FLEX DISTRIBUTION FOR BOUNDED-SUBOPTIMAL MULTI-AGENT PATH FINDING

by

Shao-Hung Chan

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

May 2026

Dedication

To my parents.

Acknowledgements

The journey to a Ph.D. is challenging. Luckily, I received a lot of support from my advisor, colleagues, friends, and family.

First of all, I would like to express my profound gratitude to my advisor, Sven Koenig. With the funding he provided, I had the opportunity to pursue my Ph.D. and dedicate myself to research, even during the COVID-19 pandemic. I also benefited from diverse international experiences, including an exchange program in Israel, an internship in Germany, and conferences in the Czech Republic and Great Britain. Under his mentorship, I learned that research is not only about ideas and experiments; it is about communication and approaching every publication or presentation with a perfectionist mindset. My Ph.D. journey would not have been as rewarding and meaningful without his patient guidance.

I want to thank my dissertation committee, namely T. K. Satish Kumar, Gaurav Sukhatme, John Carlsson, and Ariel Felner, for their time and feedback. I especially want to thank Ariel Felner for attending my dissertation defense at midnight his time. I also want to thank Lars Lindemann for his feedback on the proposal for this dissertation, and Lisa Lee, Raymond Duran, and Aaron Thompson for helping me with the graduation process.

At the University of Southern California (USC), I am so grateful to work with talented colleagues in our research group. I want to thank Jiaoyang Li for her collaboration on developing the theorems and implementation of Flex Distribution. Our discussions have enlightened my interest in Multi-Agent Path Finding. Next, I want to thank Thomy Phan for enhancing the clarity of my writing and the structure of

my ideas on Flex Distribution. I also want to thank Liron Cohen for his idea of using different bounded-suboptimality factors to find paths for agents, which served as the starting point for Flex Distribution. Additionally, I want to thank my colleagues in our research group, namely Tansel Uras, Wolfgang Hoenig, Hang Ma, Taoan Huang, Han Zhang, Yi Zheng, Weizhe Chen, Christopher Leet, Yimin Tang, and Joonyeol Sim. The discussion in each group seminar has a strong and positive impact on this dissertation.

I want to thank my collaborators at Monash University, Australia. I especially want to thank Daniel Harabor and Peter J. Stuckey for their input that solidifies the idea of Greedy Flex Distribution. I want to thank Graeme Gange for helping with the Flex Distribution paper. I also want to thank Zhe Chen, Yue Zhang, Teng Guo, and Han Zhang for the joint organization of the League of Robot Runners, a Multi-Agent Path Finding competition supervised by Daniel Harabor, Jingjin Yu, Cathy Wu, Sven Koenig, and Federico Pecora. The visualizer developed in this competition serves as an excellent tool for demonstrating the effects of Flex Distribution and the Flow-Based Guidance Heuristic.

At Ben-Gurion University of the Negev in Israel, I want to thank Ariel Felner for hosting me with the scholarship, which allowed me to work on various research topics. In addition to doing research with Ariel, I want to thank Roni Stern, Eli Boyarski, and Dor Atzmon for the collaboration on improving Priority-Based Search, and Shawn Skyler, Han Zhang, Oren Salzman, Shahaf Shperberg, Carlos Hernandez Ulloa, Jorge Baier, and William Yeoh for the collaboration on developing multi-objective heuristic search algorithms. All these collaborations have helped me develop more mature ideas on Flex Distribution.

During my internship at Bosch in Germany, I want to thank Ralph Lange, Ulli Hoffmann, Isabelle Barz, Ingo Lutkebohle, Philipp Schillinger, Florian Lier, Damian Krata, Sebastian Haug, Mirco Colosi, and Chuanlong Zang for collaborating on the multi-agent system that brings insights to this dissertation.

During my Ph.D. journey, I want to thank Nathan Sturtevant, Maxim Likhachev, Stefanos Nikolaidis, Tsung-Wei Huang, Rishi Veerapaneni, Tiannan Zhang, Hejia Zhang, Benran Zhang, Daniel Koyfman, Lior

Siag, Jonathan Morag, Avi Natan, Argaman Mordoch, Yarin Benyamin, Ta-Yang Wang, Chee-An Yu, Dian-Lun Lin, Loc Trinh, Caroline Johnston, Yusuf Hakan Kalayci, Nathan Dennler, Sepanta Zeighami, Sojhal Bloach, Johnny Tian-Zheng Wei, Connie Zhang, Andrey Shilo, Julia Balukonis, Dylan Black, Calvin Leng, Kicho Yu, Karen Pan, Te-Yi Kan, Yuan-Hsin Shih, Yun-Cheng Wang, Tsung-Shan Yang, Jie-En Yao, Hong-En Chen, Han-Ting Liao, Ting-Hao Hsu, Sing-Yao Wu, Bo-Han Kung, Miryam Huang, Rabbi Dov and Runya Wagner, Daniel Marshall, Ban Katz, Amelia Marvit, Ariel Urman, Zhivar Sourati, Daniel Hofstadt, Shu-Yin Tung, Yunfan Gao, Yufei Zhu, Shih-Min Yang, Mai Chatani, Jialei Li, Lukas Heuer, Sebastian Koch, Yuchen Liu, Niels van Duijkeren, Andrey Rudenko, Narunas Vaskevicius, Roman Bartak, Jiri Svancara, De Jane Chou, Fisher Kuo, I-ming Chen, and other friends and colleagues for discussions on various research topics and sharing joyful moments, which have helped me when writing this dissertation.

I am especially grateful to my parents, Chao-Chin Chan and I-Chen Cheng, for their unwavering love and encouragement that have helped me overcome the challenges of my research. I also want to thank Anne Han, Andrew Han, Tina Walsh, Rei-Lin Cheng, Dun-Dai Yin, Rei-Yu Cheng, Chung-Yan Yang, Hai-Nan Chan, Shu-Yue Chan Lin, and other family members for their support.

Last but not least, I want to especially express my deepest appreciation to my wife, Aglaia Iankovskaia, for enduring all the ups and downs with me. Her smile and homemade borscht brought warmth to my Ph.D journey. This dissertation would not have been possible without her.

The research was supported by the National Science Foundation (NSF) under some of the grants 1409987, 1724392, 1817189, 1837779, 1935712, 2112533, 2121028, 2321786, 2346058, and 2434916, as well as a gift from Amazon Robotics.

Table of Contents

List of Tables

List of Figures

Glossary

MAPF	Multi-Agent Path Finding. See Chapter 1 and Section 2.2.
MAPD	Multi-Agent Pickup and Delivery. See Section 1.1.1.
MAPF-LNS	Large Neighborhood Search for (Anytime) MAPF. See Section 1.1.2.
SOC	Sum of (path) Costs. See Section 2.2.
CT	Constraint Tree. See Section 2.3.
CBS	Conflict-Based Search. See Section 2.3.
PBS	Priority-Based Search. See Section 2.3.
EECBS	Explicit Estimation Conflict-Based Search. See Section 2.4.
EES	Explicit Estimation Search. See Section 2.4.1.
SOLB	Sum of Lower Bounds. See Section 2.4.1.
WDG	Weighted Dependency Graph. See Section 2.4.3.4.
GFD	Greedy Flex Distribution. See Section 3.6.
GBS	Globally Bounded-Suboptimal. See Section 4.1
LBI	Relative Lower Bound Improvement. See Equation 4.7.
FFD	Fixed-Fractional Flex Distribution. See Section 4.2.
RFD	Random-Fractional Flex Distribution. See Section 4.2.
CFD	Conflict-Based Flex Distribution. See Section 4.3.
DFD	Delay-Based Flex Distribution. See Section 4.4.
MFD	Mixed-Strategy Flex Distribution. See Section 4.5.
FA*	Focal-A*. See Section 4.6.
CC	Criss-Cross. See Section 5.1.1.
GM	Graph Model. See Section 5.1.1.
HM	Heat Map. See Section 5.1.1.
TFO	Traffic Flow Optimization. See Section 5.1.2.
SUO	Space Utililty Optimization. See Section 5.1.2.
FBGF	Flow-Based Guidance Framework. See Section 5.3.
P1	Path-Simulation Phase. See Section 5.3.1.
BCS	Bounded-Cost Search. See Section 5.3.1.
FBCS	Flexible Bounded-Cost Search. See Section 5.3.1.
FGG	Flow-Based Guidance Graph. See Section 5.3.1.
P2	Heuristic-Calculation Phase. See Section 5.3.2.
FH	Flow-Based Guidance Heuristic. See Section 5.3.2.
LaCAM	Lazy Constraints Addition Search for MAPF. See Section 5.4.4.

Notation

s_i	Start vertex of agent a_i
l_i	Target vertex of agent a_i
w	User-specified bounded-suboptimality factor.
$\Psi_i(N)$	Set of constraints for agent a_i in CT node N.
$\Psi(N) = [\, \Psi_i(N) \mid i \in [k] \,]$	
$p_i(N)$	Path from the start vertex s_i to the target vertex l_i of agent a_i that satisfies the constraints $\Psi_i(N)$.
$c_i(N)$	Cost of the path $p_i(N)$.
$c_i^*(N)$	Cost of the minimum-cost path from the start vertex s_i to the target vertex l_i of agent a_i that satisfies the constraints $\Psi_i(N)$.
$lb_i(N)$	Lower bound on the cost $c_i^*(N)$.
$C(N) = \sum_{i \in [k]} c_i(N)$	Sum of (path) costs (SOC) of CT node N.
$LB(N) = \sum_{i \in [k]} lb_i(N)$	Sum of lower bounds (SOLB) of CT node N.
$H(N)$	Cost-to-go function that never overestimates the difference between the minimum SOC over all solutions that satisfy the constraints $\Psi(N)$ and the SOLB $LB(N)$.
$F(N) = LB(N) + H(N)$	

N_F	CT node with the minimum F-value over all CT nodes in CLEANUP$_H$.
$\hat{H}(N)$	Cost-to-go function that estimates the difference between the minimum SOC over all solutions that satisfy the constraints $\Psi(N)$ and the SOC $C(N)$.
$\hat{F}(N) = C(N) + \hat{H}(N)$	
$N_{\hat{F}}$	CT node with the minimum $\hat{F}$-value over all CT nodes in OPEN$_H$.
$\mathcal{X}(N)$	Set of conflicts among the paths in CT node N.
$\lvert\mathcal{X}(N)\rvert$	Number of conflicts in $\mathcal{X}(N)$.
$N_{\mathcal{X}}$	CT node with the minimum number of conflicts in FOCAL$_H$.
C^*	SOC of an optimal solution.
LB	Lower bound on the SOC of an optimal solution.
$s_i(n) = (v, t)$	State representation of v-t node n, which indicates that agent a_i stays at vertex v at timestep t.
$g_i(n)$	Number of timesteps needed for agent a_i to move from its start vertex s_i to vertex v while following the path extracted from v-t node n with the state representation (v, t).
$h_i(n) = h_i(v)$	An admissible heuristic value that never overestimates the number of timesteps needed for agent a_i to move from vertex v to its target vertex l_i (suppose that the state representation of v-t node n is (v, t)).
$f_i(n) = g_i(n) + h_i(n)$	
$\tilde{h}_i(n) = \tilde{h}_i(v)$	Secondary heuristic of v-t node n (with the state representation (v, t)).
$\tilde{f}_i(n) = g_i(n) + \tilde{h}_i(v)$	

$x_i(n)$	Number of conflicts with the paths of the other agents when agent a_i moves from its start vertex s_i at timestep 0 to vertex v at timestep t, following the path extracted from v-t node n with the state representation (v, t).
parent(n)	Pointer to the parent of v-t node n.
$f_{\min,i}(N)$	Minimum f_i-value over all v-t nodes in OPEN$_L$ during the search of a path for agent a_i in CT node N.
$\tau_i(N)$	Threshold of the cost of the path for the low-level Focal Search during its search of a path for agent a_i in CT node N.
$\dot{P}_i(N)$	Set of all paths but $p_i(N)$ in CT node N.
$T_i(N)$	Earliest timestep when all agents but agent a_i begin to wait at their target vertices permanently and when there are no constraints in $\Psi_i(N)$ of CT node N.
$\Delta_{\max,i}(N)$	Maximum allowed flex during the search of a path for agent a_i in CT node N.
$\Delta_i(N)$	Distributed flex during the search of a path for agent a_i in CT node N.
$k_{\max}$	Path-found threshold.
c_p	Maximum penalty cost.
$h_{F,i}(v)$	FH value on a vertex v for agent a_i.

Abstract

Multi-Agent Path Finding (MAPF) is the problem of finding a list of collision-free paths, one for each agent, that move them from their respective start locations to their respective target locations in a shared environment. The objective of MAPF is to minimize the sum of (path) costs (SOC), where the cost of a path is defined as the travel time for an agent to move from its start location to its target location when it follows the path. MAPF has numerous applications in coordinating multiple agents, including automated warehouses, traffic management, quadrotor swarms, robotic multi-arm manipulation, and even the navigation of animated characters. However, solving MAPF optimally is NP-hard, which makes it a challenging problem. To trade off solution quality for runtime, researchers have been working on bounded-suboptimal MAPF, which aims to find a solution for a MAPF instance whose SOC is at most a user-specified bounded-suboptimality factor w larger than optimal. In other words, bounded-suboptimal MAPF strikes a balance between the runtime required to find a solution and the guarantee on its quality.

Explicit Estimation Conflict-Based Search (EECBS) is a state-of-the-art bounded-suboptimal MAPF algorithm. It is a two-level MAPF algorithm that is based on heuristic search. Its strategy is to iteratively find paths for agents on the low level and resolve collisions that occur between them on the high level. To resolve collisions, EECBS uses constraints to prevent agents from occupying locations at times that would result in collisions. To guarantee that the solution is bounded-suboptimal, EECBS requires the SOC of paths to be at most w larger than the SOC of the minimum-cost paths that satisfy the constraints. It achieves this by finding an individually bounded-suboptimal path for each agent. A path is individually

bounded-suboptimal iff its cost is at most a user-specified bounded-suboptimality factor w larger than the cost of a minimum-cost path that satisfies all constraints. To find an individually bounded-suboptimal path, EECBS runs Focal Search on its low level, which returns a lower bound on the cost of a minimum-cost path that satisfies the constraints and a path whose cost is at most w times the lower bound, which guarantees the path to be individually bounded-suboptimal. As each path is individually bounded-suboptimal, the SOC of each agent's path is at most w times the sum of their lower bounds, which guarantees the SOC of paths to be at most w larger than the SOC of the minimum-cost paths that satisfy the constraints (thereby guaranteeing the solution it finds is bounded-suboptimal).

Despite being the state-of-the-art bounded-suboptimal MAPF algorithm, EECBS maintains its guarantee of finding bounded-suboptimal solutions by finding individually bounded-suboptimal paths. However, the definition of bounded-suboptimal MAPF algorithms depends on the **sum** of path costs rather than the cost of each path. Thus, in this dissertation, we propose *Flex Distribution*, an approach that relaxes the requirement that the paths of agents be individually bounded-suboptimal while still guaranteeing that it finds bounded-suboptimal solutions. Given a list of paths, some of them can satisfy constraints with costs that are close to optimal. In this case, Flex Distribution allows the other paths to have costs that exceed w times their lower bounds, enabling these agents to avoid collisions. Although these paths are no longer individually bounded-suboptimal, Flex Distribution still guarantees that the SOC of each agent's path is at most w times the sum of their lower bounds, thereby guaranteeing EECBS to find a bounded-suboptimal solution.

We first propose *Greedy Flex Distribution*, which is an intuitive mechanism that allows EECBS to find paths with costs as high as possible, as long as the SOC of paths is at most w times the sum of their lower bounds. We also propose *Conflict-Based Flex Distribution*, which provides thresholds on costs based on the number of collisions between paths. In this case, EECBS can only find paths with costs at most the thresholds. We also propose *Delay-Based Flex Distribution*, which estimates the additional cost (i.e.,

delays) required for a path to satisfy the constraints and then determines its threshold accordingly. Furthermore, we propose *Mixed-Strategy Flex Distribution*, which combines Conflict-Based Flex Distribution and Delay-Based Flex Distribution in a hierarchical framework. We prove that EECBS, combined with our Flex Distribution mechanisms, is still guaranteed to find bounded-suboptimal solutions if such solutions exist. Empirically, we demonstrate that EECBS with our Flex Distribution mechanisms outperforms the one without Flex Distribution in terms of success rates within a 120-second runtime limit.

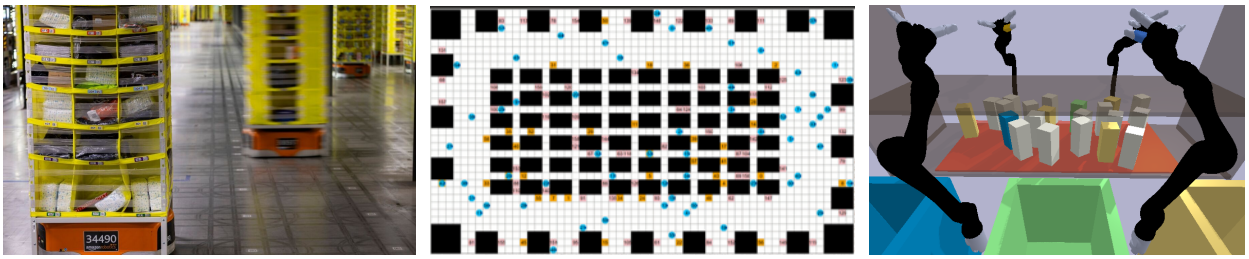
To improve the efficiency of EECBS, another line of research involves generating *guidance heuristics* that help reduce collisions when EECBS finds paths on its low level. In this dissertation, we combine Flex Distribution and guidance heuristics to further improve the efficiency of EECBS. In particular, we propose a two-phase guidance framework as a general approach for computing guidance heuristics. Then, we propose the *Flow-Based Guidance Framework*, which uses Flex Distribution when computing the *Flow-Based Guidance Heuristic*. That is, we apply Flex Distribution in both computing guidance heuristics and EECBS. Empirically, we demonstrate that EECBS with our Mixed-Strategy Flex Distribution becomes even more efficient when using our Flow-Based Guidance Heuristics. We also demonstrate that EECBS with our Mixed-Strategy Flex Distribution is more efficient when using our Flow-Based Guidance Heuristics than when using other state-of-the-art guidance heuristics.

Chapter 1

Introduction

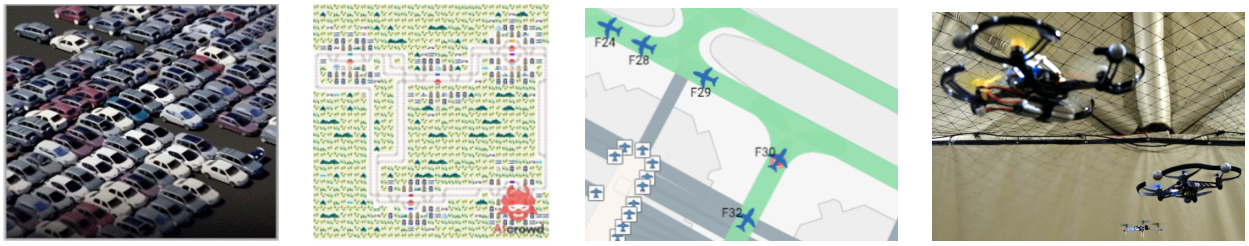
The emergence of autonomous multi-robot systems has transformed how we envision automation in complex and dynamic environments. By enabling a fleet of robots to cooperate, complex tasks can be accomplished at a level of scalability that far surpasses the capabilities of a single robot. A typical scenario involving multi-robot systems can be found in automated warehouses, where a fleet of mobile robots moves packages from one location to another, or multiple robot arms collaboratively stack and wrap packages. In this case, although the individual robots may have simple capabilities, their collective behavior can lead to sophisticated and efficient operations that boost productivity.

The core of coordinating multiple robots, or, more generally, multiple agents, is *Multi-Agent Path Finding* (MAPF), which is the problem of finding a list of collision-free paths, one for each agent, to move from their respective start locations to their respective target locations in a shared environment, while achieving some collaborative objectives. Many real-world applications can be modeled as MAPF. For example, MAPF can be applied to automated warehouses, as shown in Figure 1.1. In an automated warehouse, multiple mobile robots need to coordinate their paths to avoid collisions while optimizing their overall efficiency in delivering packages [6, 23, 66]. On the other hand, multiple robot arms need to coordinate their movements to avoid collisions while collaboratively assembling complex spatial structures or arranging objects [11, 73]. Aside from automated warehouses, MAPF can also be applied to traffic management, as shown in



(a) Amazon fulfillment center (b) Simulated scenario (c) Robot-arm operation

Figure 1.1: MAPF applications in automated warehouses: (a) a real-world Amazon fulfillment center (photo credit: [17]), (b) a simulated scenario for a warehouse robot planning competition (photo credit: [6]), and (c) robot-arm operation (photo credit: [73]).



(a) Automated garage (b) Train scheduling (c) Aircraft scheduling (d) Quadrotor swarm

Figure 1.2: MAPF applications, namely, (a) parking in automated garages (photo credit: [19]), (b) train scheduling with uncertain delays (photo credit: [29, 26]), (c) aircraft scheduling along airport taxiways (photo credit: [38]), and (d) quadrotor swarm operations (photo credit: [16]).

Figure 1.2. For example, MAPF can be applied to coordinating several connected and autonomous vehicles when traversing traffic intersections [33] or parking in automated garages [19]. MAPF can also be applied to traffic management of trains with different lengths [13, 29], multiple aircraft moving along taxiways in a large airport [38, 58], and quadrotor swarms flying in an obstacle-rich environment [22, 24, 46]. MAPF can even be applied to navigating teams of animated characters in video games [57, 45].

A solution to a MAPF instance, i.e., a list of collision-free paths (one for each agent), enables agents to move from their (unique) start locations to their (unique) target locations without collisions and then stay at their target locations permanently after following their paths. In this case, the travel time of an agent, or equivalently, the cost of its path, refers to the time for the agent to move from its start location

to its target location*. Accordingly, common objectives of MAPF include minimizing one of the following metrics:

- *sum of (path) costs* (SOC), or equivalently, *flowtime*, which is the sum of the travel time of each agent;
- *makespan*, which is the travel time of the last agent that reaches its target location, i.e., the cost of the maximum-cost path in the solution.

SOC and makespan have a Pareto-optimal structure [70]. In general, they cannot be minimized simultaneously. In this dissertation, we focus on the objective of minimizing SOC since this optimization criterion is widely used in many existing works [30, 31, 32, 56].

Although MAPF captures the need to resolve spatio-temporal collisions and optimize a performance metric, solving MAPF with the optimal SOC is NP-hard on general graphs [70], planar graphs [69], and four-neighbor grid graphs [3]. On the other hand, solving MAPF is NP-hard on directed graphs [47]. Similarly, solving MAPF with a makespan of at most a factor of 4/3 larger than optimal is NP-hard [43].

1.1 Motivation

In this section, we first explain why MAPF is an important problem. We then explain why we focus on bounded-suboptimal MAPF in this dissertation.

