כלשהם אשר מבדילים אותם מתרחישים אחרים איתם האלגוריתם מצליח להתמודד ביעילות רבה. כלומר, ישנו פער גדול בהבנה שלנו אודות הגורמים המשפיעים על הקושי החישובי של בעית MAPF ועל היכולת של האלגוריתמים השונים להתמודד איתה.

בעבודה זו אנו מתייחסים לבעיה זו ומנסים לשפוך אור על הגורמים המרכזיים שמשפיעים על הקושי של הבעיה. אנחנו עושים זאת תוך בחינה מחודשת של ניתוח סיבוכיות זמן הריצה של אלגוריתם CBS, במטרה להדק את החסם התיאורתי על זמן הריצה של האלגוריתם במקרה הגרוע.

אחת מהבעיות המרכזיות מהן אנו שואבים מוטיבציה לעבודה זו הינה הבעיה של ניהול מחסנים אוטונומי. בבעיה זו ישנו מחסן המורכב ממדפים המאחסנים מוצרים ואוסף של רובוטים אשר מסוגלים לנוע במרחב ולהרים מדפים/מוצרים ולהוביל אותם לנקודות אריזה ושילוח. הרובוטים נעים במקביל במחסן מנקודות המוצא, כל אחד אל המדף אותו הוא צריך לאסוף ומשם לנקודות הפריקה. נדרש לתכנן מסלול לכל אחד מהרובוטים כך שאינם יתנגשו ויבצעו את המשימות באופן כמה שיותר אופטימלי (מבחינת זמן הביצוע האינדיבידואלי, הכולל, מרחק הנסיעה וכד'). בעית MAPF מאפשרת למדל את משימת ניהול המחסן ופתרונות עבור המודל המתמטי של בעית MAPF יכולים לשמש, תוך התאמות למגבלות שבתנועה בעולם האמיתי, עבור ניהול מחסנים באופן אוטונומי.

ישנם יישומים רבים נוספים אשר ניתנים למידול מלא/חלקי באמצעות בעית MAPF, כגון משחקי מחשב מרובי משתתפים, ניהול תנועה של רכבים אוטונומיים ואפליקציות נוספות מבוססות רובוטים. לכלל היישומים הללו נדרש למצוא פיתרון אופטימלי ככל שניתן לבעיה בזמן חישוב מוגבל, ולכן ישנה

המחקר בוצע בהנחייתו של דוקטור אורן זלצמן, בפקולטה למדעי המחשב.

חלק מן התוצאות בחיבור זה פורסמו כמאמר מאת המחבר ושותפיו למחקר בכנס במהלך תקופת מחקר המגיסטר של המחבר:

Gordon, O., Filmus, Y., and Salzman, O. (2021). Revisiting the complexity analysis of conflict-based search: New computational techniques and improved bounds. In *Proceedings of the Fourteenth International Symposium on Combinatorial Search, SOCS 2021, Virtual Conference [Jinan, China], July 26-30, 2021*, pages 64–72. AAAI Press.

תודות

בראש ובראשונה, אני רוצה להודות למנחה שלי, דוקטור אורן זלצמן, על היותו מנחה ומנטור מצוין, על ההשראה שנתן לי לכל אורך הלימודים והמחקר, ועל שעזר להנחות ולהדריך אותי בסבלנות ועם תשוקה רבה לעולם המחקר.

בנוסף, ברצוני להודות לפרופסור יובל פילמוס מהפקולטה למדעי המחשב על תובנותיו המועילות שתרמו רבות למחקר.

ברצוני להודות גם כן לחברים מהמעבדה לרובוטיקה חישובית על הדיונים ושיתוף הפעולה המועילים, ולאשתי האהובה שעומדת לצידי בכל צעד בדרך.

לבסוף, ברצוני להקדיש עבודה זו להוריי - כל מה שהשגתי הוא בזכותם.

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי.

לקראת הבנת הקושי של בעיית תכנון מסלול מרובת סוכנים

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים במדעי המחשב

אופיר גורדון

הוגש לסנט הטכניון — מכון טכנולוגי לישראל
תשרי התשפ"ב חיפה אוקטובר 2021

לקראת הבנת הקושי של בעיית תכנון מסלול מרובת סוכנים

אופיר גורדון