1.1.1 Why Multi-Agent Path Finding (MAPF)?

The reason why MAPF is worth researching is that it offers huge flexibility, allowing for the easy extension to different variants and fitting a wide range of applications. For example, in automated warehouse scenarios, MAPF can be extended to *Lifelong MAPF* [12, 25, 37, 42, 64, 74], where agents are assigned new target locations once they reach their current ones. The objective of Lifelong MAPF is to maximize the

*When calculating the travel time of an agent, the time after the agent reaches its target location and stays there permanently is ignored.

average number of target locations that agents reach over the course of the operation. A similar extension of MAPF is to require each agent to traverse a set of known target locations. For example, *Multi-Goal MAPF* [60, 62] is an extension where an agent must traverse each of its assigned set of target locations at least once. *MAPF with Precedence Constraints* [72] is another extension that requires agents to visit their assigned target locations while satisfying temporal precedence constraints between these locations. In traffic management scenarios, *Online MAPF* [40, 61] is an extension where new agents constantly show up and disappear again, and all these agents must find collision-free paths to their target locations.

To withstand agents being delayed unexpectedly, MAPF can be extended to *Robust MAPF* [2]. In this case, given the worst-case amount of delay, agents can still reach their target locations without collisions by following their paths. To accommodate agents of different sizes, MAPF can be extended to *MAPF for Large Agents* [36], where an agent can occupy multiple adjacent locations. Also, to allow for formation control, MAPF can be extended to *Moving Agents in Formation* [35], where agents must move without collision in a particular formation toward their target locations. Additionally, to incorporate a deadline by which all agents must have reached their target locations, MAPF can be extended to *MAPF with Deadlines* [44], where the objective is to maximize the number of agents that reach their target locations without collisions from their start locations within the given deadline.

Moreover, MAPF can be extended to target-location assignment for the agents. For example, MAPF can be extended to *Anonymous MAPF*. Given the same number of unique target locations as the agents, the objective of Anonymous MAPF is to assign each agent one unique target location and find a list of collision-free paths, one for each agent, that minimizes the makespan [1]. Anonymous MAPF can be generalized to the Combined Target Assignment and Path Finding problem. In this framework, multiple Anonymous MAPF instances coexist in a shared environment, where agents are assigned unique target locations based on their specific instance. Anonymous MAPF can also be generalized to each target location having a deadline by which it must be reached by a unique agent [18].

The idea of assigning target locations to agents can be further extended to simulate the pickup and delivery behavior in automated warehouses, resulting in *Multi-Agent Pickup and Delivery* (MAPD) [39]. A pickup-and-delivery task consists of a pickup location p, a delivery location d, and a release time t, indicating that a package is released at time t at location p and needs to be delivered to location d. In this case, given a set of pickup-and-delivery tasks and a set of agents starting from their start locations, the objective of MAPD is to minimize the completion time of the last pickup-and-delivery task while avoiding collisions between the agents. Similar to MAPF, MAPD can also be extended to *Lifelong MAPD* [42], where the pickup-and-delivery tasks may be released at any time, and the objective is to minimize the average difference between the finish time and the release time of a pickup-and-delivery task, and *Multi-Goal MAPD* [67], where each pickup-and-delivery task is extended from a pair of pickup and delivery locations to a sequence of locations.

By incorporating the kinodynamic constraints of the agents, MAPF can be extended to *Multi-Agent Motion Planning*, which aims to find a list of collision-free trajectories, one for each agent. A trajectory is a time-parameterized description of an agent's motion through space. For example, by connecting the states of the agents (that each describe the position, velocity, acceleration, etc. of an agent) with motion primitives (i.e., a finite set of elementary actions that an agent can perform), MAPF can be applied to non-holonomic agents [14]. By incorporating approaches that optimize the speeds of agents (rather than assuming that agents can move and stop instantaneously), MAPF can be further extended to find collision-free trajectories with motions that are smooth in time [33, 68]. By constructing a roadmap that avoids static obstacles in a shared environment, MAPF can be extended to finding collision-free trajectories on the roadmap [24, 51]. Finally, MAPF can also be extended to handling the asynchronous execution of move actions [53, 76].

In general, MAPF is one of the foundations of multi-agent systems. By finding collision-free paths for agents, MAPF serves as an interface between task planning and motion planning.

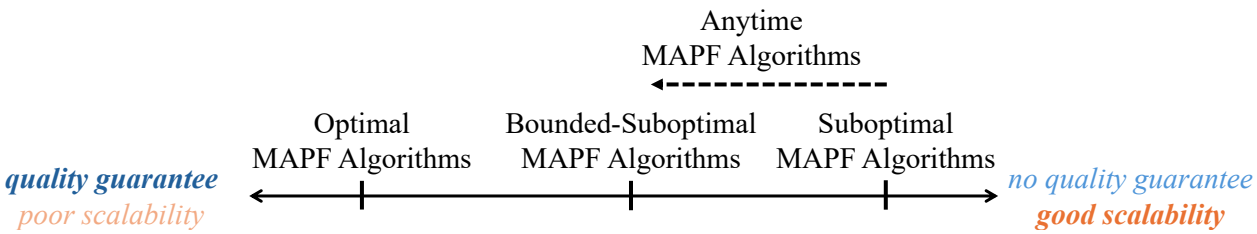


Figure 1.3: The spectrum of optimal, suboptimal, anytime, and bounded-suboptimal MAPF algorithms regarding their solution quality guarantee and their scalability.

1.1.2 Why Bounded-Suboptimal MAPF?

As shown in Figure 1.3, a spectrum of MAPF algorithms has been proposed in recent years with the objective of minimizing SOC. On the left end of the spectrum, we have *optimal MAPF algorithms*, which provide the strongest guarantees regarding the solution quality (i.e., finding solutions with the minimum SOCs). However, due to the computational intractability of finding optimal solutions for MAPF instances, these algorithms are limited in scalability as the computational overhead grows exponentially with the number of agents. Thus, researchers have been working on trading off solution quality for efficiency to scale up to MAPF instances involving thousands of agents moving in large environments, i.e., large-scale MAPF instances. Thus, on the right end of the spectrum, we have *suboptimal MAPF algorithms*, which provide solutions quickly and thus are able to solve large-scale MAPF instances [50]. Between optimal MAPF algorithms and suboptimal MAPF algorithms, we have *anytime MAPF algorithms* that quickly find an initial solution using a suboptimal MAPF algorithm and gradually improve its solution quality. However, these MAPF algorithms typically find solutions with no guarantees on the solution quality. That is, agents can take long detours or delays when following their collision-free paths.

Thus, we focus on *bounded-suboptimal MAPF algorithms*, which find a list of collision-free paths whose SOC is at most a user-specified *bounded-suboptimality factor* w larger than that of the optimal solution. That is, if an optimal solution of a MAPF instance has SOC C^*, then bounded-suboptimal MAPF algorithms guarantee to find a solution where SOC is at most $w \cdot C^*$. Bounded-suboptimal MAPF algorithms strike a balance between their scalability and their solution quality guarantee. By relaxing the solution quality

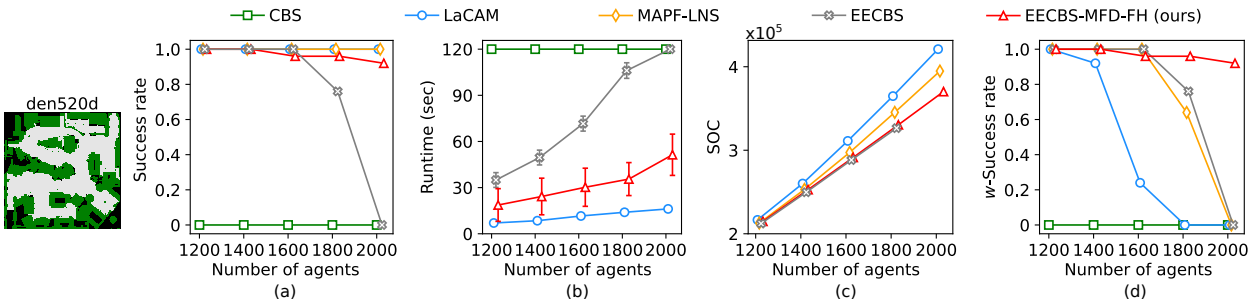


Figure 1.4: (a) success rate, (b) average runtime, (c) average sum of (path) costs (SOC), and (d) w-success rate of the state-of-the-art optimal (i.e., CBS), suboptimal (i.e., LaCAM), anytime (i.e., MAPF-LNS), and bounded-suboptimal (i.e., EECBS and our EECBS-MFD-FH) MAPF algorithms over 25 MAPF instances for each number of agents on the `den520d` graph. The vertical bars indicate the 95% confidence intervals.

guarantee from optimal to bounded-suboptimal, bounded-suboptimal MAPF algorithms achieve higher scalability than optimal MAPF algorithms. Yet, unlike suboptimal MAPF algorithms, bounded-suboptimal MAPF algorithms still provide a solution quality guarantee.

As shown in Figure 1.4, we provide some empirical results for several state-of-the-art MAPF algorithms, including *Conflict-Based Search* (CBS) [56] as an optimal MAPF algorithm, *Lazy Constraints Addition search for MAPF* (LaCAM) [50] as a suboptimal MAPF algorithm, *Large Neighborhood Search for (Anytime) MAPF* (MAPF-LNS) [27] as an anytime MAPF algorithm, and Explicit Estimation Conflict-Based Search (EECBS) [34] as a bounded-suboptimal MAPF algorithm[†]. We use the state-of-the-art versions of CBS [30, 31, 32] and LaCAM [48] rather than their vanilla versions. For MAPF-LNS, we use the state-of-the-art version of LaCAM [48] as the suboptimal MAPF algorithm to find an initial solution and then refine it with *Prioritized Planning* [57], resulting in one of the state-of-the-art versions of MAPF-LNS. For EECBS, we use the version of Li et al. [34], but without the weighted dependency graph heuristics due to its huge computational overhead[‡]. Additionally, we incorporate all enhancements proposed in this dissertation, resulting in EECBS-MFD-FH. We set the runtime limit to 120 seconds and the bounded-suboptimality factor w to 1.10. We run all MAPF algorithms on the same set of MAPF instances with the numbers of agents $[1200, 1400, 1600, 1800, 2000]$ on the `den520d` graph from the MAPF benchmark suite [59], where

[†]See Li et al. [34] for empirical comparisons of EECBS with other bounded-suboptimal MAPF algorithms.
[‡]See Section 2.4.3.4 for the weighted dependency graph heuristics.

25 MAPF instances with random start and target locations are generated for each number of agents[§]. We evaluate the success rates, average runtimes, and average SOCs over all MAPF instances with the same number of agents. We define the *solved MAPF instances* of a given MAPF algorithm per number of agents as the MAPF instances solved within the 120-second runtime limit. We define the success rate per number of agents as the number of solved MAPF instances divided by the number of all MAPF instances with the same number of agents (which is 25). Additionally, each MAPF algorithm maintains a lower bound on the optimal solution during the search. Accordingly, we define the w*-success rate* of a given MAPF algorithm per number of agents as the number of solved MAPF instances whose solutions have SOC at most the bounded-suboptimality factor w larger than the lower bound that this MAPF algorithm maintains when it terminates, divided by the number of all MAPF instances with the same number of agents (which is 25). For the calculation of the average runtime, the runtime is set to the runtime limit (i.e., 120 seconds) if a MAPF algorithm cannot find a solution within the runtime limit. The average SOC per number of agents is calculated by averaging the SOC over all MAPF instances solved by a MAPF algorithm within the runtime limit.

In Figure 1.4 (b), since MAPF-LNS iteratively refines the solution quality until the runtime reaches the runtime limit, we omit its runtime. In Figure 1.4 (c), since CBS solves zero MAPF instances for any number of agents within the 120-second runtime limit and EECBS solves zero MAPF instances for 2,000 agents, they are missing all dots or one dot, respectively.

As shown in Figures 1.4 (a) and 1.4 (b), although CBS guarantees to find optimal solutions, its computational intractability results in solving zero MAPF instances within the 120-second runtime limit, demonstrating its poor scalability. On the other hand, LaCAM solves all MAPF instances within the 120-second runtime limit (i.e., with a success rate of 1.0) regardless of the number of agents, and has the lowest average runtime among all MAPF algorithms, demonstrating its strong scalability in solving MAPF instances.

[§]See Section 5.4.1 for more detailed configuration settings, including the hardware platform.

However, as shown in Figure 1.4 (c), since LaCAM finds solutions without providing guarantees on the solution quality, it typically finds solutions with high SOCs. Furthermore, as shown in Figure 1.4 (d), apart from CBS (that solves zero MAPF instances within the runtime limit), LaCAM exhibits the worst w-success rates among the remaining MAPF algorithms. Although we can use the state-of-the-art anytime MAPF algorithm, i.e., MAPF-LNS, to iteratively improve the solutions, its average SOC and w-success rate remain worse than those of the state-of-the-art bounded-suboptimal MAPF algorithm, i.e., EECBS, and EECBS-MFD-FH.

In short, compared to optimal, suboptimal, and anytime MAPF algorithms, bounded-suboptimal MAPF algorithms offer a promising approach for balancing scalability and solution quality.

1.2 Contributions

In this dissertation, we improve the efficiency of a state-of-the-art bounded-suboptimal MAPF algorithm called EECBS [34]. EECBS is a two-level MAPF algorithm based on heuristic search. Its strategy is to iteratively find one path for each agent on the low level and then resolve collisions between them on the high level. To resolve collisions, EECBS uses constraints to prevent agents from occupying locations at times that would result in collisions. To find a bounded-suboptimal solution, EECBS finds an individually bounded-suboptimal path for each agent that satisfies its constraints. A path is individually bounded-suboptimal iff its cost is at most the bounded-suboptimality factor w larger than the cost of a minimum-cost path that satisfies the constraints. To find an individually bounded-suboptimal path, EECBS uses Focal Search on its low level [52]. Focal Search returns a *lower bound* on the cost of a minimum-cost path that satisfies the constraints and a path whose cost is at most a *threshold*, which is w times the lower bound, resulting in the path being individually bounded-suboptimal. By requiring each path to be individually bounded-suboptimal, EECBS ensures that the SOC of each agent's path is at most the bounded-suboptimality factor w larger than the sum of their lower bounds, thereby ensuring that the SOC

of each agent's path is at most the bounded-suboptimality factor w larger than the SOC of each agent's minimum-cost path that satisfies the constraints.

Despite being the state-of-the-art bounded-suboptimal MAPF algorithm, EECBS maintains its guarantee of finding bounded-suboptimal solutions by finding individually bounded-suboptimal paths. However, the definition of bounded-suboptimal solutions depends on the **sum** of path costs rather than the cost of each path. Thus, it is unnecessary to force the path of each agent to be individually bounded-suboptimal.

Since EECBS unnecessarily requires each path to be individually bounded-suboptimal, the hypothesis of this dissertation is as follows:

"Given a user-specified bounded-suboptimal factor, one can speed up EECBS by relaxing the requirement that the paths of agents be individual bounded-suboptimal while maintaining the guarantee of finding a bounded-suboptimal solution to a MAPF instance if a solution exists."

In particular, the contributions of this dissertation are as follows:

1. We propose *Flex Distribution*, an approach for EECBS that relaxes the requirement that the paths of agents be individually bounded-suboptimal while maintaining the guarantee of finding a bounded-suboptimal solution to a MAPF instance if a solution exists. When EECBS finds a list of paths, one for each agent, that satisfy constraints, it may find some agent's paths with costs close to the costs of minimum-cost paths that satisfy the constraints. In this case, Flex Distribution allows EECBS to find the paths of the other agent with costs exceeding the bounded-suboptimality factor w larger than their lower bounds, which enables these agents to make detours to avoid collisions (i.e., find paths with higher costs but fewer collisions than those without Flex Distribution), thereby allowing EECBS to terminate more quickly. Although these paths are no longer individually bounded-suboptimal, EECBS with Flex Distribution still guarantees that the SOC of each agent's path is at most the bounded-suboptimality factor w larger than the sum of their lower bounds, thereby guaranteeing to find a bounded-suboptimal solution.

2. We propose *Greedy Flex Distribution*, which is an intuitive mechanism that allows EECBS to find paths with costs as high as possible, as long as the SOC of each agent's path is at most the bounded-suboptimality factor w larger than the sum of their lower bounds. We also propose *Conflict-Based Flex Distribution*, which determines thresholds of Focal Search based on the number of collisions between paths. We also propose *Delay-Based Flex Distribution*, which estimates the additional cost (i.e., delays) required for a path to satisfy the constraints and then determines its threshold accordingly. Furthermore, we propose *Mixed-Strategy Flex Distribution*, which combines Conflict-Based Flex Distribution and Delay-Based Flex Distribution in a hierarchical framework. We prove that EECBS, combined with our Flex Distribution mechanisms, is still guaranteed to find bounded-suboptimal solutions if such solutions exist. Empirically, we show that EECBS with our Flex Distribution mechanisms outperforms the one without Flex Distribution in terms of success rates for a 120-second runtime limit.

3. To improve the efficiency of EECBS, another line of research involves generating *guidance heuristics* that help reduce collisions when EECBS finds paths on its low level. In this dissertation, we combine Flex Distribution and guidance heuristics to further improve the efficiency of EECBS. In particular, we propose a two-phase guidance framework for computing guidance heuristics. Then, we propose the *Flow-Based Guidance Framework*, which uses Flex Distribution when computing the *Flow-Based Guidance Heuristic*. That is, we apply Flex Distribution in both computing guidance heuristics and EECBS. Empirically, we show that EECBS with our Mixed-Strategy Flex Distribution solves MAPF instances even more quickly when using our Flow-Based Guidance Heuristics. We also show that EECBS with our Mixed-Strategy Flex Distribution solves MAPF instances more quickly when using our Flow-Based Guidance Heuristics than when using other state-of-the-art guidance heuristics.

1.3 Dissertation Structure

The structure of this dissertation is as follows: In Chapter 2, we first introduce the preliminaries, including the standard terminology of heuristic search, the problem formulation of MAPF, and a category of MAPF algorithms called the Conflict-Based Framework. We then introduce EECBS, a state-of-the-art bounded-suboptimal MAPF algorithm that belongs to the Conflict-Based Framework. In Chapter 3, we propose our Flex Distribution approach that tackles the limitations of EECBS. This includes the definition of flex and the Greedy Flex Distribution for distributing flex. In Chapter 4, we propose other mechanisms of distributing flex for EECBS, including the Conflict-Based Flex Distribution, the Delay-Based Flex Distribution, and the Mixed-Strategy Flex Distribution. In Chapter 5, we propose the Flow-Based Guidance Framework, which uses Flex Distribution to compute the Flow-Based Guidance Heuristic, which guides the low-level Focal Search to find paths for the agents.

Chapter 2

Preliminaries

In this chapter, we first introduce the standard terminology of heuristic search. Then, we introduce the problem formulation of Multi-Agent Path Finding (MAPF). After that, we introduce a two-level heuristic-search-based framework, called the Conflict-Based Framework [4], which is a class of MAPF algorithms commonly used to solve MAPF. The Conflict-Based Framework iteratively finds one path for each agent on the low level and then resolves conflicts (i.e., collisions) on the high level. Then, we introduce Explicit Estimation Conflict-Based Search (EECBS) [34], a MAPF algorithm that belongs to the Conflict-Based Framework. EECBS is the state-of-the-art bounded-suboptimal MAPF algorithm. It uses Explicit Estimation Search [63] on the high level and Focal Search [52] on the low level. Finally, we introduce existing enhancements that improve the efficiency of EECBS, including bypassing conflicts, symmetry breaking, prioritizing conflicts, and the Weighted Dependency Graph heuristic [34].

2.1 Standard Terminology of Heuristic Search

In this section, we follow Russell and Norvig [55] for the standard terminology used in heuristic search. A problem instance for a heuristic search algorithm, or equivalently, a search instance, consists of an initial state and a goal state, where a *state* represents a specific configuration of the problem instance. A goal test is used to determine if a state is a goal state. A problem instance also consists of *actions* that are applicable

in a given state, where each action has a *step cost*. A *transition model* of a problem instance returns the successor state that results from applying an action in a state. Here, we focus on *deterministic* problem instances, i.e., the transition model returns only one successor state. The *state space* of a problem instance is the set of all states reachable from the initial state by any sequence of actions. A *path* in the state space is a sequence of states, which implies that two adjacent states are connected by an action. The *cost* of a path is the sum of the step costs of the actions along the path. A *solution* is a sequence of actions from the initial state to the goal state. It can also be represented as a path from the initial state to the goal state.

The *search space* is the subset of the state space that a heuristic search algorithm actually reaches during the search. Typically, a heuristic search algorithm uses a *search node* to represent a state. A heuristic search algorithm begins with a *root* search node that contains the initial state of the problem instance. It then iteratively selects a search node and runs a goal test, which determines whether the state of the selected search node is the goal state. A search node that contains the goal state is referred to as a *goal* search node. If the selected search node is a goal search node, then the heuristic search algorithm terminates and returns a solution. Otherwise, the heuristic search algorithm performs a *node expansion*, which involves applying all possible actions to the state of the selected node and generating child search nodes, one for each successor state. After that, the heuristic search algorithm begins the next iteration by selecting a previously unselected search node. The iterative process of a heuristic search algorithm results in a *search tree*, and a search node from the *frontier* of the search tree is selected at each iteration.

A particular approach of heuristic search is called *best-first search*. At each iteration, a search node n with its state $s(n)$ is selected according to a cost function $f(n)$. The cost function is an estimate of the smallest possible cost of a path from the initial state via state $s(n)$ to the goal state. Thus, the node n with the minimum f-value $f(n)$ is selected first. A* [21] is a best-first search algorithm that evaluates search nodes n with $f(n) = g(n) + h(n)$, where $g(n)$ is the cost needed to reach state $s(n)$ from the initial state,

and the *heuristic* $h(n)$ is an estimate of the cost needed from state $s(n)$ to the goal state (i.e., a cost-to-go function that assigns an estimated cost to state $s(n)$).

Definition 2.1 (Admissible Heuristic)**.** *A heuristic $h(n)$ is **admissible** iff, for each search node n, it never overestimates the minimum cost of any path from the state $s(n)$ to the goal state.*

Typically, A* maintains the frontier of the search tree with an *open list* (OPEN), which is typically implemented with a priority queue that sorts the search nodes in increasing order of their f-values.

2.2 Problem Formulation of MAPF

Given a shared environment and a set of k agents $A = \{a_i \mid i \in [k]\}$, where $[k] = [1, 2, \ldots, k]$, each with a start location and a target location, Multi-Agent Path Finding (MAPF) is the deterministic problem of finding a list of collision-free paths, one for each agent, while minimizing their sum of travel time, or, equivalently, sum of path costs. An agent waits at its target location permanently after following its path.

In this dissertation, we use the problem formulation of MAPF from Stern et al. [59]. The shared environment is a four-neighbor undirected grid graph $G = (V, E)$ with a finite number of vertices (and edges), where each vertex represents a location. Each agent a_i has a start vertex $s_i \in V$ and a target vertex $l_i \in V$, where

$$(s_i, l_i) \neq (s_j, l_j) \text{ iff } i \neq j,\ \forall i, j \in [k]. \tag{2.1}$$

Time is discretized into timesteps. At timestep 0, each agent is at its start vertex. At each timestep, an agent can either *move* to an adjacent vertex or *wait* at its current vertex. Each action requires one timestep to execute. We define a *path* p_i of agent a_i as a sequence of vertices from its start vertex s_i to its target vertex l_i, where $p_i[0] = s_i$ and $p_i[t] \in V$ indicates the vertex at which agent a_i is at timestep t. We define the *cost* c_i of path p_i as the number of timesteps needed for agent a_i to move from its start vertex s_i (at

timestep 0) to its target vertex l_i while following its path p_i, ignoring the timesteps when agent a_i waits at its target vertex l_i without leaving it.

As each agent follows its path, there are two types of conflicts or, equivalently, collisions: vertex conflicts and edge conflicts. A *vertex conflict* $\langle a_i, a_j, v, t\rangle$ occurs when agents a_i and a_j attempt to reach the same vertex $v \in V$ at the same timestep t, i.e., $p_i[t] = p_j[t] = v$. An *edge conflict* $\langle a_i, a_j, u, v, t\rangle$ occurs when agents a_i and a_j attempt to traverse the same edge $(u, v) \in E$ between two adjacent vertices u and v in opposite directions at the same timestep t (or, more precisely, from timestep $t-1$ to timestep t with $t \geq 1$). Without loss of generality, the notation of an edge conflict represents the situation where agent a_i moves from vertex u to v and agent a_j moves from vertex v to u at timestep t, i.e., $p_i[t-1] = p_j[t] = u$ and $p_i[t] = p_j[t-1] = v$.

A *solution* of a MAPF instance is a list of conflict-free (or, equivalently, collision-free) paths $[p_i \mid i \in [k]]$, one for each agent a_i to move from its start vertex s_i to its target vertex l_i. The objective of MAPF is to minimize the *sum of (path) costs* (SOC) $\sum_{i\in[k]} c_i$. We say that a solution is *optimal* iff its SOC is minimal. We use C^* to denote the minimum SOC. We say that a solution is *bounded-suboptimal* iff its SOC is at most $w \cdot C^*$, where $w \geq 1$ is a user-specified *bounded-suboptimality factor*. That is, the SOC of a bounded-suboptimal solution is at most the bounded-suboptimality factor w larger than the SOC of an optimal solution. We say that a solution is *suboptimal* iff it is neither optimal nor bounded-suboptimal.

2.3 Conflict-Based Framework

In this dissertation, we focus on MAPF algorithms based on heuristic search, which can typically be divided into two categories. One of the categories involves using heuristic search in the joint-state space, where *joint states* represent different ways of placing all k agents on the vertices V, one agent per vertex. Since each agent can perform at most five actions at each timestep on a four-neighbor grid graph (i.e., move in each of the four cardinal directions plus wait at its current vertex), there exist at most 5^k combinations

of actions for k agents. Thus, each search node has at most 5^k children, which can be exponential in the number of agents. However, since the number of child search nodes can be exponential in the number of agents, heuristic search in the joint-state space results in large runtime overhead in finding optimal and bounded-suboptimal MAPF [48, 56].

To improve the efficiency of solving MAPF with a guarantee on the solution quality, previous works have developed a strategy that iteratively finds one path for each agent individually and then resolves the conflicts afterwards. This results in a two-level framework called the *Conflict-Based Framework* [4]. On the high level, the Conflict-Based Framework resolves conflicts by using constraints to prevent agents from being at vertices at timesteps that result in conflicts. On the low level, it finds paths for agents that satisfy their constraints. A *vertex constraint* $\langle a_i, v, t \rangle$ indicates that agent a_i is not allowed to be at vertex v at timestep t. An *edge constraint* $\langle a_i, u, v, t \rangle$ indicates that agent a_i is not allowed to traverse an edge from vertex u to vertex v at timestep t (or, more precisely, from timestep $t-1$ to timestep t with $t \geq 1$). On the high level, the Conflict-Based Framework runs a heuristic search algorithm on a search tree called the *Constraint Tree* (CT). Each CT node contains a list of sets of constraints, one set for each agent. Each CT node also contains a list of paths, one for each agent, that satisfy the constraints. That is, the Conflict-Based Framework uses a list of paths for its state representation, rather than a joint state. At each iteration, the Conflict-Based Framework selects a CT node and checks if its list of paths is a solution (i.e., is conflict-free). If not, then it expands the selected CT node by selecting a conflict and generating two child CT nodes, each with additional constraints that, respectively, prevent one of the agents from being at the vertex (or traversing the edge) at the timestep when the conflict occurs. We use the term "high-level" to refer to any heuristic search algorithm that aims to find a solution on the CT. On the low level, the Conflict-Based Framework attempts to find paths, one for each agent, that satisfy the constraints in a CT node. It runs a heuristic search algorithm on the vertex-timestep space, where each search node indicates the vertex at which an agent is located at a given timestep. We use the term "low-level" to refer to any

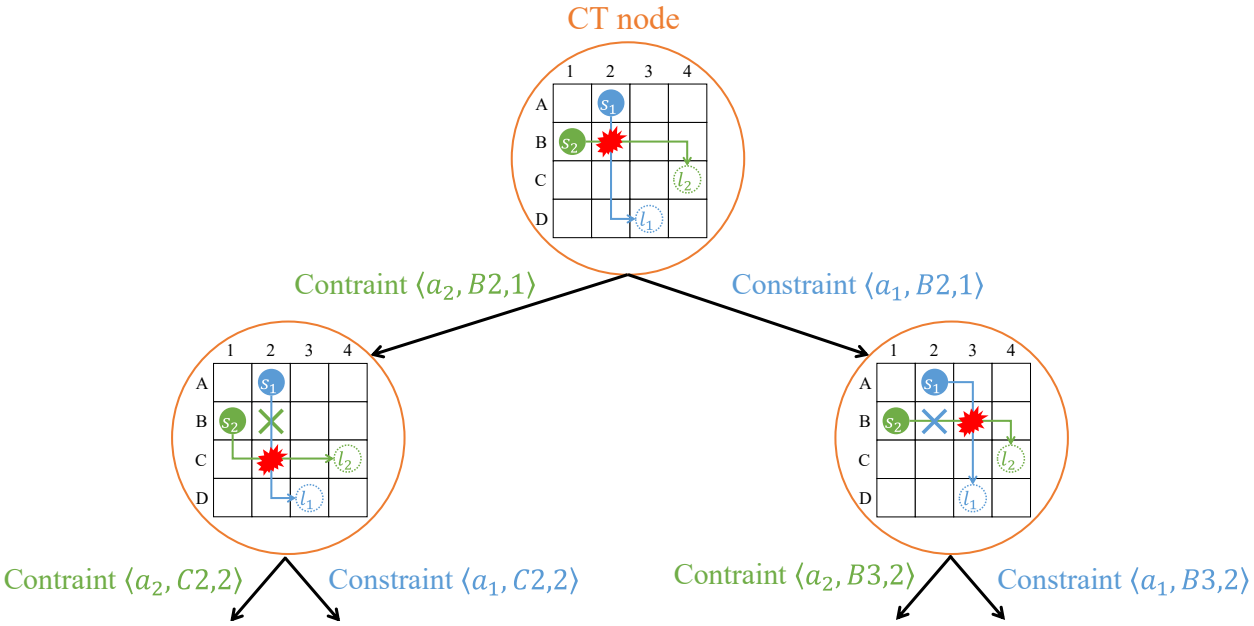


Figure 2.1: An example of CT node expansions when solving a MAPF instance with two agents on a 4×4 grid graph with start vertices $[s_1, s_2] = [A2, B1]$ and target vertices $[l_1, l_2] = [D3, C4]$.

heuristic search algorithm that aims to find a path for an agent. Figure 2.1 shows an illustrative example of CT node expansions when solving a MAPF instance with two agents.

The Conflict-Based Framework can be instantiated with the heuristic search algorithms used on the high and low levels. For example, Conflict-Based Search (CBS) [56] and its variants [5, 30] are optimal MAPF algorithms that use A* on both the high and the low levels. Enhanced Conflict-Based Search is a bounded-suboptimal MAPF algorithm that uses Focal Search [52] on both the high and the low levels. Priority-Based Search [41] and its variants [4, 10] are MAPF algorithms that solve MAPF suboptimally by running a depth-first search on the high level, with agents being assigned priorities, where the low-priority agents should avoid any conflicts with the high-priority agents. For example, suppose that agent a_i has a higher priority than agent a_j. If agent a_i moves from vertex u to vertex v from timestep t to $t+1$, then Priority-Based Search and its variants need to find a path for agent a_j that satisfies two vertex constraints $\langle a_j, u, t\rangle$ and $\langle a_j, v, t+1\rangle$ and an edge constraint $\langle a_j, u, v, t, t+1\rangle$. Priority-Based Search and its variants use A* on the low level to find paths that minimize either the cost [4, 41] or the number of conflicts [10].

2.4 Explicit Estimation Conflict-Based Search (EECBS)

Based on the Conflict-Based Framework, Explicit Estimation Conflict-Based Search (EECBS) [34] is a two-level bounded-suboptimal MAPF algorithm. On the high level, EECBS uses a bounded-suboptimal heuristic search algorithm called Explicit Estimation Search (EES) [63]. On the low level, EECBS uses another bounded-suboptimal heuristic search algorithm called Focal Search [52].

2.4.1 High-Level Explicit Estimation Search (EES)

On the high level, EECBS constructs a CT to resolve conflicts with constraints and runs *Explicit Estimation Search* (EES) [63] on the CT. Since EECBS belongs to the Conflict-Based Framework, it iteratively selects a CT node from the frontier of the CT and checks if its list of paths is a solution (i.e., is conflict-free). If not, then EECBS expands the selected CT node by selecting a conflict between its paths and generating two child CT nodes, each with additional constraints that, respectively, prevent one of the agents from being at the vertex (or traversing the edge) at the timestep when the conflict occurs. If EECBS successfully finds paths for agents that satisfy the constraints in a child CT node, then this child CT node becomes one of the CT nodes in the frontier. Table 2.1 shows the main components of each CT node N used in EECBS.

EECBS maintains three lists: CLEANUP$_H$, OPEN$_H$, and FOCAL$_H$, each implemented as a priority queue. CLEANUP$_H$ is a regular open list of A* that sorts the frontier of the CT in increasing order of a cost function

$$F(N) = LB(N) + H(N) \tag{2.2}$$

of CT node N, where $LB(N) \geq 0$ and $H(N) \geq 0$, breaking ties in favor of CT nodes with smaller numbers of conflicts. $LB(N) = \sum_{i \in [k]} lb_i(N)$ is the *sum of lower bounds* (SOLB) of CT node N, where $lb_i(N)$ is a lower bound on the cost of a minimum-cost path for agent a_i that satisfies the constraints $\Psi_i(N)$ in CT node N. $H(N)$ is an admissible heuristic, i.e., a cost-to-go function that never overestimates

Table 2.1: Main components of a CT node N

Component	Notation	Definition
Constraints	$\Psi(N) = [\,\Psi_i(N) \mid i \in [k]\,]$	List of sets of constraints.
Paths	$[\,p_i(N) \mid i \in [k]\,]$	List of paths that satisfy the constraints $\Psi(N)$.
Lower bounds	$[\,lb_i(N) \mid i \in [k]\,]$	Lower bounds on the costs of the minimum-cost paths satisfying the constraints $\Psi(N)$.
Costs of paths	$[\,c_i(N) \mid i \in [k]\,]$	Costs of paths $[\,p_i(N) \mid i \in [k]\,]$.
Costs of the minimum-cost paths	$[\,c_i^*(N) \mid i \in [k]\,]$	Costs of the minimum-cost paths satisfying the constraints $\Psi(N)$.
Sum of the lower bounds (SOLB)	$LB(N) = \sum_{i \in [k]} lb_i(N)$	
Sum of costs (SOC)	$C(N) = \sum_{i \in [k]} c_i(N)$	
Admissible heuristics	$H(N)$	Cost-to-go function that never overestimates the difference between the minimum SOC over all solutions that satisfy the constraints $\Psi(N)$ and the SOLB $LB(N)$.
Inadmissible heuristics	$\hat{H}(N)$	Cost-to-go function estimating the difference between the minimum SOC over all solutions that satisfy the constraints $\Psi(N)$ and the SOC $C(N)$.
Admissible cost function	$F(N) = LB(N) + H(N)$	
Inadmissible cost function	$\hat{F}(N) = C(N) + \hat{H}(N)$	
Conflicts	$\mathcal{X}(N)$	Set of conflicts over the paths $[p_i(N) \mid i \in [k]]$.
Number of conflicts	$\lvert\mathcal{X}(N)\rvert$	Number of conflicts in $\mathcal{X}(N)$.

the difference between the minimum SOC over all solutions that satisfy the constraints $\Psi(N) = [\Psi_i(N) \mid i \in [k]]$ and SOLB $LB(N)$. Since $H(N)$ is admissible, $F(N) = LB(N) + H(N)$ is a lower bound on the minimum SOC over all solutions that satisfy the constraints $\Psi(N)$. We use N_F to denote a CT node with the minimum F-value over all CT nodes in CLEANUP$_H$, i.e., $N_F = \arg\min_{N' \in \text{CLEANUP}_H} F(N')$, and use $LB = F(N_F) = LB(N_F) + H(N_F)$ to denote the minimum F-value over all CT nodes in CLEANUP$_H$. While $LB(N) = \sum_{i \in [k]} lb_i(N)$ only takes the SOLB from the low-level search into account, LB takes both the SOLB $LB(N_F)$ and the high-level admissible heuristic value $H(N_F)$ into account.

OPEN$_H$ is a regular open list of A* that sorts the frontier of the CT in increasing order of another cost function $\hat{F}(N) = C(N) + \hat{H}(N)$ of CT node N, breaking ties in favor of CT nodes with smaller numbers

of conflicts. $C(N) = \sum_{i \in [k]} c_i(N)$ is the SOC of CT node N, where $c_i(N)$ is the cost of path $p_i(N)$ for agent a_i that satisfies the constraints $\Psi_i(N)$. $\hat{H}(N)$ is a potentially inadmissible heuristic, which is a cost-to-go function that estimates the difference between the minimum SOC that satisfies the constraints $\Psi(N)$ and SOC $C(N)$. That is, $\hat{F}(N)$ is an estimate of the minimum SOC over all solutions that satisfy the constraints $\Psi(N)$. We use $N_{\hat{F}}$ to denote the CT node with the minimum $\hat{F}$-value in OPEN$_H$, i.e., $N_{\hat{F}} = \arg\min_{N \in \text{OPEN}_H} \hat{F}(N)$. That is, $\hat{F}(N_{\hat{F}})$ is an estimate of the SOC of the optimal solution, and EECBS suspects that CT node $N_{\hat{F}}$ is in the branch of CT that leads to an optimal solution.

FOCAL$_H$ contains those CT nodes N in OPEN$_H$ with $\hat{F}(N) \leq w \cdot \hat{F}(N_{\hat{F}})$, sorted in increasing order of the number of conflicts $|\mathcal{X}(N)|$, breaking ties in favor of CT nodes with smaller F-values. That is, FOCAL$_H$ is a subset of OPEN$_H$ and thus is also a subset of CLEANUP$_H$. We use $N_{\mathcal{X}}$ to denote the CT node with the minimum number of conflicts in FOCAL$_H$, i.e., $N_{\mathcal{X}} = \arg\min_{N \in \text{FOCAL}_H} |\mathcal{X}(N)|$.

To select a CT node at each iteration, EECBS uses the following selection rules in order:

(E1) select CT node $N_{\mathcal{X}}$ if $C(N_{\mathcal{X}}) \leq w \cdot LB$;

(E2) otherwise, select CT node $N_{\hat{F}}$ if $C(N_{\hat{F}}) \leq w \cdot LB$; and

(E3) otherwise, select CT node N_F.

According to these selection rules, the SOC of the selected CT node is at most the bounded-suboptimality factor w larger than LB.

Lemma 2.1. *At each iteration before EECBS expands a CT node, if there is a CT node N whose list of paths is a solution, then it is in the subtree rooted at at least one of the CT nodes in CLEANUP$_H$. That is, there is at least one of the CT nodes in CLEANUP$_H$ where the list of paths in CT node N satisfies its constraints.*

Proof. We prove this by induction. For the base case, CLEANUP$_H$ contains only the root CT node. Since the root CT node contains zero constraints, CT node N is in the subtree rooted at the root CT node in CLEANUP$_H$. For the inductive hypothesis, we assume that, at an iteration before EECBS expands a CT

node, CT node N is in the subtree rooted at at least one of the CT nodes in CLEANUP$_H$. Then, when EECBS expands a CT node $\hat{N}$ from CLEANUP$_H$, it selects a conflict $\mathcal{X}(\hat{N})$ and generates two child CT nodes with additional constraints that resolve the conflict. Thus, a solution that satisfies the constraints $\Psi(\hat{N})$ must satisfy the constraints of at least one of the child CT nodes of CT node $\hat{N}$. Thus, a CT node whose list of paths is a solution that satisfies the constraints $\Psi(\hat{N})$ is in the subtree of one of the child CT nodes of CT node $\hat{N}$. As EECBS inserts these two child CT nodes into CLEANUP$_H$ and starts the next iteration, we prove that our hypothesis holds. Thus, by induction, Lemma 2.1 holds. □

Property 2.1. *LB is a lower bound on the SOC of an optimal solution.*

Proof. Given a CT node N, since $F(N) = LB(N) + H(N)$ and $H(N)$ is admissible, $F(N)$ is a lower bound on the minimum SOC over all solutions that satisfy the constraints $\Psi(N)$. That is, for a CT node in the subtree rooted at CT node N whose list of paths is a minimum-SOC solution, $LB(N)$ is a lower bound on its SOC. According to Lemma 2.1, a CT node whose list of paths is an optimal solution is in the subtree rooted at at least one of the CT nodes, say CT node N, in CLEANUP$_H$. Since $LB = F(N_F) = \min_{N' \in \text{CLEANUP}_H} F(N')$ is the minimum F-value over all CT nodes in CLEANUP$_H$, LB is a lower bound on the SOC of an optimal solution. □

According to Property 2.1, since LB is the lower bound on the SOC of an optimal solution, if the list of paths of the selected CT node $\hat{N}$ is a solution (i.e., is conflict-free), then this solution is bounded-suboptimal (i.e., $C(\hat{N}) \leq w \cdot LB \leq w \cdot C^*$). If EECBS selects a CT node according to the Select Rules (E1) or (E2), then it removes the selected CT node from CLEANUP$_H$, OPEN$_H$, and FOCAL$_H$. If EECBS selects a CT node according to the Select Rule (E3), then it removes the selected CT node from CLEANUP$_H$ and OPEN$_H$, and FOCAL$_H$ if the selected CT node is in FOCAL$_H$.

If the list of paths is not conflict-free, then EECBS expands the selected CT node. The rationale behind the Selection Rule (E1) is that, the smaller the number of conflicts $|\mathcal{X}(N_{\mathcal{X}})|$ of CT node $N_{\mathcal{X}}$, the more likely

EECBS can expand few CT nodes to find a CT node whose list of paths is a solution in the subtree of CT node $N_{\mathcal{X}}$. Thus, the more EECBS selects a CT node with the Selection Rule (E1), the more likely it is to find a solution quickly. The rationale behind the Selection Rule (E2) is that EECBS suspects that the CT node $N_{\hat{F}}$ is located on the branch of the CT that leads to an optimal solution. Thus, expanding it can help EECBS to find a solution (hopefully with a small SOC) quickly. The rationale behind the Selection Rule (E3) is that, after EECBS expands CT node N_F, the new minimum F-value of all CT nodes in CLEANUP$_H$, i.e., the new LB, may be larger than $F(N_F)$. By increasing LB, the conditions of Selection Rules (E1) and (E2) are more likely to hold in future iterations.

To expand a CT node $\hat{N}$, EECBS first selects a conflict. Given a selected vertex conflict $\langle a_i, a_j, v, t\rangle$ (or, respectively, edge conflict $\langle a_i, a_j, u, v, t\rangle$), EECBS generates two child CT nodes, N and N', initialized with the same constraints, paths, and lower bounds as contained in CT node $\hat{N}$. Then, EECBS adds an additional vertex constraint $\langle a_i, v, t\rangle$ (or, respectively, edge constraint $\langle a_i, u, v, t\rangle$) to the set of constraints $\Psi_i(N)$, and an additional vertex constraint $\langle a_j, v, t\rangle$ (or, respectively, edge constraint $\langle a_j, v, u, t\rangle$) to the set of constraints $\Psi_j(N')$*. EECBS uses the low-level search for each child CT node to find paths for agents that satisfy their sets of constraints. If the paths of a child CT node are found, then this child CT node is added to CLEANUP$_H$ and OPEN$_H$. After that, EECBS updates the CT nodes N_F to the CT node with the minimum F-value in CLEANUP$_H$ and the CT node $N_{\hat{F}}$ to the CT node with the minimum $\hat{F}$-value in OPEN$_H$. Then, it adds any CT nodes $N \in$ OPEN$_H$ with $\hat{F}(N) \leq w \cdot \hat{F}(N_{\hat{F}})$ to FOCAL$_H$.

Given a CT node N, to compute its inadmissible heuristic $\hat{H}(N)$, Li et al. [34] follow Thayer et al. [63] and compute its *one-step distance error* $\epsilon_d(N) = |\mathcal{X}(bc(N))| - (|\mathcal{X}(N)| - 1)$ and its *one-step cost error* $\epsilon_h(N) = C(bc(N)) - C(N)$, where $bc(N)$ is the *best child CT node* of CT node N, i.e., the child CT node with the lowest $\hat{F}$-value, breaking ties in favor of the child CT node with a smaller number of conflicts. Both values, $\epsilon_d(N)$ and $\epsilon_h(N)$, can be computed after each CT node expansion. Additionally, Li et al. [34]

*When using symmetry breaking [31, 34], multiple constraints may be added.

follow Thayer et al. [63] and use a *global error model*, which assumes that the distribution of one-step errors across the entire search space is uniform and can be estimated as an average of all observed one-step errors. Thus, Li et al. [34] maintain a running average of the one-step distance error $\bar{\epsilon_d}$ and the one-step cost error $\bar{\epsilon_h}$. Then, the inadmissible heuristic of CT node N is defined as

$$\hat{H}(N) = \frac{\bar{\epsilon_h}}{1 - \bar{\epsilon_d}} \cdot |\mathcal{X}(N)|, \tag{2.3}$$

where $\bar{\epsilon_h}/(1 - \bar{\epsilon_d})$ is an estimate of the SOC increment per conflict resolution. According to Equation 2.3, the larger the $|\mathcal{X}(N)|$, the larger the $\hat{H}(N)$. This indicates that the more conflicts a CT node N contains, the larger the difference between the SOC (which is estimated as $\hat{F}(N) = C(N) + \hat{H}(N)$) of a solution that satisfies the constraints $\Psi(N)$ and $C(N)$.

Algorithm 2.1 (with its tool functions in Algorithm 2.2) shows the pseudo-code of the high-level EES of EECBS, including its existing enhancements (see Section 2.4.3) and our proposed Flex Distribution (see Section 3.3 for the definition). The existing enhancements of EECBS described in this dissertation include bypassing conflicts [Lines 36-40], symmetry breaking [Line 21 in Algorithm 2.2], prioritizing conflicts [Line 20 in Algorithm 2.2], and the Weighted Dependency Graph heuristic [Lines 9, 16-19]. EECBS first generates a root CT node N_0 and inserts it into CLEANUP$_H$ and OPEN$_H$. It then updates FOCAL$_H$ based on OPEN$_H$ [Lines 3-11]. At each iteration, EECBS selects a CT node $\hat{N}$ and checks if its paths are conflict-free. If so, then EECBS returns its list of paths as the solution, which is guaranteed to be bounded-suboptimal [Line 15]. If $H(\hat{N})$ has not yet been computed, EECBS computes it and inserts the CT node $\hat{N}$ back into CLEANUP$_H$ and OPEN$_H$, and updates FOCAL$_H$ based on OPEN$_H$ [Lines 16-19]. Otherwise, EECBS determines the priority of each conflict in $\mathcal{X}(\hat{N})$ [Lines 20-21 in Algorithm 2.2], which includes determining its cardinality and applying symmetry breaking. EECBS then selects the conflict with the highest priority in $\mathcal{X}(\hat{N})$ and generates two sets of constraints Ψ_1 and Ψ_2 for resolving it [Line 20]. For

each set of constraints, EECBS generates a child CT node N that contains the set of constraints [Line 24]. For a generated child CT node N, EECBS finds paths that satisfy its constraints and finds all conflicts in its list of paths $[p_i(N) \mid i \in [k]]$ [Lines 25-34]. Then, EECBS applies bypassing conflicts if CT node N satisfies conditions (C1-C4) (see Section 2.4.3.1) [Lines 36-40]. If bypassing conflicts does not occur, then EECBS computes the inadmissible heuristic $H(N)$ for each generated CT node N, inserts it into CLEANUP$_H$ and OPEN$_H$, and updates FOCAL$_H$ based on OPEN$_H$ [Lines 42-44]. EECBS then computes $\epsilon_d(\hat{N})$ and $\epsilon_h(\hat{N})$ and updates $\bar{\epsilon_d}$ and $\bar{\epsilon_h}$ [Line 45].

2.4.2 Low-Level Focal Search

On the low level, to find a path for an agent a_i in a CT node N that satisfies the constraints $\Psi_i(N)$, EECBS constructs a search tree with *vertex-timestep* (v-t) nodes. The *state representation* $s_i(n)$ of a v-t node n is a tuple (v, t) indicating that agent a_i is at vertex v at timestep t. The *root v-t node* is a v-t node with state representation $(s_i, 0)$. Given a v-t node n with state representation (v, t), EECBS can *extract* a path from v-t node n with a cost t by back-tracking it to the root v-t node, which results in a sequence q of vertices with $q[0] = s_i$ and $q[t] = v$. A *goal v-t node* is a v-t node with state representation (l_i, t_l) where a path extracted from it satisfies the constraints $\Psi_i(N)$.

For a v-t node n with state representation (v, t), we define priority function $f_i(n) = g_i(n) + h_i(n) = g_i(n) + h_i(v)$, where $g_i(n) = t$ is the number of timesteps needed for agent a_i to move from its start vertex s_i at timestep 0 to vertex v at timestep t following the path extracted from v-t node n, and $h_i(n) = h_i(v)$ is an admissible heuristic that never overestimates the number of timesteps needed for agent a_i to move from vertex v to its target vertex l_i. We set $h_i(v)$ to the cost of a minimum-cost path from vertex v to the target vertex l_i[†]. We also define another priority function $\tilde{f}_i(n) = g_i(n) + \tilde{h}_i(n) = g_i(n) + \tilde{h}_i(v)$, where $\tilde{h}_i(n) = \tilde{h}_i(v)$ is a *secondary heuristic*. We set $\tilde{h}_i(v)$ to $h_i(v)$ unless mentioned otherwise. Lastly, we define

[†]Given a MAPF instance with a graph $G = (V, E)$, we compute the heuristic $h_i(v)$ by running Dijkstra's algorithm from target vertex l_i to each vertex $v \in V$.

Table 2.2: Main components of a v-t node n

Component	Notation	Definition
State representation	$s_i(n) = (v, t)$	A tuple (v, t) indicating that agent a_i is at vertex v at timestep t.
g_i-value	$g_i(n)$	Number of timesteps needed for agent a_i to move from its start vertex s_i at timestep 0 to vertex v at timestep t, following the path extracted from v-t node n with its state representation (v, t).
h_i-value	$h_i(n) = h_i(v)$	Admissible heuristic value that never overestimates the number of timesteps needed for agent a_i to move from vertex v to its target vertex l_i.
f_i-value	$f_i(n) = g_i(n) + h_i(n)$	
$\tilde{h}_i$-value	$\tilde{h}_i(n) = \tilde{h}_i(v)$	Secondary heuristic value.
$\tilde{f}_i$-value	$\tilde{f}_i(n) = g_i(n) + \tilde{h}_i(n)$	
Number of conflicts	$x_i(n)$	Number of conflicts with the paths of the other agents when agent a_i moves from its start vertex s_i at timestep 0 to vertex v at timestep t, following the path extracted from v-t node n with its state representation (v, t).
Pointer to the parent	*parent*(n)	

$x_i(n)$ as the number of conflicts with the paths of the other agents $\{a_j \mid j \in [k] \setminus \{i\}\}$ when agent a_i moves from its start vertex s_i at timestep 0 to vertex v at timestep t, following the path extracted from v-t node n. Table 2.2 shows the main component of each v-t node n used in the low-level Focal Search.

EECBS then runs *Focal Search* [52] on the low level to find a bounded-suboptimal path $p_i(N)$ with a cost $c_i(N)$ and a lower bound $lb_i(N)$ on the optimal path that satisfies the constraints $\Psi_i(N)$. Apart from a CLOSED$_L$ list that contains all expanded v-t nodes, the low-level Focal Search maintains two other lists, OPEN$_L$ and FOCAL$_L$, both of which are implemented as a priority queue. OPEN$_L$ sorts the frontier of the search tree, i.e., all generated but not yet expanded v-t nodes n, in increasing order of their f_i-values $f_i(n)$. We use $f_{\min,i}(N)$ to denote the minimum f_i-value over all v-t nodes in OPEN$_L$, i.e.,

$$f_{\min,i}(N) = \min_{n \in \text{OPEN}_L} f_i(n). \tag{2.4}$$

Since $h_i(n)$ is an admissible heuristic, and given that OPEN$_L$ consistently contains at least one v-t node n with state representation (v, t) from a minimum-cost path $p^*(N)$ that satisfies the constraints $\Psi_i(N)$ (i.e., $p^*(N)[t] = v$) during the search [21], it follows that $f_{\min,i}(N)$ is a lower bound on the cost $c_i^*(N)$ of a minimum-cost path that satisfies the constraints $\Psi_i(N)$ (i.e., $f_{\min,i}(N) \leq c_i^*(N)$).

On the other hand, FOCAL$_L$ contains those v-t nodes n in OPEN$_L$ with $f_i(n) \leq \tau_i(N) = w \cdot f_{\min,i}(N)$, where we use $\tau_i(N)$ to denote the *threshold*. The low-level Focal Search sorts these v-t nodes n in increasing order of their number of conflicts $x_i(n)$, breaking ties in favor of v-t nodes with smaller $\tilde{f}_i$-values. If there are v-t nodes with the same number of conflicts and the same $\tilde{f}_i$-value, then the low-level Focal Search breaks ties in favor of v-t nodes with smaller values of $\tilde{h}_i(n)$, and then random selection for any further unresolved ties.

At each iteration, the low-level Focal Search first updates FOCAL$_L$ if $f_{\min,i}(N)$ has increased, and then selects the v-t node $\hat{n}$ in FOCAL$_L$ that has the minimum number of conflicts $x_i(n)$. If the selected v-t node n in FOCAL$_L$ is a goal v-t node, then the low-level FOCAL Search has found a path that satisfies the constraints $\Psi_i(N)$ with cost $c_i(N) = f_i(n)$. Otherwise, the low-level Focal Search expands the selected v-t node. When the low-level Focal Search finds a path and terminates, EECBS sets $lb_i(N) = f_{\min,i}(N)$, which is a lower bound on the cost of a minimum-cost path that satisfies the constraints $\Psi_i(N)$, and sets the threshold $\tau_i(N) = w \cdot lb_i(N)$. Since $h_i(s_i)$ is admissible and the root CT node N_0 does not contain any constraint, a lower bound on the cost of the minimum-cost path is $lb_i(N_0) = h_i(s_i)$ and the threshold is $\tau_i(N_0) = w \cdot lb_i(N_0) = w \cdot h_i(s_i)$.

Since the low-level Focal Search always selects a v-t node n in FOCAL$_L$ at each iteration, $f_{\min,i}(N) \leq f_i(n) \leq \tau_i(N) = w \cdot f_{\min,i}(N)$. When the selected v-t node n is the goal v-t node and the low-level

Focal Search terminates, since EECBS sets $lb_i(N) = f_{\min,i}(N)$ and $\tau_i(N) = w \cdot lb_i(N)$, the path is always individually bounded-suboptimal with its cost $c_i(N)$ satisfying

$$lb_i(N) \leq c_i(N) \leq \tau_i(N) = w \cdot lb_i(N) \leq w \cdot c_i^*(N), \tag{2.5}$$

which results in the SOC of the CT node N satisfying

$$\begin{aligned} &\sum_{i\in[k]} lb_i(N) \leq \sum_{i\in[k]} c_i(N) \leq \sum_{i\in[k]} \tau_i(N) = w \cdot \sum_{i\in[k]} lb_i(N); \\ &LB(N) \leq C(N) \leq w \cdot LB(N) \leq w \cdot (LB(N) + H(N)) = w \cdot F(N). \end{aligned} \tag{2.6}$$

On the high level, since LB is a lower bound on the SOC of the optimal solution, the Selection Rules (E1) and (E2) ensure that the SOC of the selected CT node is at most the bounded-suboptimality factor w larger than the SOC of the optimal solution. On the other hand, according to Inequalities 2.5 and 2.6, the SOC of CT node N_F (i.e., the CT node with the minimum F-value in CLEANUP$_H$) satisfies

$$C(N_F) \leq w \cdot LB(N_F) \leq w \cdot F(N_F) = w \cdot LB, \tag{2.7}$$

which indicates that the Selection Rule (E3) ensures that the SOC of the selected CT node, i.e., N_F, is at most the bounded-suboptimality factor w larger than the SOC of the optimal solution. Thus, by following the Selection Rules (E1), (E2), and (E3), EECBS finds a bounded-suboptimal solution if the paths in the selected CT node are conflict-free.

Algorithm 2.3 (with its tool functions in Algorithm 2.4) shows the pseudo-code of the low-level Focal Search of EECBS. At the beginning, the low-level Focal Search computes the *time horizon* $T_i(N)$ from the constraints $\Psi_i(N)$ and the paths $\dot{P}_i(N) = \{p_j(N) \mid j \in [k] \setminus \{i\}\}$ of the other agents [Line 4]. The time

horizon $T_i(N)$ is defined as the earliest timestep when all the other agents begin to wait at their target vertices permanently, and no more constraints in $\Psi_i(N)$ occur[‡].

The low-level Focal Search first generates a root v-t node n_0, initializes the minimum f_i-value and the threshold $\tau_i(N)$, inserts root v-t node n_0 into OPEN$_L$, and updates FOCAL$_L$ based on OPEN$_L$ [Lines 6-11]. At each iteration, the low-level Focal Search first updates the minimum f_i-value $f_{\min,i}(N)$ over all v-t nodes in OPEN$_L$ and updates FOCAL$_L$ based on OPEN$_L$ [Lines 13-18]. Then, it selects a v-t node $\hat{n}$ in FOCAL$_L$ with the minimum number of conflicts $x_i(\hat{n})$ and checks if it is a goal v-t node [Lines 19-22]. If so, then the low-level Focal Search returns the path extracted by v-t node $\hat{n}$, the cost of the path, and the lower bound on the cost of the minimum-cost path that satisfies the constraints $\Psi_i(N)$ [Lines 22-25]. Otherwise, the low-level Focal Search expands the v-t node $\hat{n}$ and generates its child v-t nodes.

When the low-level Focal Search generates a child v-t node n with state representation (v, t), it checks if v-t node n dominates a v-t node $\bar{n}$ with state representation $(\bar{v}, \bar{t})$ in CLOSED$_L$ or OPEN$_L$. For timesteps at or after the time horizon $T_i(N)$, all the other agents have been waiting at their target vertices permanently, and no constraint occurs. If $t < T_i(N)$, then we say v-t node n *dominates* v-t node $\bar{n}$ iff $v = \bar{v}$, $t = \bar{t}$, and $x_i(n) < x_i(\bar{n})$. Since $h_i(n) = h_i(v) = h_i(\bar{v}) = h_i(\bar{n})$, $t = \bar{t}$ (i.e., $g_i(n) = g_i(\bar{n})$) indicates $f_i(n) = f_i(\bar{n})$. That is, given a path through vertex v resulting from expanding v-t node $\bar{n}$, there is always a path through vertex v resulting from expanding v-t node n with the same cost but fewer conflicts. In this case, the low-level Focal Search is free to prune v-t node $\bar{n}$. On the other hand, given a path through vertex v resulting from expanding v-t node $\bar{n}$, even if there is always a path, say p_n, through vertex v resulting from expanding v-t node n with the same cost but fewer conflicts, the low-level Focal Search does not prune v-t node $\bar{n}$ since there may be a path through vertex v resulting from expanding v-t node $\bar{n}$ that has a larger cost but satisfies the constraints $\Psi_i(N)$ or has fewer conflicts than path p_n. If $t \geq T_i(N)$, then we say v-t node n dominates v-t node $\bar{n}$ iff $v = \bar{v}, \bar{t} \geq T_i(N)$, and either $f_i(n) < f_i(\bar{n})$ or $(f_i(n) = f_i(\bar{n}) \wedge x_i(n) < x_i(\bar{n}))$.

[‡]Since all the other agents begin to wait at their target vertices permanently, no edge constraints occur at or after the time horizon.

Since $h_i(n) = h_i(v) = h_i(\bar{v}) = h_i(\bar{n})$, $f_i(n) < f_i(\bar{n})$ indicates $g_i(n) = t < \bar{t} = g_i(\bar{n})$. That is, given a path through vertex v resulting from expanding v-t node $\bar{n}$, there is always a path through vertex v resulting from expanding v-t node n with a lower cost or with the same cost but fewer conflicts. In this case, the low-level Focal Search is free to prune v-t node $\bar{n}$. We implement dominance checking [Lines 31-36] with the FindNode [Lines 42-50 in Algorithm 2.4] and IsDominant [Lines 51-52 in Algorithm 2.4] functions. If v-t node n dominates $\bar{n}$, then the low-level Focal Search prunes v-t node $\bar{n}$ by replacing $g_i(\bar{n})$, $f_i(\bar{n})$, $x_i(\bar{n})$, $\bar{t}$ and *parent*$(\bar{n})$ with $g_i(n)$, $f_i(n)$, $x_i(n)$, t and *parent*(n), respectively [Line 57 in Algorithm 2.4]. After that, if v-t node $\bar{n}$ is in CLOSED$_L$, then the low-level Focal Search removes it from CLOSED$_L$ and inserts it back into OPEN$_L$ and updates FOCAL$_L$ if needed [Lines 58-60 in Algorithm 2.4]. Otherwise (i.e., v-t node $\bar{n}$ is in OPEN$_L$), the low-level Focal Search updates the priority of v-t node $\bar{n}$ in OPEN$_L$, and updates FOCAL$_L$ if needed [Lines 61-67 in Algorithm 2.4].

Finally, if there is no more v-t node in OPEN$_L$ during the search, then the low-level Focal Search returns "No path" and terminates [Line 37].

2.4.3 Existing Enhancements of EECBS

In this section, we introduce the existing enhancements used to improve the efficiency of EECBS, including bypassing conflicts, symmetry breaking, prioritizing conflicts, and the Weighted Dependency Graph (WDG) heuristic.

2.4.3.1 Bypassing Conflicts

Bypassing conflicts is an enhancement originally used to improve the efficiency of CBS [5]. When CBS expands a CT node $\hat{N}$ and generates a child CT node N, if CT node $\hat{N}$ has the same SOC as CT node N but with fewer conflicts, then CBS replaces the paths in CT node $\hat{N}$ with the paths in CT node N and

discards all generated child CT nodes of CT node $\hat{N}$. Otherwise, CBS keeps all generated child CT nodes of CT node $\hat{N}$.

Li et al. [34] extend bypassing conflicts to EECBS. Since EECBS finds a bounded-suboptimal rather than an optimal solution, the conditions for bypassing conflicts can be relaxed. When EECBS expands a CT node $\hat{N}$ and generates a child CT node N, bypassing conflicts occurs if

(C1) CT node $\hat{N}$ is not selected from CLEANUP$_H$,

(C2) $\forall i \in [k], c_i(N) \leq w \cdot lb_i(\hat{N})$,

(C3) $C(N) \leq w \cdot LB$, and

(C4) $|\mathcal{X}(N)| < |\mathcal{X}(\hat{N})|$.

When EECBS expands the CT node selected from CLEANUP$_H$, i.e., CT node N_F, the new CT node N_F in CLEANUP$_H$ can have a larger F-value than the previous one, which increases LB. According to the rationales of the Selection Rules (E1), (E2), and (E3), increasing LB makes the conditions of Selection Rules (E1) and (E2) more likely to hold in future iterations, thereby helping EECBS find a solution quickly (see Section 2.4.1). Thus, EECBS uses Condition (C1) to prevent bypassing conflicts of CT nodes selected from CLEANUP$_H$ since expanding these CT nodes has a chance to increase LB, and bypassing conflicts would lose that chance. EECBS uses Conditions (C2) to ensure that each path remains individually bounded-suboptimal after bypassing conflicts. EECBS uses (C3) to ensure that CT node N (which replaces the currently expanding CT node $\hat{N}$) will be expanded in the next iteration. EECBS uses condition (C4) to reduce the number of conflicts and avoid repeatedly finding the same set of paths while bypassing conflicts.

Since $lb_i(N)$ is a lower bound on the cost of a minimum-cost path that satisfies the set of constraints $\Psi_i(N)$, which can be different from $\Psi_i(\hat{N})$ (depending on the constraints that are used to resolve the selected conflict), EECBS does not replace the lower bounds $lb_i(\hat{N})$ with $lb_i(N)$ when bypassing conflicts. In

this dissertation, due to Flex Distribution, we can relax Condition (C2) of bypassing conflicts, as described in Section 3.5.

2.4.3.2 Symmetry Breaking

Symmetry breaking is an enhancement originally used to improve the efficiency of CBS [31], where *(pairwise) symmetry* occurs when two agents have many different paths to their target vertices, but every pairwise combination of the paths results in a conflict. By using symmetry breaking, CBS avoids multiple CT node expansions that exhaustively examine all pairwise combinations of the paths.

Li et al. [31] classify vertex/edge conflicts into different types. The first type is called a corridor conflict, which occurs when two agents traverse a corridor in opposite directions. The second type is called a target conflict, which occurs when one agent a_j traverses the target vertex l_i of the other agent a_i, while agent a_i waits at its target vertex l_i permanently. For each type of conflict, Li et al. [31] design constraints to resolve it. To resolve a corridor conflict, Li et al. [31] prevent one agent from entering the corridor until the other agent has traversed it. To resolve a target conflict, Li et al. [31] distinguish two cases. The first case is that agent a_i (the agent that waits at its target vertex) has to find a new path that allows agent a_j to traverse the target vertex l_i without any conflict. The second case is that agent a_i is allowed to wait at its target vertex l_i permanently, and all the other agents $\{a_m \mid m \in [k] \setminus \{i\}\}$ (including a_j) should avoid collisions with it, which can require finding paths for more than one agent (see Section 4.4).

Li et al. [34] extend symmetry breaking to EECBS[§]. In this dissertation, we use symmetry breaking for corridor conflicts and target conflicts to enhance the efficiency of EECBS and combine it with Flex Distribution, as described in Section 4.4.

[§]Li et al. [34] also classify conflicts into a type called rectangle conflict. However, it is less effective and thus is omitted in this dissertation.

2.4.3.3 Prioritizing Conflicts

Prioritizing conflicts is an enhancement originally used to improve the efficiency of CBS [5]. When CBS selects a CT node for expansion, prioritizing conflicts determines which conflict to resolve.

Li et al. [34] extend prioritizing conflicts to EECBS. A conflict can be classified into three types: cardinal, semi-cardinal, and non-cardinal. Given a CT node $\hat{N}$ selected for expansion, to classify a conflict between two agents a_i and a_j that follow paths $p_i(\hat{N})$ and $p_j(\hat{N})$, respectively, Li et al. [34] consider how CBS resolves the conflict. This involves expanding the CT node $\hat{N}$ and generating two child CT nodes N and N'. A conflict is *cardinal* iff both $\sum_{i\in[k]} c_i^*(N)$ and $\sum_{i\in[k]} c_i^*(N')$ are larger than $\sum_{i\in[k]} c_i^*(\hat{N})$. A conflict is *semi-cardinal* iff exactly one of $\sum_{i\in[k]} c_i^*(N)$ or $\sum_{i\in[k]} c_i^*(N')$ is larger than $\sum_{i\in[k]} c_i^*(\hat{N})$. A conflict is *non-cardinal* iff neither $\sum_{i\in[k]} c_i^*(N)$ nor $\sum_{i\in[k]} c_i^*(N')$ is larger than $\sum_{i\in[k]} c_i^*(\hat{N})$.

However, prioritizing conflicts can result in a large runtime overhead. Thus, Li et al. [34] apply prioritizing conflict according to the following conditions:

(1) if the CT node is selected from CLEANUP$_H$, i.e., CT node N_F is selected for expansion, then all the conflicts in the CT node are classified.

(2) otherwise, for each conflict in the selected CT node, if at least one of the paths in which the conflict occurs has its cost equal to its lower bound, then the conflict is classified.

Condition (1) exists since resolving cardinal conflicts tends to generate child CT nodes with larger SOLBs than those of their parent CT node N_F. After EECBS expands CT node N_F, the new CT node with the minimum F-value in CLEANUP$_H$ may have an F-value larger than $F(N_F)$. In this case, by prioritizing conflicts when CT node N_F is selected for expansion, EECBS tends to increase LB. According to the rationales of the Selection Rules (E1), (E2), and (E3), increasing LB makes the conditions of Selection Rules (E1) and (E2) more likely to hold in future iterations, thereby helping EECBS find a solution quickly (see Section 2.4.1). In contrast, when CT node N_F is selected for expansion, without prioritizing conflicts,

EECBS may select a conflict that results in generating two child CT nodes with the same SOLB as CT node N_F. In this case, the lower bound LB remains unchanged after EECBS expands CT node N_F. Condition (2) exists since resolving cardinal conflicts can increase the costs of paths of the agents in this case, thereby making the SOCs of the generated child CT nodes closer to the SOCs of the solutions in their subtrees.

EECBS thus selects conflicts in the order of cardinal, semi-cardinal, non-cardinal, and unclassified conflicts, breaking ties in the order of target conflicts, corridor conflicts, and other conflicts. For example, EECBS selects a semi-cardinal conflict only when no cardinal conflict exists, and selects a non-cardinal conflict only when no cardinal or semi-cardinal conflict exists.

2.4.3.4 Weighted Dependency Graph Heuristic

The *Weighted Dependency Graph* (WDG) heuristic is an enhancement originally used to improve the efficiency of CBS [30]. It is an admissible heuristic for the high-level A* of CBS.

Li et al. [34] apply the WDG heuristic to EECBS, which remains an admissible heuristic. Given a CT node N, Li et al. [34] construct a Weighted Dependency Graph (WDG), which is a weighted undirected graph

$$G_{WD}(N) = (V_{WD}(N), E_{WD}(N), W_{WD}(N)), \tag{2.8}$$

where each vertex $v_i(N) \in V_{WD}(N)$ represents an agent a_i, each edge $e_{i,j}(N) = \{v_i(N), v_j(N)\} \in E_{WD}(N)$ represents that at least one conflict exists between agents a_i and a_j when they follow their respective paths $p_i(N)$ and $p_j(N)$, and the weight $w_{i,j}(N) \in W_{WD}(N)$ of each edge $e_{i,j}(N)$ equals

$$w_{i,j}(N) = C^*_{i,j}(N) - c^*_i(N) - c^*_j(N), \tag{2.9}$$

where $C^*_{i,j}(N)$ is the minimum SOC of conflict-free paths for agents a_i and a_j that satisfy the constraints $\Psi_i(N)$ and $\Psi_j(N)$, respectively; $c^*_i(N)$ is the cost of the minimum-cost path for agent a_i that satisfies

the constraints $\Psi_i(N)$, and $c_j^*(N)$ is the cost of the minimum-cost path for agent a_j that satisfies the constraints $\Psi_j(N)$. Then, to compute the WDG heuristic, Li et al. [34] transform the WDG to an edge-weighted minimum vertex cover problem [30] that consists of a list of non-negative integers $[x_i \mid i \in [k]]$, one for each vertex $v_i(N) \in V_{WD}(N)$, and a set of constraints $x_i + x_j \geq w_{i,j}(N)$, one for each edge $e_{i,j}(N)$. The objective is to identify $x_1, \ldots, x_k$ that minimize $\sum_{i \in [k]} x_i$ and satisfy the constraints in the problem, which equals the WDG heuristic value $H(N)$ of CT node N. That is, the WDG heuristic provides a non-overestimation of the additional cost that each pair of conflicting agents will contribute to the SOC of a minimum-cost solution that satisfies the constraints $\Psi(N)$.

Since computing the WDG heuristic values for CT nodes results in large runtime overhead, Li et al. [34] compute them lazily. That is, rather than computing the WDG heuristic value for a CT node when it is generated, Li et al. [34] compute the WDG heuristic value for a CT node when it is selected. Moreover, Li et al. only compute the WDG heuristic value for a CT node that is selected by the Selection Rule (E3) (i.e., CT node N_F) since the purpose of computing the WDG heuristic is to improve LB. According to the rationales of the Selection Rules (E1), (E2), and (E3), increasing LB makes the conditions of Selection Rules (E1) and (E2) more likely to hold in future iterations, thereby helping EECBS find a solution quickly (see Section 2.4.1). After computing the WDG heuristic value for the selected CT node, Li et al. [34] insert it back into CLEANUP$_H$ and OPEN$_H$, and update FOCAL$_H$ based on OPEN$_H$.

However, in this dissertation, our MAPF instances contain more agents than those in Li et al. [34]. We found that computing the WDG heuristic values for CT nodes selected by the Selection Rule (E3) can still result in a large runtime overhead, slowing down EECBS (with or without Flex Distribution). Thus, in this dissertation, we omit the implementation of the WDG heuristic. However, we show in Section 3.3 that Flex Distribution can be used in EECBS with any admissible heuristic on the high level, including the WDG heuristic, and is guaranteed to find bounded-suboptimal solutions.

Algorithm 2.1 High-Level EES

1: **procedure** EES(a MAPF instance with k agents, bounded-suboptimality factor w)
2: CLEANUP$_H$, OPEN$_H$, FOCAL$_H$ $\leftarrow$ empty list
3: Generate the root CT node N_0 with $\Psi(N_0) = [\Psi_i(N_0) = \emptyset \mid i \in [k]]$
4: $\dot{P}(N_0) \leftarrow \emptyset$
5: **for** $i \in [k]$ **do** ▷ *Find initial paths without constraint.*
6: $(p_i(N_0), c_i(N_0), lb_i(N_0)) \leftarrow$ FocalSearch(s_i, l_i, $\emptyset$, $\dot{P}(N_0)$, 0)
7: $\dot{P}(N_0) \leftarrow \dot{P}(N_0) \cup \{p_i(N_0)\}$
8: $\mathcal{X}(N_0) \leftarrow$ Find all conflicts in paths $[p_i(N_0) \mid i \in [k]]$.
9: $H(N_0) \leftarrow$ Compute the WDG Heuristic of CT node N_0
10: $\bar{\epsilon_d}, \bar{\epsilon_h} \leftarrow 0$
11: InsertNode(N_0, CLEANUP$_H$, OPEN$_H$, FOCAL$_H$)
12: **while** CLEANUP$_H$ **not** empty **do**
13: $LB \leftarrow \min_{N \in \text{CLEANUP}_H} F(N)$ ▷ $LB = F(N_F)$.
14: $\hat{N} \leftarrow$ SelectNode(CLEANUP$_H$, OPEN$_H$, FOCAL$_H$)
15: **if** $|\mathcal{X}(\hat{N})| = 0$ **then return** $[p_i(\hat{N}) | i \in [k]]$
16: **if** $\hat{N}$ is selected from CLEANUP$_H$ **and** $H(\hat{N})$ is null **then**
17: $H(\hat{N}) \leftarrow$ Compute the WDG Heuristic of CT node $\hat{N}$; $F(\hat{N}) \leftarrow LB(\hat{N}) + H(\hat{N})$
18: InsertNode($\hat{N}$, CLEANUP$_H$, OPEN$_H$, FOCAL$_H$)
19: **continue**
20: *conflict* $\leftarrow$ SelectConflict($\mathcal{X}(\hat{N})$)
21: Generate the two sets of constraint Ψ_1 and Ψ_2 for resolving *conflict*
22: *children* $\leftarrow \emptyset$; $m \leftarrow 1$
23: **while** $m \leq 2$ **do**
24: $N \leftarrow$ InitializeChildNode($\hat{N}$, Ψ_m)
25: **for** $i \in [k]$ **do**
26: **if** $p_i(N)$ violates constraints $\Psi_i(N)$ **then**
27: $\dot{P}_i(N) \leftarrow \{p_j(N) \mid j \in [k] \setminus \{i\}\}$
28: **if** Flex Distribution is used **then**
29: $(p_i(N), c_i(N), lb_i(N)) \leftarrow$ FocalSearchWithFlex(s_i, l_i, $\Psi_i(N)$, $\dot{P}_i(N)$, $lb_i(\hat{N})$)
30: **else**
31: $(p_i(N), c_i(N), lb_i(N)) \leftarrow$ FocalSearch(s_i, l_i, $\Psi_i(N)$, $\dot{P}_i(N)$, $lb_i(\hat{N})$)
32: **if** $p_i(N)$ is not found **then**
33: $m \leftarrow m + 1$; Go to Line 23
34: $\mathcal{X}(N) \leftarrow$ Find all the conflicts in paths $[p_i(N) \mid i \in [k]]$
35: $C(N) \leftarrow \sum_{i \in [k]} c_i(N)$; $LB(N) \leftarrow \sum_{i \in [k]} lb_i(N)$; $F(N) \leftarrow LB(N)$; $H(N) \leftarrow$ null
36: **if** CT node N satisfies conditions (C1-C4) for bypassing conflicts **then** ▷ *See Section 2.4.3.1.*
37: **for** $i \in [k]$ **do**
38: $p_i(\hat{N}) \leftarrow p_i(N)$; $c_i(\hat{N}) \leftarrow c_i(N)$
39: $C(\hat{N}) \leftarrow \sum_{i \in [k]} c_i(N)$; $\mathcal{X}(\hat{N}) \leftarrow \mathcal{X}(N)$
40: Go to Line 15
41: *children* $\leftarrow$ *children* $\cup \{N\}$; $m \leftarrow m + 1$
42: **for** $N \in$ *children* **do**
43: $\hat{H}(N) \leftarrow \bar{\epsilon_h}/(1 - \bar{\epsilon_d}) \cdot |\mathcal{X}(N)|$
44: InsertNode(N, CLEANUP$_H$, OPEN$_H$, FOCAL$_H$)
45: Compute $\epsilon_d(\hat{N})$ and $\epsilon_h(\hat{N})$; update $\bar{\epsilon_d}$ and $\bar{\epsilon_h}$
46: **return** No solution

Algorithm 2.2 Tool Functions for High-Level EES

procedure InsertNode(N, CLEANUP$_H$, OPEN$_H$, FOCAL$_H$)
  Insert CT node N into CLEANUP$_H$ and OPEN$_H$
  for $N' \in$ OPEN$_H$ **do**
    if $\hat{F}(N') \leq w \cdot \hat{F}(N_{\hat{F}})$ **then** Insert CT node N' into FOCAL$_H$

procedure SelectNode(CLEANUP$_H$, OPEN$_H$, FOCAL$_H$)
  if $C(N_{\mathcal{X}}) \leq w \cdot LB$ **then**
    Remove CT node $N_{\mathcal{X}}$ from CLEANUP$_H$, OPEN$_H$, and FOCAL$_H$
    return $N_{\mathcal{X}}$
  if $C(N_{\hat{F}}) \leq w \cdot LB$ **then**
    Remove CT node $N_{\hat{F}}$ from CLEANUP$_H$, OPEN$_H$, and FOCAL$_H$
    return $N_{\hat{F}}$
  Remove CT node N_F from CLEANUP$_H$, OPEN$_H$
  if N_F **in** FOCAL$_H$ **then**
    Remove CT node N_F from FOCAL$_H$
  return N_F

procedure SelectConflict($\mathcal{X}(\hat{N})$)
  for *conflict* $\in \mathcal{X}(\hat{N})$ **do**
    $(p_i(\hat{N}), p_j(\hat{N})) \leftarrow$ pair of paths that *conflict* occurs
    if $\hat{N}$ is expanded from CLEANUP$_H$ **or** $c_i(\hat{N}) = lb_i(\hat{N})$ **or** $c_j(\hat{N}) = lb_j(\hat{N})$ **then**
      Classify *conflict* into cardinal, semi-cardinal, and non-cardinal
    Determine *conflict* as either a target conflict, a corridor conflict, or none of them above.
  return the conflict with the highest priority in $\mathcal{X}(\hat{N})$

procedure InitializeChildNode($\hat{N}$, Ψ)
  $N \leftarrow$ a copy of CT node $\hat{N}$
  for $\psi \in \Psi$ **do**
    $a_i \leftarrow$ agent that constraint ψ is applied to
    $\Psi_i(N) \leftarrow \Psi_i(N) \cup \{\psi\}$
  return N

Algorithm 2.3 Low-Level Focal Search

procedure FocalSearch(s_i, l_i, $\Psi_i(N)$, $\dot{P}_i(N)$, $lb_i(\hat{N})$)
  ▷ *$\hat{N}$ is the parent CT node of CT node N, so $lb_i(\hat{N}) = 0$ if N is the root CT node.* ◁
  ▷ $\dot{P}_i(N) = \{p_j(N) \mid j \in [k] \setminus \{i\}\}$. ◁
  $T_i(N) \leftarrow$ Compute the time horizon from constraints $\Psi_i(N)$ and paths $\dot{P}_i(N)$ ▷ *See Section 2.4.2.*
  Determine the priority function value $\tilde{f}_i(n)$ of v-t node n
  Generate the root v-t node n_0 with state representation $(s_i, 0)$
  $g_i(n_0) \leftarrow 0$; $h_i(n_0) \leftarrow h_i(s_i)$; $f_i(n_0) \leftarrow g_i(n_0) + h_i(n_0)$; $\tilde{h}_i(n_0) \leftarrow \tilde{h}_i(s_i)$; $\tilde{f}_i(n_0) \leftarrow g_i(n_0) + \tilde{h}_i(n_0)$; $x_i(n_0) \leftarrow 0$
  $f_{\min,i}(N) \leftarrow f_i(n_0)$
  $\tau_i(N) \leftarrow w \cdot f_{\min,i}(N)$
  OPEN$_L$, FOCAL$_L$, CLOSED$_L \leftarrow$ empty list
  InsertNode(n_0, $\tau_i(N)$, OPEN$_L$, FOCAL$_L$)
  while OPEN$_L$ is **not** empty **do**
    $n_f \leftarrow$ v-t node with the minimum f_i-value in OPEN$_L$
    if $f_{\min,i}(N) < f_i(n_f)$ **then**
      $f_{\min,i}(N) \leftarrow f_i(n_f)$
      $\tau_{new} \leftarrow w \cdot f_{\min,i}(N)$
      UpdateFocal($\tau_i(N)$, τ_{new}, OPEN$_L$, FOCAL$_L$)
      $\tau_i(N) \leftarrow \tau_{new}$
    $\hat{n} \leftarrow$ v-t node with the minimum x_i-value in FOCAL$_L$
    Remove v-t node $\hat{n}$ from OPEN$_L$ and FOCAL$_L$
    Insert $\hat{n}$ into CLOSED$_L$
    if IsTargetReached($\hat{n}$, l_i, $\Psi_i(N)$) **then**
      $lb_i \leftarrow f_{\min,i}(N)$
      $p_i \leftarrow$ Extract the path by back-tracking v-t node $\hat{n}$; $c_i \leftarrow$ cost of path p_i
      return (p_i, c_i, lb_i)
    neighbors $\leftarrow$ FindNeighbors($\hat{n}$) ▷ *Find the neighboring states.*
    for $(v, t) \in$ *neighbors* **do**
      if (v, t) does **not** satisfy constraints $\Psi_i(N)$ **then**
        continue
      $n \leftarrow$ GenerateChildNode($\hat{n}$, v, t, $\dot{P}_i(N)$)
      $\bar{n} \leftarrow$ FindNode(n, $T_i(N)$, CLOSED$_L$, OPEN$_L$)
      if $\bar{n}$ is null **then**
        InsertNode(n, $\tau_i(N)$, OPEN$_L$, FOCAL$_L$)
        continue
      if IsDominant(n, $\bar{n}$) **then**
        UpdatePriority(n, $\bar{n}$, $\tau_i(N)$, CLOSED$_L$, OPEN$_L$, FOCAL$_L$)
  return No path

Algorithm 2.4 Tool Functions for the Low-Level Focal Search

procedure InsertNode(n, $\tau_i(N)$, OPEN$_L$, FOCAL$_L$)
    Insert v-t node n into OPEN$_L$
    Sort v-t nodes in OPEN$_L$ in increasing order of f_i-values, breaking ties in favor of v-t nodes with smaller x_i-values
    if $f_i(n) \leq \tau_i(N)$ **then**
        Insert v-t node n into FOCAL$_L$
        Sort v-t nodes in FOCAL$_L$ in increasing order of x_i-values, breaking ties in favor of v-t nodes with smaller $\tilde{f}_i$-values

procedure UpdateFocal(τ_{old}, τ_{new}, OPEN$_L$, FOCAL$_L$)
    for v-t node $n \in$ OPEN$_L$ **do**
        if $\tau_{old} < f_i(n) \leq \tau_{new}$ **then**
            Insert v-t node n into FOCAL$_L$

procedure IsTargetReached(n, l_i, $\Psi_i(N)$)
    $(v, t) \leftarrow$ state representation of v-t node n
    if $v \neq l_i$ **then return** false
    for $\langle a_i, l_i, t_\psi \rangle \in \Psi_i(N)$ **do**
        if $t \leq t_\psi$ **then**
            return false
    return true

procedure FindNeighbors(n)
    $(v, t) \leftarrow$ state representation of v-t node n
    neighbors $\leftarrow \{(v, t+1)\}$ ▷ *Insert the wait action.*
    $V' \leftarrow$ Find the successor vertex for each possible move action at vertex v
    for $v' \in V'$ **do**
        neighbors $\leftarrow$ *neighbors* $\cup \{(v', t+1)\}$ ▷ *For the move action.*
    return *neighbors*

procedure CountConflicts(n, $\dot{P}_i(N)$)
    $(v, t) \leftarrow$ state representation of v-t node n
    $x_v \leftarrow$ number of vertex conflicts with paths $\dot{P}_i(N)$ at vertex v at timestep t
    if n is **not** the root v-t node **then**
        $\hat{n} \leftarrow$ parent of v-t node n with state representation $(u, t-1)$
        if $u \neq v$ **then**
            $x_e \leftarrow$ number of edge conflicts with paths $\dot{P}_i(N)$ when agent a_i moves from vertex u at timestep $t-1$ to vertex v at timestep t
    return $x_v + x_e$

procedure GENERATECHILDNODE($\hat{n}$, v, t, $\dot{P}_i(N)$)
  ▷ *v-t node $\hat{n}$ has state representation $(\hat{v}, \hat{t})$.* ◁
  Generate v-t node n with state representation (v, t)
  parent(n) $\leftarrow \hat{n}$
  $g_i(n) \leftarrow g_i(\hat{n}) + 1$ ▷ *Unit-cost.*
  $h_i(n) \leftarrow h_i(v)$; $f_i(n) \leftarrow g_i(n) + h_i(n)$
  $\tilde{h}_i(n) \leftarrow \tilde{h}_i(v)$; $\tilde{f}_i(n) \leftarrow g_i(n) + \tilde{h}_i(n)$
  $x_i(n) \leftarrow x_i(\hat{n})+$ COUNTCONFLICTS(n, $\dot{P}_i(N)$)
  return n

procedure FINDNODE(n, $T_i(N)$, CLOSED$_L$, OPEN$_L$)
  $(v, t) \leftarrow$ state representation of v-t node n
  LIST $\leftarrow$ concatenate CLOSED$_L$ and OPEN$_L$
  $\bar{n} \leftarrow$ null
  if $t < T_i(N)$ **then**
    $\bar{n} \leftarrow$ Find v-t node in *LIST* with state representation (v, t)
  else
    $\bar{n} \leftarrow$ Find v-t node in *LIST* with state representation $(v, \bar{t})$ with $\bar{t} \geq T_i(N)$
  return $\bar{n}$

procedure ISDOMINANT(n_1, n_2)
  return $f_i(n_1) < f_i(n_2)$ **or** $(f_i(n_1) = f_i(n_2)$ **and** $x_i(n_1) < x_i(n_2))$

procedure UPDATEPRIORITY(n, $\bar{n}$, $\tau_i(N)$, CLOSED$_L$, OPEN$_L$, FOCAL$_L$)
  $(v, t) \leftarrow$ state representation of v-t node n
  $(v, \bar{t}) \leftarrow$ state representation of v-t node $\bar{n}$
  $\bar{f}_i \leftarrow f_i(\bar{n})$
  $g_i(\bar{n}) \leftarrow g_i(n)$; $f_i(\bar{n}) \leftarrow f_i(n)$; $\tilde{f}_i(\bar{n}) \leftarrow \tilde{f}_i(n)$; $x_i(\bar{n}) \leftarrow x_i(n)$; $\bar{t} \leftarrow t$; *parent*($\bar{n}$) $\leftarrow$ *parent*(n)
  if $\bar{n} \in$ CLOSED$_L$ **then**
    Remove v-t- node $\bar{n}$ from CLOSED$_L$
    INSERTNODE($\bar{n}$, $\tau_i(N)$, OPEN$_L$, FOCAL$_L$)
  else ▷ *$\bar{n}$ is in OPEN$_L$*
    Update the priority of $\bar{n}$ in OPEN$_L$
    if $f_i(\bar{n}) \leq \tau_i(N)$ **then**
      if $\bar{f}_i > \tau_i(N)$ **then**
        Insert v-t node $\bar{n}$ into FOCAL$_L$
      else
        Update the priority of v-t node $\bar{n}$ in FOCAL$_L$

Chapter 3

Flex Distribution

To improve the efficiency of EECBS, we introduce our new approach, called *Flex Distribution* [9], which relaxes the requirement that the paths of agents be individually bounded-suboptimal while maintaining the guarantee of finding a bounded-suboptimal solution to a MAPF instance if a solution exists. Given a path that satisfies constraints in a CT node, we define its flex as the difference between the threshold, which is w times the lower bound on the cost of a minimum-cost path that satisfies the constraints, and the cost. When EECBS runs the low-level Focal Search to find a path for an agent, the core idea of Flex Distribution is to increase its threshold by using the flex of the paths of the other agents. With the increase in the threshold, EECBS might now be able to find paths with fewer conflicts on the low level (see Figure 3.12). Additionally, since Flex Distribution no longer requires each agent's path to be individually bounded-suboptimal, EECBS can relax Condition (C2) for bypassing conflicts on the high level, which might allow it to terminate more quickly. Yet, the solutions found by EECBS with Flex Distribution are still guaranteed to be bounded-suboptimal*. To distribute flex, we propose an intuitive mechanism called Greedy Flex Distribution (GFD). When EECBS uses the low-level Focal Search to find a path for an agent, GFD uses all the flex of the paths of the other agents to increase the threshold of the agent. Our empirical evaluation

*Since Flex Distribution only modifies the low-level Focal Search and the high-level bypassing condition, it can also be applied to other Conflict-Based Framework instantiations that solve MAPF bounded-suboptimally, e.g., Enhanced Conflict-Based Search (ECBS) [8]. Also, we could apply Flex Distribution to other domains that use the Conflict-Based Framework, such as Virtual Network Embedding [75].

shows that GFD can accelerate the search of EECBS for large-scale MAPF instances. Our contributions are:

- We propose Flex Distribution, a new approach to accelerating EECBS.
- We show how Flex Distribution can be used to find paths on the low level and relax Condition (C2) for bypassing conflicts on the high level. We prove that EECBS with Flex Distribution is guaranteed to find a bounded-suboptimal solution to a MAPF instance if a solution exists.
- We propose Greedy Flex Distribution (GFD) as a mechanism for determining the distributed flex. Our empirical result shows that for the 120-second runtime limit and the bounded-suboptimality factor $w = 1.01$, using GFD results in a more than twofold increase in the success rate of EECBS over all MAPF instances on all graphs, raising it from 0.18 to 0.396. In particular, while EECBS has the success rate of 0.08 over all MAPF instances on the `city` graph (which is the largest graph in our empirical evaluation) for the bounded-suboptimality factor $w = 1.01$, EECBS with GFD has the success rate of 0.52.

3.1 Bounded-Suboptimality of EECBS

When EECBS expands a CT node $\hat{N}$, it selects a conflict and resolves it by generating two child CT nodes with the additional constraints. Suppose that one of the child CT nodes N contains the constraints Ψ_i that are used to resolve the selected conflict, i.e., $\Psi_i(N) = \Psi_i(\hat{N}) \cup \Psi_i$. Since imposing additional constraints can only increase the cost of an optimal path that satisfies the constraints, the lower bound $lb_i(\hat{N})$ in CT node $\hat{N}$ remains a lower bound on the cost of an optimal path that satisfies the constraints $\Psi_i(N)$ in CT node N. That is, when running the low-level Focal Search, both $lb_i(\hat{N})$ and $f_{\min,i}(N)$ are lower bounds on the cost of an optimal path that satisfies the constraints $\Psi_i(N)$. Thus, rather than setting the threshold

$\tau_i(N) = w \cdot f_{\min,i}(N)$ as in [34] (see Lines 9 and 16 in Algorithm 2.3) during the search of the low-level Focal Search, we set

$$\tau_i(N) = w \cdot \max\{lb_i(\hat{N}), f_{\min,i}(N)\}. \tag{3.1}$$

When the low-level Focal Search finds a path and terminates, rather than setting the lower bound $lb_i(N) = f_{\min,i}(N)$ as Li et al. [34] (see Line 23 in Algorithm 2.3), we set

$$lb_i(N) = \max\{lb_i(\hat{N}), f_{\min,i}(N)\}. \tag{3.2}$$

In this case, the path found by the low-level Focal Search remains individually bounded-suboptimal. Thus, the SOC of CT node N still satisfies Equation 2.6. Also, for any CT node $\acute{N}$ in the subtree of CT node N, $lb_i(N) \leq lb_i(\acute{N})$, and thus $LB(N) \leq LB(\acute{N})$. On the other hand, for the root CT node N_0 that does not have a parent CT node, since it does not contain any constraint, we follow Li et al. [34] and set $\tau_i(N_0) = w \cdot f_{\min,i}(N_0) = w \cdot h_i(s_i)$ and $lb_i(N_0) = f_{\min,i}(N_0) = h_i(s_i)$, where $h_i(s_i)$ is the cost of the minimum-cost path from the start vertex s_i to the target vertex l_i of agent a_i.

Definition 3.1 (Individually Bounded-Suboptimal Path)**.** *A path of an agent a_i in a CT node N is **individually bounded-suboptimal** iff its cost $c_i(N)$ satisfies*

$$c_i(N) \leq \tau_i(N) = w \cdot \max\{lb_i(\hat{N}), f_{\min,i}(N)\} = w \cdot lb_i(N), \tag{3.3}$$

We provide the following definition for a CT node N with its SOC $C(N) = \sum_{i\in[k]} c_i(N)$ and SOLB $LB(N) = \sum_{i\in[k]} lb_i(N)$.

Definition 3.2 (Local Suboptimality)**.** *The **local suboptimality** of a CT node N is $C(N)/LB(N)$.*

Definition 3.3 (Locally Bounded-Suboptimal CT Node). *A CT node N is **locally bounded-suboptimal** iff its local suboptimality is at most the bounded-suboptimality factor w, i.e., $C(N)/LB(N) \leq w$ or, equivalently, $C(N) \leq w \cdot LB(N)$.*

By requiring each path in each CT node to be individually bounded-suboptimal, EECBS ensures that each CT node is locally bounded-suboptimal.

At each iteration on the high level, EECBS updates the lower bound $LB = F(N_F) = LB(N_F) + H(N_F)$ on the SOC of the optimal solution and then selects a CT node based on Selection Rules (E1), (E2), and (E3) (see Section 2.4.1). We thus provide the following definitions:

Definition 3.4 (Global Suboptimality). *The **global suboptimality** of a CT node N is $C(N)/LB$.*

Definition 3.5 (Globally Bounded-Suboptimal CT Node). *A CT node N is **globally bounded-suboptimal** iff its global suboptimality is at most the bounded-suboptimality factor w, i.e., $C(N)/LB \leq w$ or, equivalently, $C(N) \leq w \cdot LB$.*

Equivalently, the condition of Selection Rules (E1) and (E2) is to check if CT nodes $N_{\mathcal{X}}$ and $N_{\hat{F}}$, respectively, are globally bounded-suboptimal. We can combine these definitions with Selection Rules (E1), (E2), and (E3) to establish the following theorem:

Theorem 3.1. *If each CT node is locally bounded-suboptimal, then the solution found by EECBS is guaranteed to be bounded-suboptimal.*

Proof. Given a CT node N, since its heuristic $H(N)$ is admissible, $F(N) = LB(N) + H(N)$ is the lower bound on the SOC of the minimum-SOC solution that satisfies constraints $\Psi(N)$. According to Lemma 2.1, as we define $N_F = \arg\min_{N \in \text{CLEANUP}_H} F(N)$, $F(N_F)$ (i.e., the minimum F-value over all CT nodes in CLEANUP$_H$) is a lower bound on the SOC C^* of an optimal solution. EECBS maintains $LB = F(N_F)$ during the search, i.e.,

$$LB = F(N_F) = LB(N_F) + H(N_F) \leq C^*. \tag{3.4}$$

At each iteration, EECBS uses Selection Rules (E1), (E2), and (E3) to select a CT node. If EECBS selects the CT node $\hat{N}$ according to selection rules (E1) and (E2), i.e., $\hat{N} \in \{N_{\mathcal{X}}, N_{\hat{F}}\}$, then the SOC of each selected CT node N satisfies

$$C(N) \leq w \cdot LB \leq w \cdot C^*. \tag{3.5}$$

On the other hand, if EECBS selects a CT node $\hat{N} = N_F$ by the Selection Rule (E3), then since each CT node is locally bounded-suboptimal,

$$C(N_F) \leq w \cdot LB(N_F) \leq w \cdot F(N_F) = w \cdot LB \leq w \cdot C^*. \tag{3.6}$$

Thus, by Inequalities 3.5 and 3.6, if the paths in the selected CT node are conflict-free, i.e., a solution, then EECBS returns a bounded-suboptimal solution. □

3.2 Limitations of EECBS

EECBS ensures that the solution it finds is bounded-suboptimal according to Theorem 3.1. However, it is unnecessary for each path to be individually bounded-suboptimal to ensure that each CT node is locally bounded-suboptimal. That is, the SOC of a CT node N can satisfy $C(N) \leq w \cdot LB(N)$ without each path being individually bounded-suboptimal. Thus, we propose *Flex Distribution*, which relaxes the individual bounded-suboptimality of each path while still guaranteeing that the SOC of each CT node is locally bounded-suboptimal. This ensures that the solution found by EECBS with Flex Distribution remains bounded-suboptimal. In general, the definition of bounded-suboptimal solutions depends on the **sum** of the costs of all paths rather than the cost of each individual path (see Section 2.2). Thus, given a CT node, we can apply Flex Distribution to increase the thresholds of its paths as long as the resulting SOC is at most the bounded-suboptimality factor w larger than the SOLB. We use the following premises to explain how EECBS with Flex Distribution works.

(1) At the beginning of the search, suppose that EECBS generates the root CT N_0 node with one individually bounded-suboptimal path for each agent, i.e., $\forall i \in [k]$, $c_i(N_0) \leq w \cdot lb_i(N_0)$. Since the root CT node N_0 does not contain any constraint, $\forall i \in [k]$, $lb_i(N_0) = f_{\min,i}(N_0) = h_i(s_i)$, where $h_i(s_i)$ is the cost of the minimum-cost path from the start vertex s_i to the target vertex l_i of agent a_i (see Section 2.4.2). That is, in this premise, EECBS with Flex Distribution does not use Flex Distribution when finding paths for agents in the root CT node N_0.

(2) Also, at an iteration, suppose that EECBS with Flex Distribution expands a CT nodes $\hat{N}$ and generates one of its child CT nodes N with the set of constraints $\Psi_i(N) = \Psi_i(\hat{N}) \cup \Psi_i$ for agent a_i, where Ψ_i is the set of constraints that are used to resolve the selected conflict.

3.3 Definition of Flex

To generate one of the child CT nodes N during the expansion of a CT node $\hat{N}$, EECBS needs to find a path for an agent a_i that satisfies the constraints $\Psi_i(N) = \Psi_i(\hat{N}) \cup \Psi_i$. Since Ψ_i only affects agent a_i, the paths of the other agents $\{a_j \mid j \in [k] \setminus \{i\}\}$ remain unchanged, i.e., $\forall j \in [k] \setminus \{i\}, p_j(N) = p_j(\hat{N}), lb_j(N) = lb_j(\hat{N}), c_j(N) = c_j(\hat{N})$. When EECBS with Flex Distribution uses the low-level Focal Search to find the path for agent a_i, it increases the threshold by using the sum of differences between the thresholds $w \cdot lb_j(N)$ and the costs of the paths $c_j(N)$ from the other $k-1$ agents.

Definition 3.6 (Flex). *The **flex** of the path of an agent a_i in a CT node N is defined as $w \cdot lb_i(N) - c_i(N)$, where $c_i(N)$ is the cost of the path and $lb_i(N)$ is the lower bound on the cost of the minimum-cost path that satisfies the constraints $\Psi_i(N)$.*

Definition 3.7 (Distributed Flex). *The **distributed flex** $\Delta_i(N)$ is the amount of flex from the paths of the other agents $\{a_j \mid j \in [k] \setminus \{i\}\}$ used to find the path for agent a_i that satisfies the constraints $\Psi_i(N)$ in a CT node N.*

3.4 Finding Paths with Flex Distribution

When EECBS with Flex Distribution finds a path for agent a_i that satisfies the constraints $\Psi_i(N)$ in a CT node N, to ensure that $C(N) \leq w \cdot LB(N)$, the maximum amount of flex allowed to be used is the sum of flex from the paths of the other agents $\{a_j \mid j \in [k] \setminus \{i\}\}$.

Definition 3.8 (Maximum Allowed Flex)**.** *When EECBS finds a path for agent a_i that satisfies the constraints $\Psi_i(N)$ in a CT node N, the paths of the other agents $\{a_j \mid j \in [k] \setminus \{i\}\}$ remain unchanged, i.e.,*

$$\forall j \in [k] \setminus \{i\}, p_j(N) = p_j(\hat{N}), lb_j(N) = lb_j(\hat{N}), \textit{and } c_j(N) = c_j(\hat{N}), \tag{3.7}$$

where CT node $\hat{N}$ is the parent of CT node N. In this case, the ***maximum allowed flex*** *is defined as*

$$\Delta_{\max,i}(N) = \sum_{j \in [k] \setminus \{i\}} (w \cdot lb_j(N) - c_j(N)). \tag{3.8}$$

During the search of EECBS with Flex Distribution, the core idea is to modify the threshold on the low level to

$$\tau_i(N) = w \cdot \max\{lb_i(\hat{N}), f_{\min,i}(N)\} + \Delta_i(N), \tag{3.9}$$

where $\Delta_i(N)$ is the distributed flex. That is, we can increase the threshold $\tau_i(N)$ given by Equation 3.1 if the maximum allowed flex $\Delta_{\max,i}(N)$ and the distributed flex $\Delta_i(N) \leq \Delta_{\max,i}(N)$ are positive. Also, the original version of EECBS (EECBS without Flex Distribution) is a special case where the distribution flex $\Delta_i(N)$ is set to 0. If EECBS with Flex Distribution needs to find paths for more than one agent in a CT node N (e.g., when resolving a target conflict), then it runs the low-level Focal Searches for the paths sequentially, each time updating the lower bound and the path cost for the agent whose path was updated.

We now prove that the low-level Focal Search with Flex Distribution is guaranteed to find a bounded-suboptimal path, if a path exists.

Lemma 3.1. *For any $\xi \geq 0$, the number of v-t nodes n with $g_i(n) \leq \xi$ is finite.*

Proof. During the search for a path of agent a_i, for any v-t node n with its state representation (v, t), $g_i(n) = t$ is the number of timesteps needed for agent a_i to move from its start vertex s_i to vertex v. According to the problem formulation of MAPF (see Section 2.2), the graph $G = (V, E)$ of a MAPF instance contains a finite number of vertices $|V|$. Also, each action takes exactly 1 timestep, which establishes a strictly positive lower bound on the cost of any state transition. Thus, for any $\xi \geq 0$, the possible values for timestep t are restricted to the finite set $\{0, 1, \ldots, \xi\}$. Since the number of v-t nodes n with $g(n) \leq \xi$ is the product of the number of vertices in the graph G and the number of possible timesteps, it is bounded by $|V| \times (\xi + 1)$. Since $|V|$ and ξ are finite, the number of v-t nodes with $g(n) \leq \xi$ is finite. □

Lemma 3.2. *If every v-t node has a non-negative h_i-value, then, for any arbitrary number $\xi \geq 0$, the number of v-t nodes n with $f_i(n) = g_i(n) + h_i(n) \leq \xi$ is finite.*

Proof. During the search for a path of agent a_i, since $h_i(n) \geq 0$ for every v-t node n, the set of v-t nodes n with $f_i(n) = g_i(n) + h_i(n) \leq \xi$ is a subset of the set of v-t nodes n' with $g_i(n') \leq \xi$. That is, any v-t node with $f_i(n) \leq \xi$ satisfies $g_i(n) \leq \xi$. Since there are finite v-t nodes n whose $g_i(n) \leq \xi$ according to Lemma 3.1, there is a finite number of v-t nodes n with $f_i(n) \leq \xi$. □

Theorem 3.2. *Given a CT node N, if the low-level Focal Search maintains $\tau_i(N) \geq f_{\min,i}(N)$ during the search, then it is guaranteed to find a path for agent a_i that satisfies the constraints $\Psi_i(N)$ with cost $c_i(N) \leq \tau_i(N)$ when the search terminates, if a path that satisfies the constraints exists.*

Proof. Since $w \cdot c_i^*(N) \geq 0$ (where $c_i^*(N)$ is the cost of the minimum-cost path that satisfies the constraints $\Psi_i(N)$), the number of v-t nodes n with $f_i(n) \leq w \cdot c_i^*(N)$ is finite according to Lemma 3.2. During the

search, the low-level Focal Search uses OPEN$_L$ to maintain the frontier of the search tree. At each iteration, the low-level Focal Search expands a v-t node from FOCAL$_L$, which is a subset of OPEN$_L$, and inserts it into CLOSED$_L$. Once a v-t node is expanded, it is not removed from CLOSED$_L$ and re-inserted into OPEN$_L$ unless it is dominated by a newly generated v-t node at an iteration. Since the number of v-t nodes (with $f_i(n) \leq w \cdot c_i^*(N)$) is finite, each v-t node is dominated by finitely many v-t nodes, and thus the number of re-insertions of every expanded v-t node is finite. Thus, each v-t node is expanded in a finite number of times. Since FOCAL$_L$ contains the v-t nodes n in OPEN$_L$ whose $f_i(n) \leq \tau_i(N)$ and $f_{\min,i}(N)$ is the minimum f_i-value over all v-t nodes in OPEN$_L$, $\tau_i(N) \geq f_{\min,i}(N)$ indicates that every v-t node with the minimum f_i-value in OPEN$_L$ is always in FOCAL$_L$ during the search. That is, if OPEN$_L$ is not empty, then FOCAL$_L$ is never empty. Thus, the low-level Focal Search must continue expanding the frontier until either a path that satisfies the constraints is found or OPEN$_L$ is empty. Since the low-level Focal Search expands every v-t node a finite number of times and expands a v-t node at each iteration, it is guaranteed to terminate in a finite number of iterations. Thus, the low-level Focal Search is guaranteed to find a path that satisfies the constraints $\Psi_i(N)$, if a path that satisfies the constraints exists. Also, since FOCAL$_L$ contains v-t nodes with $f_i(n) \leq \tau_i(N)$ and the low-level Focal Search selects a v-t node in FOCAL$_L$ at each iteration, if the selected v-t node results in a path that satisfies the constraints $\Psi_i(N)$, then $c_i(N) \leq \tau_i(N)$ when the search terminates. □

Given a CT node N, we can prove that the low-level Focal Search with Flex Distribution is guaranteed to find a path for agent a_i that satisfies the constraints $\Psi_i(N)$.

Theorem 3.3. *Given a CT node N whose parent CT node is $\hat{N}$, suppose that EECBS maintains lower bound $lb_i(N) = \max\{lb_i(\hat{N}), f_{\min,i}(N)\}$ for each agent a_i. If both of the following conditions hold:*

- *the parent CT node $\hat{N}$ is locally bounded-suboptimal, and*
- *$\Delta_i(N) = \Delta_{\max,i}(N)$ or $\Delta_i(N) \geq 0$,*

then the low-level Focal Search with Flex Distribution is guaranteed to find a path for an agent a_i that satisfies the constraints $\Psi_i(N)$ with the cost satisfying

$$c_i(N) \leq \tau_i(N) = w \cdot \max\{lb_i(\hat{N}), f_{\min,i}(N)\} + \Delta_i(N), \tag{3.10}$$

if a path that satisfies constraints exists.

Proof. If $\Delta_i(N) \geq 0$, then the threshold satisfies

$$\begin{aligned} \tau_i(N) &= w \cdot \max\{lb_i(\hat{N}), f_{\min,i}(N)\} + \Delta_i(N) \\ &\geq w \cdot \max\{lb_i(\hat{N}), f_{\min,i}(N)\} \\ &\geq \max\{lb_i(\hat{N}), f_{\min,i}(N)\} \\ &\geq f_{\min,i}(N) \end{aligned} \tag{3.11}$$

during the search. Thus, according to Theorem 3.2, the low-level Focal Search with Flex Distribution is guaranteed to find a path for agent a_i that satisfies the constraints $\Psi_i(N)$ with cost $c_i(N) \leq \tau_i(N)$ if a path that satisfies constraints exists.

We now prove the case when $\Delta_i(N) = \Delta_{\max,i}(N)$. Since the paths of the other agents $\{a_j \mid j \in [k] \setminus \{i\}\}$ remain unchanged from the parent CT node $\hat{N}$ to CT node N, the distributed flex (which is now the maximum allowed flex) becomes

$$\begin{aligned} \Delta_i(N) = \Delta_{\max,i}(N) &= \sum_{j \in [k] \setminus \{i\}} (w \cdot lb_j(N) - c_j(N)) = \sum_{j \in [k] \setminus \{i\}} (w \cdot lb_j(\hat{N}) - c_j(\hat{N})) \\ &= \sum_{j \in [k]} (w \cdot lb_j(\hat{N}) - c_j(\hat{N})) - (w \cdot lb_i(\hat{N}) - c_i(\hat{N})) \\ &= w \cdot LB(\hat{N}) - C(\hat{N}) - w \cdot lb_i(\hat{N}) + c_i(\hat{N}). \end{aligned} \tag{3.12}$$

Since the parent CT node $\hat{N}$ is locally bounded-suboptimal, $w \cdot LB(\hat{N}) - C(\hat{N}) \geq 0$. Also, since $lb_i(\hat{N})$ is a lower bound on the cost of the minimum-cost path that satisfies the constraints $\Psi_i(\hat{N})$, $c_i(\hat{N}) \geq lb_i(\hat{N})$. Thus, the distributed flex satisfies

$$\Delta_i(N) \geq -w \cdot lb_i(\hat{N}) + c_i(\hat{N}) \geq -w \cdot lb_i(\hat{N}) + lb_i(\hat{N}). \tag{3.13}$$

Accordingly, when the low-level Focal Search finds a path that satisfies the constraints $\Psi_i(N)$ for agent a_i in CT node N, the threshold satisfies

$$\begin{aligned}
\tau_i(N) &= w \cdot \max\{lb_i(\hat{N}), f_{\min,i}(N)\} + \Delta_i(N) \\
&\geq w \cdot \max\{lb_i(\hat{N}), f_{\min,i}(N)\} - w \cdot lb_i(\hat{N}) + lb_i(\hat{N}) \\
&= w \cdot \max\{0, f_{\min,i}(N) - lb_i(\hat{N})\} + lb_i(\hat{N}) \\
&\geq \max\{0, f_{\min,i}(N) - lb_i(\hat{N})\} + lb_i(\hat{N}) \\
&\geq \max\{lb_i(\hat{N}), f_{\min,i}(N)\} \\
&\geq f_{\min,i}(N).
\end{aligned} \tag{3.14}$$

Thus, according to Theorem 3.2, regardless of which $\Delta_{\max,i}(N)$ is positive, 0, or negative, provided that $\Delta_i(N) = \Delta_{\max,i}(N)$, the low-level Focal Search is guaranteed to find a path for agent a_i that satisfies the constraints $\Psi_i(N)$ with cost $c_i(N) \leq \tau_i(N)$ in CT node N, if a path that satisfies the constraints $\Psi_i(N)$ exists. □

We now prove that EECBS with Flex Distribution is guaranteed to find a bounded-suboptimal solution to a MAPF instance if a solution exists. We first prove that the solution found by EECBS with Flex Distribution is guaranteed to be bounded-suboptimal. Then, we prove that EECBS with Flex Distribution is *solution-complete*. That is, it is guaranteed to find a (bounded-suboptimal) solution to a MAPF instance if a solution exists. According to Theorem 3.3, it is trivial to establish the following Theorems 3.4 and 3.5:

Theorem 3.4. *Suppose that the parent CT node $\hat{N}$ of a CT node N is locally bounded-suboptimal. When the low-level Focal Search with Flex Distribution finds a path for an agent a_i that satisfies the constraints $\Psi_i(N)$ in CT node N, if the distributed flex $\Delta_i(N)$ satisfies*

$$\begin{cases} 0 \leq \Delta_i(N) \leq \Delta_{\max,i}(N), & \text{if } \Delta_{\max,i}(N) \geq 0 \\ \Delta_i(N) = \Delta_{\max,i}(N), & \text{otherwise,} \end{cases} \tag{3.15}$$

then CT node N is locally bounded-suboptimal.

Proof. According to Theorem 3.3, if $\Delta_i(N) = \Delta_{\max,i}(N)$ or $\Delta_i(N) \geq 0$, then the low-level Focal Search with Flex Distribution is guaranteed to find a path for an agent a_i that satisfies the constraints $\Psi_i(N)$ with the cost satisfying Inequality 3.10, if a path that satisfies constraints exists. Also, if the distributed flex $\Delta_i(N)$ satisfies Inequality 3.15, then $\Delta_i(N) \leq \Delta_{\max,i}(N)$. Thus, according to Inequality 3.10,

$$\begin{aligned} c_i(N) \leq \tau_i(N) &= w \cdot \max\{lb_i(\hat{N}), f_{\min,i}(N)\} + \Delta_i(N) \\ &\leq w \cdot \max\{lb_i(\hat{N}), f_{\min,i}(N)\} + \Delta_{\max,i}(N) \\ &\leq w \cdot lb_i(N) + \sum_{j \in [k] \setminus \{i\}} (w \cdot lb_j(N) - c_j(N)). \end{aligned} \tag{3.16}$$

where $lb_i(N) = \max\{lb_i(\hat{N}), f_{\min,i}(N)\}$ is a lower bound on an optimal path for agent a_i that satisfies the constraints $\Psi_i(N)$. Thus, the SOC of the CT node N satisfies

$$\begin{aligned} C(N) &= c_i(N) + \sum_{j \in [k] \setminus \{i\}} c_j(N) \\ &\leq w \cdot lb_i(N) + \sum_{j \in [k] \setminus \{i\}} (w \cdot lb_j(N) - c_j(N)) + \sum_{j \in [k] \setminus \{i\}} c_j(N) \\ &= w \cdot LB(N). \end{aligned} \tag{3.17}$$

Thus, according to Definition 3.3, each CT node N is locally bounded-suboptimal. □

Theorem 3.5. *When the low-level Focal Search with Flex Distribution finds a path for an agent a_i that satisfies the constraints $\Psi_i(N)$ in CT node N, if the distributed flex $\Delta_i(N)$ satisfies Inequality 3.15, then the solution found by EECBS with Flex Distribution is guaranteed to be bounded-suboptimal.*

Proof. If the distributed flex satisfies the above condition during the search, then each CT node is locally bounded-suboptimal according to Theorem 3.4. Thus, the solution found by EECBS with Flex Distribution is bounded-suboptimal according to Theorem 3.1. □

Definition 3.9 (Solution-Completeness)**.** *A MAPF algorithm is **solution-completeness** iff it is guaranteed to terminate with a solution to a MAPF instance if a solution exists.*

A solution-complete MAPF algorithm is not guaranteed to terminate and return "no solution" if no solution exists. In fact, if a MAPF instance is unsolvable, EECBS (with or without Flex Distribution) may not terminate. However, there exists a linear-time algorithm that determines if a MAPF instance is solvable [71]. Thus, we can run this algorithm prior to EECBS (with or without Flex Distribution).

Theorem 3.6. *When the low-level Focal Search with Flex Distribution finds a path for an agent a_i that satisfies the constraints $\Psi_i(N)$ in CT node N, if the distributed flex $\Delta_i(N)$ satisfies Inequality 3.15, then EECBS with Flex Distribution is solution-complete.*

Proof. Suppose that the MAPF instance has a solution, and let C^* be the SOC of an optimal solution. Since there are only finitely many conflicts for paths with costs at most $w \cdot C^*$, and once a conflict is resolved at a CT node, it does not reappear in its subtree, the number of CT nodes with SOCs at most $w \cdot C^*$ is finite. Also, since EECBS with Flex Distribution never selects for expansion a CT node whose the SOC exceeds $w \cdot C^*$ according to Selection Rules (E1), (E2), and (E3) (see Inequalities 3.5 and 3.6), the number of CT nodes that EECBS with Flex Distribution expands has to be finite. Also, according to Lemma 2.1, a CT node whose list of paths is a solution is in the subtree rooted at at least one of the CT nodes in CLEANUP$_H$, meaning that EECBS with Flex Distribution stops only when it has selected a CT node whose list of paths is a solution.

Thus, EECBS with Flex Distribution must find a solution with finitely many CT node expansions, if a solution exists. □

Theorem 3.7. *When the low-level Focal Search with Flex Distribution finds a path for an agent a_i that satisfies the constraints $\Psi_i(N)$ in CT node N, if the distributed flex $\Delta_i(N)$ satisfies Inequality 3.15, then EECBS with Flex Distribution is guaranteed to find a bounded-suboptimal solution to a MAPF instance if a solution exists.*

Proof. EECBS with Flex Distribution is solution-complete according to Theorem 3.6, and the solution that it finds is guaranteed to be bounded-suboptimal according to Theorem 3.5. Thus, EECBS with Flex Distribution is guaranteed to find a bounded-suboptimal solution to a MAPF instance if a solution exists. □

Algorithm 3.1 shows the pseudo-code of how Flex Distribution is implemented in the low-level Focal Search, where the blue text marks the differences to the low-level Focal Search without Flex Distribution in Algorithm 2.3. Namely, the low-level Focal Search computes the distributed flex $\Delta_i(N)$ from the paths of the other agents $\{a_j \mid j \in [k] \setminus \{i\}\}$ in CT node N [Line 5] and modifies the threshold accordingly [Lines 10 and 17]. After the low-level Focal Search finds a path, it updates the lower bound [Line 24].

3.5 Bypassing Conflicts with Flex Distribution

When EECBS expands a CT node $\hat{N}$ and generates one of its child CT nodes N, Condition (C2) of bypassing conflicts requires $c_i(N) \leq w \cdot lb_i(\hat{N})$ for each agent a_i (see Section 2.4.3.1). This is to ensure that each path remains individually bounded-suboptimal after bypassing conflicts. On the other hand, Flex Distribution relaxes the requirement that each path be individually bounded-suboptimal, provided that each CT node is locally bounded-suboptimal. Thus, to bypass conflicts when flex distribution is introduced, EECBS with Flex Distribution maintains Conditions (C1), (C3), and (C4) and relaxes Condition (C2) to

$$C(N) \leq w \cdot LB(\hat{N}), \tag{3.18}$$

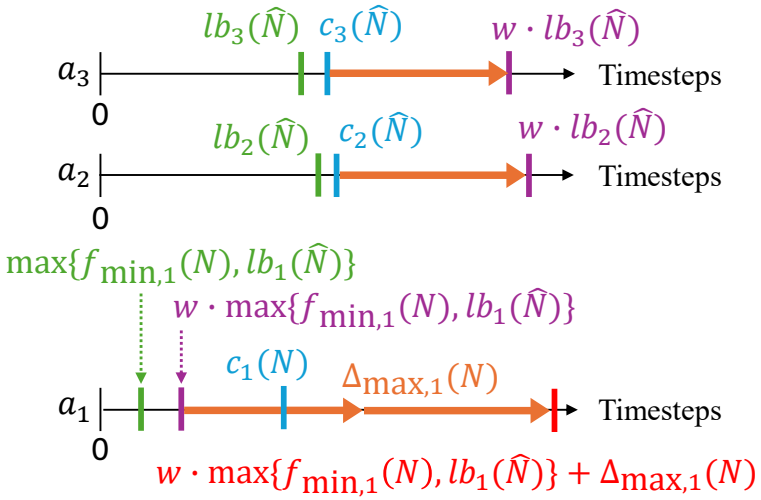


Figure 3.1: An example of how GFD increases the threshold with a positive maximum allowed flex from the flex of the paths of agents a_2 and a_3. Suppose that EECBS-GFD expands CT node $\hat{N}$ and generates one of its child CT nodes N, which requires finding a path for agent a_1. The green bars are the lower bounds, the cyan bars are the costs of the paths, the purple bars are thresholds that are the bounded-suboptimality factor w larger than the lower bounds, and the red bar is the threshold increased by GFD. The orange arrows are the flex of the paths of agents a_2 and a_3.

which results in CT node $\hat{N}$ remaining locally bounded-suboptimal after bypassing conflicts. With the relaxation of Condition (C2), each CT node in EECBS with Flex Distribution remains locally bounded-suboptimal. Thus, according to Theorem 3.1, the solution found by EECBS with Flex Distribution is guaranteed to be bounded-suboptimal when applying bypassing conflicts.

3.6 Greedy Flex Distribution (GFD)

One intuitive way of using flex is to distribute the maximum allowed flex each time when the low-level Focal Search needs to find a path that satisfies the constraints $\Psi_i(N)$ for an agent a_i in a CT node N. That is,

$$\Delta_i(N) = \Delta_{\max,i}(N). \tag{3.19}$$

We call this Flex Distribution mechanism *Greedy Flex Distribution* (GFD). By using GFD, if the maximum allowed flex is positive, then the threshold is increased. This makes the low-level Focal Search easier (and

thus often faster) to find a path that satisfies the constraints because this path can have a larger cost than without Flex Distribution. On the other hand, even if the maximum allowed flex is negative, the low-level Focal Search is still guaranteed to find a path that satisfies constraints (see Theorem 3.3).

Figures 3.1 and 3.2 show examples of the variable relationships in the situation where the low-level Focal Search finds a path for agent a_1 and the paths of agents a_2 and a_3 remain unchanged when EECBS with GFD (EECBS-GFD) expands CT node $\hat{N}$ and generates CT node N. Figure 3.1 shows the case when EECBS-GFD increases the threshold with a positive maximum allowed flex when finding a path for agent a_i. The paths of agents a_2 and a_3 in the parent CT node $\hat{N}$ have costs $c_2(\hat{N}) \leq w \cdot lb_2(\hat{N})$ and $c_3(\hat{N}) \leq w \cdot lb_3(\hat{N})$, which results in the positive flex $w \cdot lb_2(\hat{N}) - c_2(\hat{N})$ (the orange arrow pointing to the right in the line of a_2) and $w \cdot lb_3(\hat{N}) - c_3(\hat{N})$ (the orange arrow pointing to the right in the line of a_3), respectively. Thus, as the paths of agents a_2 and a_3 remain unchanged, the maximum allowed flex for finding a path for agent a_1 (the connected orange arrows pointing to the right in the line of a_1)

$$\Delta_{\max,1}(N) = \sum_{j \in \{2,3\}} (w \cdot lb_j(N) - c_j(N)) = \sum_{j \in \{2,3\}} (w \cdot lb_j(\hat{N}) - c_j(\hat{N})) \tag{3.20}$$

is positive. EECBS-GFD thus increases the threshold from $w \cdot \max\{f_{\min,1}(N), lb_1(\hat{N})\}$ (the purple bar in the line of a_1) to $w \cdot \max\{f_{\min,1}(N), lb_1(\hat{N})\} + \Delta_{\max,1}(N)$ (the red bar in the line of a_1). Then, the low-level Focal Search finds a path with cost $c_1(N)$ (the cyan bar in the line of a_1) and terminates. Finally, EECBS-GFD sets the lower bound $lb_1(N)$ to $\max\{f_{\min,1}(N), lb_1(\hat{N})\}$ (the green bar in the line of a_1).

On the other hand, Figure 3.2 shows the case when EECBS-GFD decreases the threshold with a negative maximum allowed flex when finding a path for agent a_i. The paths of agents a_2 and a_3 in the parent CT node $\hat{N}$ have costs $c_2(\hat{N}) > w \cdot lb_2(\hat{N})$ and $c_3(\hat{N}) > w \cdot lb_3(\hat{N})$, which results in the negative flex $w \cdot lb_2(\hat{N}) - c_2(\hat{N})$ (the orange arrow pointing to the left in the line of a_2) and $w \cdot lb_3(\hat{N}) - c_3(\hat{N})$ (the orange arrow pointing to the left in the line of a_3), respectively. Thus, the maximum allowed flex for

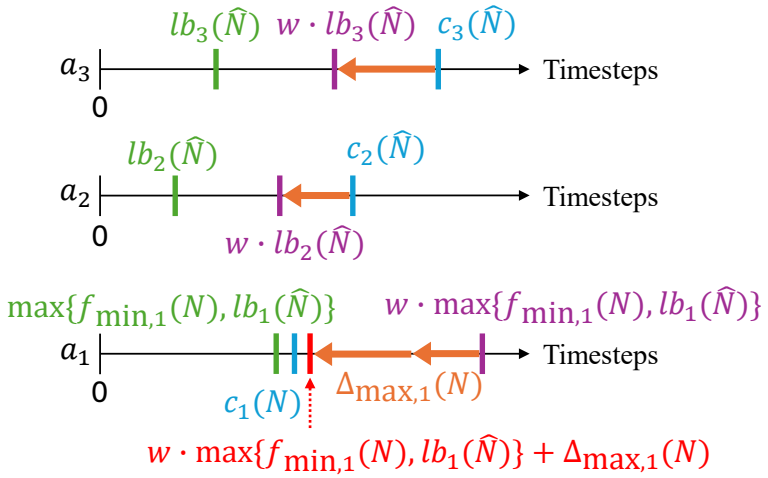


Figure 3.2: An example of how GFD decreases the threshold with a negative maximum allowed flex from the flex of the paths of agents a_2 and a_3. Suppose that EECBS-GFD expands CT node $\hat{N}$ and generates one of its child CT nodes N, which requires finding a path for agent a_1. The green bars are the lower bounds, the cyan bars are the costs of the paths, the purple bars are thresholds that are the bounded-suboptimality factor w larger than the lower bounds, and the red bar is the threshold decreased by GFD. The orange arrows are the flex of the paths of agents a_2 and a_3.

finding a path for agent a_1 (the connected orange arrows pointing to the left in the line of a_1) is negative (i.e., $\Delta_{\max,1}(N) < 0$). EECBS-GFD thus decreases the threshold from $w \cdot \max\{f_{\min,1}(N), lb_1(\hat{N})\}$ (the purple bar in the line of a_1) to $w \cdot \max\{f_{\min,1}(N), lb_1(\hat{N})\} + \Delta_{\max,1}(N)$ (the red bar in the line of a_1). Then, the low-level Focal Search finds a path with cost $c_1(N)$ (the cyan bar in the line of a_1) and terminates. Finally, EECBS-GFD sets the lower bound $lb_1(N)$ to $\max\{f_{\min,1}(N), lb_1(\hat{N})\}$ (the green bar in the line of a_1).

3.7 A Toy Example

We use a MAPF instance to show how GFD benefits EECBS. As shown in Figure 3.3 (a), the MAPF instance contains three agents a_1 (blue), a_2 (green), and a_3 (orange) on a 4×4 four-neighbor grid graph with the respective start vertices $[s_1, s_2, s_3] = [A2, B1, A4]$ and target vertices $[l_1, l_2, l_3] = [D3, C4, A3]$. We use EECBS with the bounded-suboptimality factor $w = 1.2$. For each agent $a_{i \in \{1,2,3\}}$, we use the cost of the

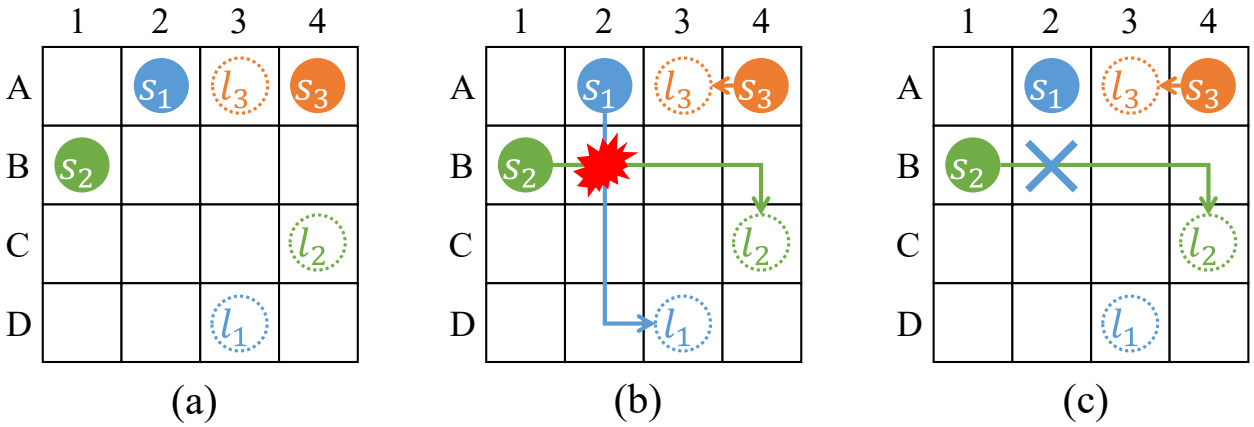


Figure 3.3: (a) An example MAPF instance with 3 agents a_1 (blue), a_2 (green), and a_3 (orange) on a 4×4 four-neighbor grid graph with the start vertices $[s_1, s_2, s_3] = [A2, B1, A4]$ and target vertices $[l_1, l_2, l_3] = [D3, C4, A3]$. (b) The paths of the agents and the selected conflict $\langle a_1, a_2, B2, 1\rangle$ (the red explosion) to resolve in the root CT node N_0 of EECBS. (c) The constraint $\langle a_1, B2, 1\rangle$ (the blue cross) when generating one of the child CT nodes N from the root CT node N_0.

minimum-cost path from each vertex to the target vertex $l_{i\in\{1,2,3\}}$ as the admissible heuristic for the low-level Focal Search. Here, the low-level Focal Search sorts the v-t nodes n in FOCAL$_L$ in increasing order of their number of conflicts $x_i(n)$, breaking ties in favor of v-t nodes with smaller f_i-values. The low-level Focal Search uses random selection for any further unresolved ties.

To find a solution for the MAPF instance, EECBS first generates a root CT node N_0 on the high level by finding paths individually for each agent via the low-level Focal Search, as shown in Figure 3.3 (b). The lower bounds and the costs of the paths are, respectively, $lb_1(N_0) = c_1(N_0) = 4$, $lb_2(N_0) = c_2(N_0) = 4$, and $lb_3(N_0) = c_3(N_0) = 1$, resulting in $LB(N_0) = C(N_0) = 4 + 4 + 1 = 9$. There is $|\mathcal{X}(N_0)| = 1$ conflict, namely $\langle a_1, a_2, B2, 1\rangle$. To resolve this conflict, EECBS generates two child CT nodes from the root CT node, where one of the child CT nodes N contains the constraint $\langle a_1, B2, 1\rangle$ that agent a_1 cannot be at vertex $B2$ at timestep 1, as shown in Figure 3.3 (c). In this case, when EECBS expands the root CT node N_0 and generates one of the child CT nodes N, it runs the low-level Focal Search to find a path for agent a_1 that satisfies constraint $\langle a_1, B2, 1\rangle$.

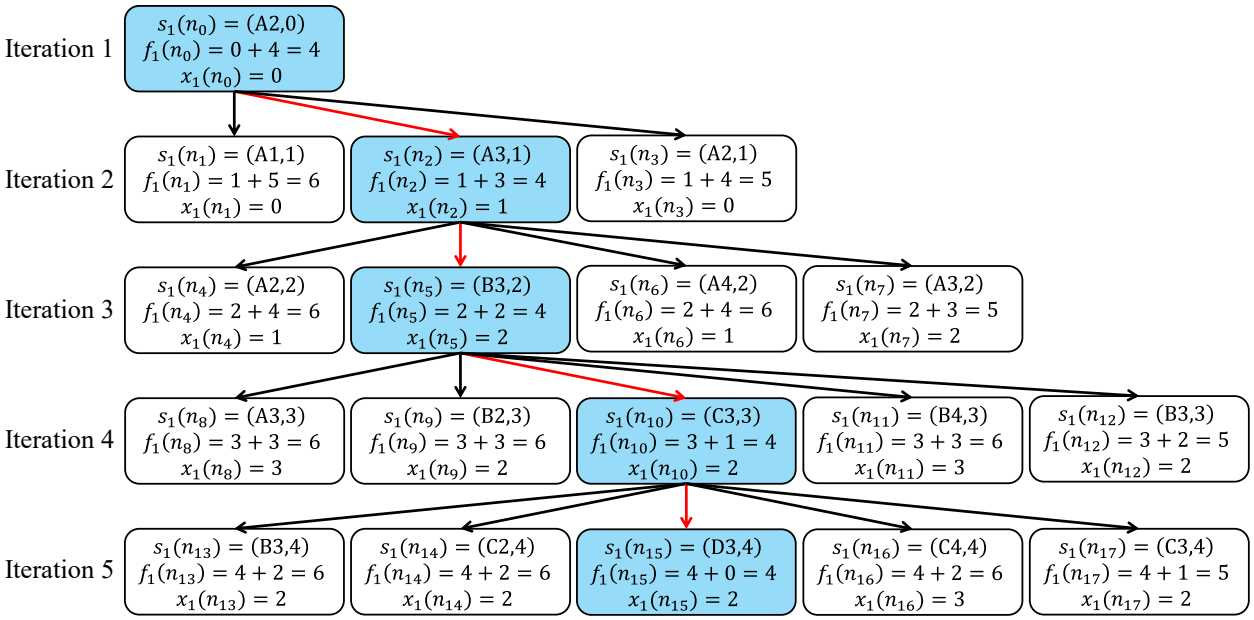


Figure 3.4: Search tree of the low-level Focal Search without Flex Distribution when EECBS finds a path for agent a_1 that satisfies constraint $\langle a_1, B2, 1\rangle$ in CT node N. Each block represents a v-t node n with the state representation $s_1(n) = (v, t)$, the f-value in the format $f_1(n) = g_1(n) + h_1(v)$, and the number of conflicts $x_1(n)$. The white blocks are the v-t nodes that remain in OPEN$_L$ during the search. The blue blocks are the v-t nodes that are expanded at each iteration and then moved to CLOSED$_L$. The red arrows indicate the found path $[A2, A3, B3, C3, D3]$.

3.7.1 Focal Search without Flex Distribution

We first show how the low-level Focal Search (with bounded-suboptimality factor $w = 1.2$) without Flex Distribution finds a path for agent a_1 that satisfies the constraint $\langle a_1, B2, 1\rangle$ when EECBS expands the root CT node N_0 and generates one of its child CT nodes N. Figure 3.4 shows the search tree of v-t nodes during the search. Since the minimum f_1-value among all v-t nodes in OPEN$_L$ remains $f_{\min,1}(N) = 4$ at each iteration (since there always exists a v-t node n with $f_1(n) = 4$ in OPEN$_L$ during the search), the threshold on the cost of the path remains $w \cdot \max\{lb_1(N_0), f_{\min,1}(N)\} = 1.2 \cdot 4 = 4.8$. Since every action takes one timestep to execute, the f_1-values of the v-t nodes are integers. Thus, the low-level Focal Search includes only v-t nodes n with $f_1(n) \leq 4$ in FOCAL$_L$. At Iteration 2, the low-level Focal Search expands v-t node n_2 with state representation $s_1(n_2) = (A3, 1)$, $f_1(n_2) = 4$, and $x_1(n_2) = 1$, which results in the conflict $\langle a_1, a_3, A3, 1\rangle$.

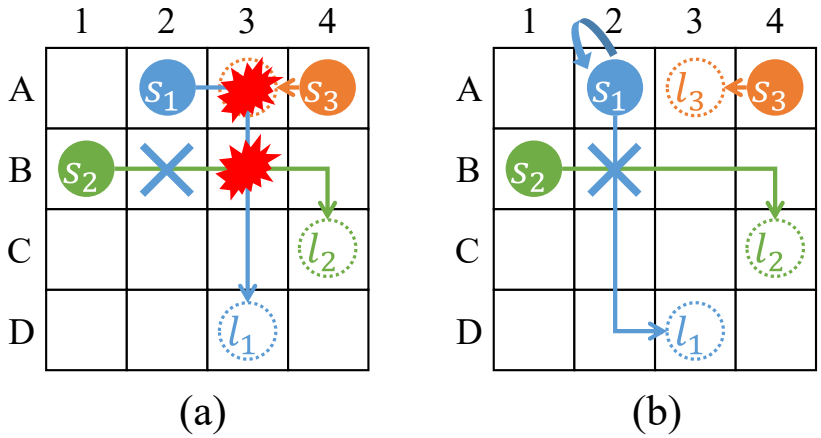


Figure 3.5: Path for agent a_1 that satisfies constraint $\langle a_1, B2, 1\rangle$ in CT node N. (a) shows the path without flex distribution, resulting in 2 conflicts $\langle a_1, a_2, A3, 1\rangle$ and $\langle a_1, a_2, B3, 2\rangle$. (b) shows the path with GFD, resulting in no conflicts by introducing a wait action from timestep 0 to timestep 1 to satisfy constraint $\langle a_1, B2, 1\rangle$.

As shown in Figure 3.4, after expanding v-t nodes n_0, n_2, n_5, n_{10}, and n_{15}, the low-level Focal Search finds the path $p_1(N) = [A2, A3, B3, C3, D3]$ for agent a_1 that satisfies the constraint $\langle a_1, B2, 1\rangle$ with the cost $c_1(N) = 4 \leq w \cdot \max\{lb_1(N), f_{\min,1}(N)\} = 4.8$. However, as shown in Figure 3.5 (a), the generated child CT node N has two conflicts, namely $\langle a_1, a_3, A3, 1\rangle$ and $\langle a_1, a_2, B3, 2\rangle$, resulting in $|\mathcal{X}(N)| = 2$. With the lower bound and the cost of the path of agent a_1 remaining 4, the SOLB and SOC of the generated child CT node N remain $LB(N) = C(N) = 9$. However, the generated child CT node N now has more conflicts than the root CT node. To resolve the selected conflict $\langle a_1, a_2, B2, 1\rangle$, EECBS also generates another CT node N' with constraint $\langle a_2, B2, 1\rangle$. Without Flex Distribution, the low-level Focal Search still finds a path $p_2(N') = [B1, C1, C2, C3, C4]$ for agent a_2 in CT node N' with $lb_2(N') = c_2(N') = 4$. However, given that the path $p_1(N') = p_1(N_0) = [A2, B2, C2, D2, D3]$, the generated child CT node N' now has a conflict $\langle a_1, a_2\rangle$, resulting in $|\mathcal{X}(N')| = 1$. Thus, EECBS still needs further CT node expansions to find a solution.

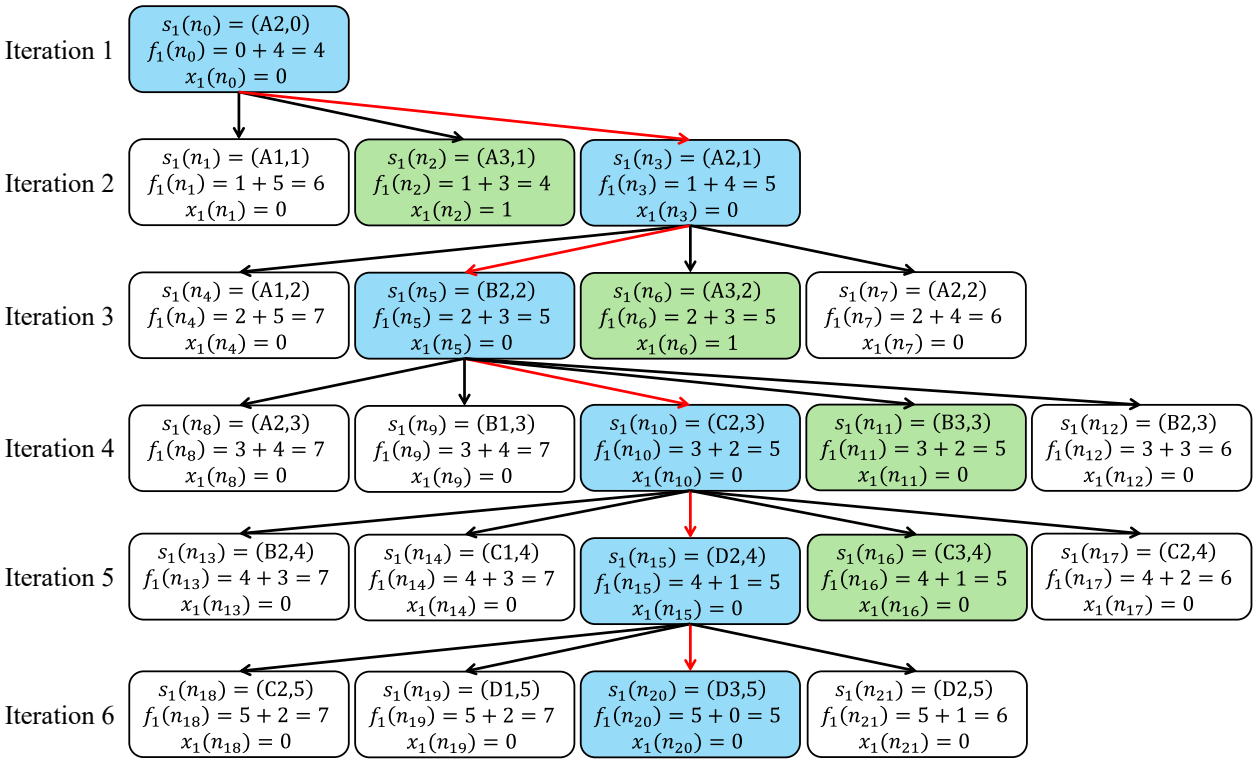


Figure 3.6: Search tree of the low-level Focal Search with GFD when EECBS finds a path for agent a_1 that satisfies constraint $\langle a_1, B2, 1\rangle$ in CT node N. We use the same representation of each block and each arrow as in Figure 3.4. Additionally, the green blocks are the v-t nodes that remain in FOCAL$_L$ during the search. The found path is $[A2, A2, B2, C2, D2, D3]$.

3.7.2 Focal Search with Greedy Flex Distribution

We next show how EECBS with GFD finds a path that satisfies constraint $\langle a_1, B2, 1\rangle$ for agent a_1 in CT node N. Since the paths for agents a_2 and a_3 remain unchanged in CT node N while finding a path for agent a_1, the flex of the paths for agents a_2 and a_3 are, respectively,

$$\begin{aligned} w \cdot lb_2(N) - c_2(N) = w \cdot lb_2(N_0) - c_2(N_0) = 1.2 \cdot 4 - 4 = 0.8 \text{ and} \\ w \cdot lb_3(N) - c_3(N) = w \cdot lb_3(N_0) - c_3(N_0) = 1.2 \cdot 1 - 1 = 0.2. \end{aligned} \tag{3.21}$$

Thus, the maximum allowed flex for finding a path for agent a_1 is

$$\Delta_{\max,1}(N) = 0.8 + 0.2 = 1. \tag{3.22}$$

Figure 3.6 shows the search tree during the low-level Focal Search. Since the minimum f_1-value of all v-t nodes in OPEN$_L$ remains $f_{\min,1}(N) = 4$ at each iteration, with the help of GFD, the threshold becomes

$$\tau_1(N) = w \cdot \max\{lb_1(N_0), f_{\min,1}(N)\} + \Delta_{\max,1}(N) = 1.2 \cdot 4 + 1 = 5.8, \tag{3.23}$$

meaning that v-t nodes n with $f_1(n) = 5$ are also included in FOCAL$_L$ during the search. In this case, at Iteration 2, rather than expanding v-t node n_2 with $f_1(n_2) = 4$ and $x_1(n_2) = 1$ like the low-level Focal Search without Flex Distribution (which results in the conflict $\langle a_1, a_3, A3, 1\rangle$), the low-level Focal Search with GFD expands v-t node n_3 with $f_1(n_3) = 5$ and $x_1(n_3) = 0$.

As shown in Figure 3.6, after expanding v-t nodes n_0, n_3, n_5, n_{10}, n_{15}, and n_{20}, the low-level Focal Search finds the path $p_1(N) = [A2, A2, B2, C2, D2, D3]$ for agent a_1 that satisfies the constraint $\langle a_1, B2, 1\rangle$ with the cost $c_1(N) = 5$ and the lower bound $lb_1(N) = \max\{lb_1(N_0), f_{\min,1}(N)\} = 4$. That is, path $p_1(N)$ contains an extra wait action at vertex $A2$ from timestep 0 to timestep 1 to satisfy constraint $\langle a_1, B2, 1\rangle$. Also, the path of agent a_3 is $p_3(N) = [A4, A3]$, indicating that agent a_3 moves from vertex $A4$ to its target vertex $A3$ from timestep 0 to timestep 1 and waits there permanently. Thus, at Iteration 2, by expanding v-t node n_3 with state representation $s_1(n_3) = (A2, 1)$ rather than v-t node n_2 with state representation $s_1(n_2) = (A3, 1)$, the low-level Focal Search avoids conflict $\langle a_1, a_3, A3, 1\rangle$. Also, at Iteration 3, by expanding v-t node n_5 with state representation $s_1(n_5) = (B2, 2)$ rather than v-t node n_6 with state representation $s_1(n_6) = (A3, 2)$, the low-level Focal Search avoids conflict $\langle a_1, a_3, A3, 2\rangle$.

With the increase of the threshold provided by GFD, the low-level Focal Search finds a path for agent a_1 that satisfies constraint $\langle a_1, B2, 1\rangle$ without any conflicts, resulting in CT node N being conflict-free (i.e., $|\mathcal{X}(N)| = 0$). Although the path $p_1(N)$ is no longer individually bounded-suboptimal as

$$c_1(N) = 5 \not\leq w \cdot lb_1(N) = w \cdot \max\{lb_1(N_0), f_{\min,1}(N)\} = 1.2 \cdot 4 = 4.8, \tag{3.24}$$

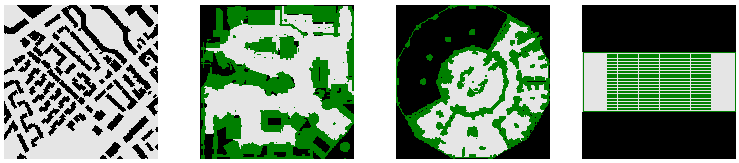
the generated CT node N is still locally bounded-suboptimal since

$$\begin{aligned} C(N) &= \sum_{i \in [1,2,3]} c_i(N) = 5 + 4 + 1 = 10 \\ &\leq w \cdot LB(N) = w \cdot \sum_{i \in [1,2,3]} lb_i(N) = 1.2 \cdot (4 + 4 + 1) = 10.8. \end{aligned} \tag{3.25}$$

To resolve the selected conflict $\langle a_1, a_2, B2, 1 \rangle$, EECBS also generates another CT node N' with constraint $\langle a_2, B2, 1 \rangle$. By using GFD, the low-level Focal Search finds the path $p_2(N') = [B1, C1, C1, C2, C3, C4]$ for agent a_2 in CT node N' with $lb_2(N') = 4$ and $c_2(N') = 5$. The path contains a wait action at vertex $C1$ at timestep 1, resulting in the generated CT node N' also being conflict-free (i.e., $|\mathcal{X}(N')| = 0$) and locally bounded-suboptimal since $C(N') = 10 \leq w \cdot LB(N') = 1.2 \cdot (4 + 4 + 1) = 10.8$. Thus, both CT nodes N and N' contain solutions to the MAPF instance. With CT nodes N and N' being the only two CT nodes in CLEANUP$_H$, the lower bound LB on the SOC of an optimal solution is 9, and CT nodes N and N' are globally bounded-suboptimal and included in FOCAL$_H$. Thus, EECBS-GFD can find a bounded-suboptimal solution in the next iteration without further expanding CT nodes, thereby accelerating the search compared to EECBS without GFD.

3.8 Empirical Evaluation

In this section, we evaluate EECBS (without Flex Distribution) and EECBS with GFD (EECBS-GFD). We implement enhancements for both MAPF algorithms, including bypassing conflicts, prioritizing conflicts, and symmetry breaking (see Section 2.4.3). We implement all MAPF algorithms in `C++` (compiled with `GCC-11.3.0`) and run the experiments with `CentOS Linux` on an `Intel Xeon-2640 v4` processor with 16 GB of memory. We first describe the configuration used for the experiments. Then, we compare their performance of EECBS and EECBS-GFD. Also, we evaluate the effectiveness of GFD for accelerating EECBS.



Name	city	den520d	ost003d	warehouse
w x h	256 x 256	256 x 257	194 x 194	161 x 63
|V|	47,747	28,178	13,214	5,699

Figure 3.7: The widths w, heights h, and numbers of vertices $|V|$ of the four-neighbor grid graphs `city`, `den520d`, `ost003d`, and `warehouse`.

	1-degree	2-degree	3-degree	4-degree	ratio
`city`	209	3,469	2,125	41,944	0.88
`den520d`	48	696	2,220	25,214	0.89
`ost003d`	73	711	1,217	11,213	0.85
`warehouse`	0	2,464	312	2,923	0.51

Table 3.1: The number of vertices of different degrees in each four-neighbor grid graph. The column "ratio" is the ratio of the number of vertices of degree 4 and the total number of vertices in each graph.

3.8.1 Configuration Setting

We use four four-neighbor grid graphs from the MAPF benchmark suite [59]: `city`, `den520d`, `ost003d`, and `warehouse`. Figure 3.7 shows the layout, width, height, and number of vertices of each graph. The width of the corridors in `warehouse` graph is 1, meaning that only one agent can traverse a corridor at a time[†]. Table 3.1 shows the number of vertices of degrees from 1 to 4 and the ratio of the number of vertices of degree of 4 and the total number of vertices in each graph.

For each graph, there are 25 `random`-scene files in the MAPF benchmark suite, each of which contains a list of non-repeated start vertices and a list of non-repeated target vertices. We use the bounded-suboptimality factors $w = \{1.01, 1.02, 1.05, 1.10\}$. For each graph and bounded-suboptimality factor, we select 5 numbers of agents. For each number of agents k, we select the top k start vertices and the top k target vertices from each `random`-scene file. Table 3.2 shows the number of agents k used for each bounded-suboptimality factor w on each graph. Thus, the number of MAPF instances evaluated for each

[†]The file names of the `city` and `warehouse` graphs are `Boston_0_256` and `warehouse-10-20-10-2-1` in the MAPF benchmark suite. In the `city` graph, there are 21 vertices of degree 0 (i.e., unconnected vertices). We consider these vertices to be obstacles and do not count them as vertices.

Graph	w	Number of agents k	Graph	w	Number of agents k
city	1.01	{600, 700, 800, 900, 1,000}	ost003d	1.01	{100, 200, 300, 400, 500}
	1.02	{1,100, 1,200, 1,300, 1,400, 1,500}		1.02	{100, 200, 300, 400, 500}
	1.05	{1,200, 1,400, 1,600, 1,800, 2,000}		1.05	{400, 500, 600, 700, 800}
	1.10	{1,200, 1,400, 1,600, 1,800, 2,000}		1.10	{400, 500, 600, 700, 800}
den520d	1.01	{600, 700, 800, 900, 1,000}	warehouse	1.01	{100, 150, 200, 250, 300}
	1.02	{1,000, 1,100, 1,200, 1,300, 1,400}		1.02	{100, 150, 200, 250, 300}
	1.05	{1,200, 1,400, 1,600, 1,800, 2,000}		1.05	{100, 150, 200, 250, 300}
	1.10	{1,200, 1,400, 1,600, 1,800, 2,000}		1.10	{200, 250, 300, 350, 400}

Table 3.2: Number of agents k used for creating MAPF instances for each graph and bounded-suboptimality factor w.

	$w = 1.01$		$w = 1.02$		$w = 1.05$		$w = 1.10$	
	EECBS	GFD	EECBS	GFD	EECBS	GFD	EECBS	GFD
city	0.080	0.52	0.088	0.608	0.448	0.840	0.872	0.912
den520d	0.016	0.400	0.048	0.536	0.512	0.672	0.752	0.632
ost003d	0.296	0.376	0.568	0.624	0.496	0.360	0.656	0.448
warehouse	0.328	0.288	0.408	0.448	0.640	0.792	0.840	0.776
Total	0.180	0.396	0.278	0.554	0.524	0.666	0.780	0.692

Table 3.3: Success rates of EECBS and EECBS-GFD for the 120-second runtime limit over all MAPF instances. For each bounded-suboptimality factor w, the columns "EECBS" contain the success rates of EECBS, the columns "GFD" contain the success rates of EECBS-GFD, and the row "Total" contains the success rates over all MAPF instances.

bounded-suboptimal factor w on each graph is $5 \times 25 = 125$, and the number of MAPF instances evaluated for a given bounded-suboptimality factor w is $125 \times 4 = 500$ (namely, 125 MAPF instances per graph $\times$ 4 graphs). We set the runtime limit to 120 seconds for a MAPF algorithm to find a solution for a MAPF instance. If a MAPF algorithm fails to find a solution for a MAPF instance within the runtime limit, we set its runtime to the runtime limit (i.e., 120 seconds).

3.8.2 Performance Comparison

Figure 3.8 shows the success rates of EECBS and EECBS-GFD for the 120-second runtime limit, each bounded-suboptimality factor w, and each number of agents over all MAPF instances on each graph. Figure 3.9 and Table 3.3 show the success rates of EECBS and EECBS-GFD for the 120-second runtime limit and each bounded-suboptimality factor w over all MAPF instances on each graph. For the 120-second runtime

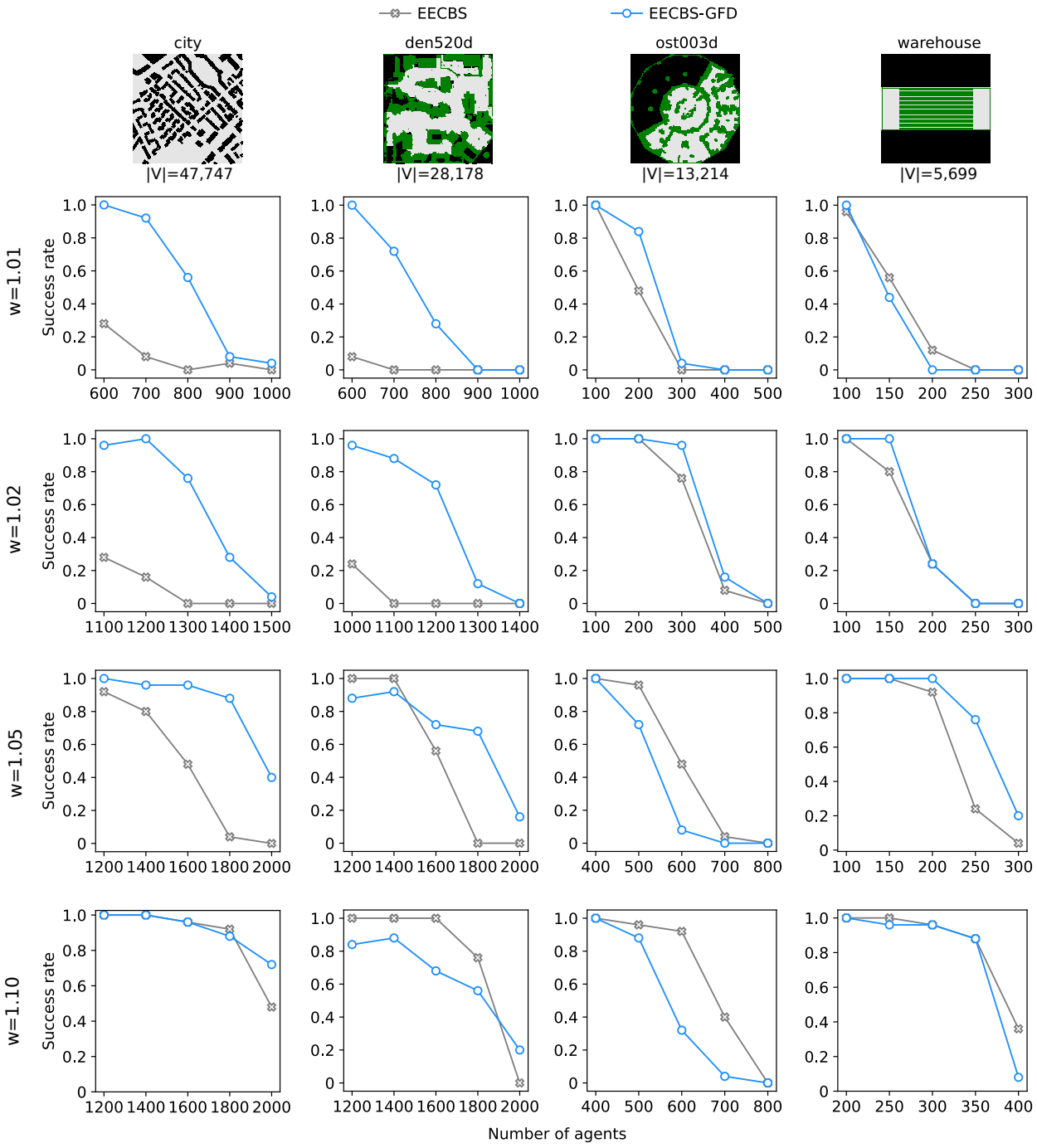


Figure 3.8: Success rates of EECBS and EECBS-GFD for the 120-second runtime limit, each bounded-suboptimality factor w, and each number of agents over all MAPF instances on each graph. $|V|$ indicates the number of vertices of each graph.

limit and the bounded-suboptimality factor $w = 1.01$, using GFD results in a more than twofold increase in the success rate of EECBS over all MAPF instances on all graphs, raising it from 0.18 to 0.396. For

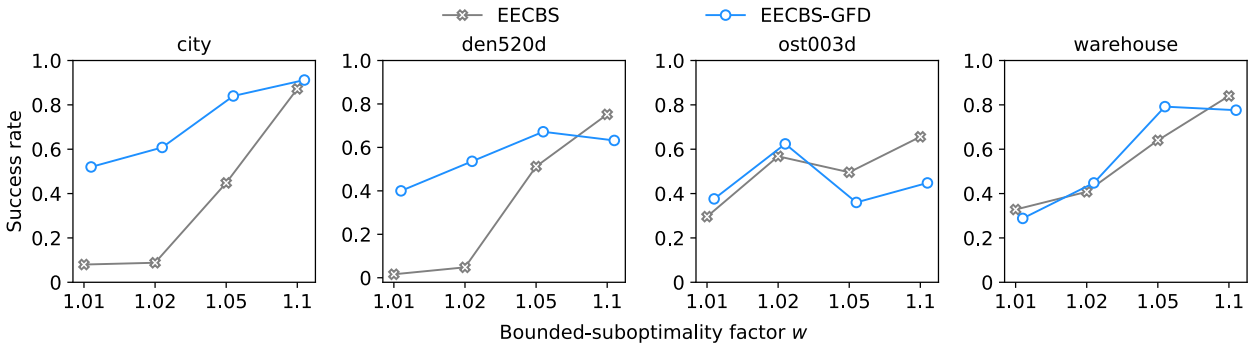


Figure 3.9: Success rates of EECBS and EECBS-GFD for the 120-second runtime limit and each bounded-suboptimality factor w over all MAPF instances on each graph. The values are shown in Table 3.3.

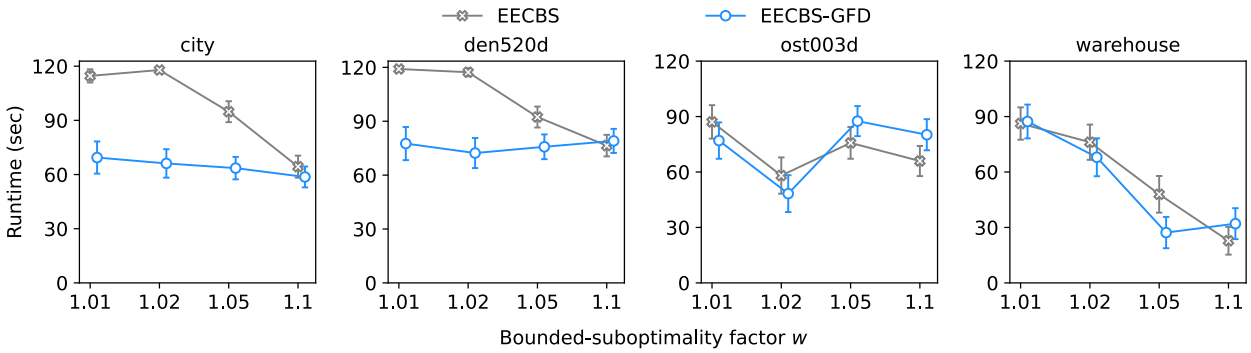


Figure 3.10: Average runtimes (in seconds) of EECBS and EECBS-GFD for each bounded-suboptimality factor w over all MAPF instances on each graph. The vertical bars indicate the 95% confidence intervals.

MAPF instances on large graphs, such as the `city` and `den520d` graphs, EECBS-GFD has higher success rates than EECBS for small bounded-suboptimality factors $w = \{1.01, 1.02\}$. In particular, while EECBS has the success rate of 0.08 over all MAPF instances on the `city` graph (the largest graph in our empirical evaluation) for the bounded-suboptimality factor $w = 1.01$, EECBS with GFD has the success rate of 0.52. However, for MAPF instances on the smaller `ost003d` and `warehouse` graphs, EECBS-GFD can have lower success rates than EECBS for large bounded-suboptimality factors $w = \{1.05, 1.10\}$. This indicates that EECBS-GFD may have some limitations, which will be discussed in Section 4.1. Figure 3.10 shows the average runtimes of EECBS and EECBS-GFD. Typically, the higher the success rates, the lower the average runtimes. Figure 3.11 shows the runtimes of EECBS versus EECBS-GFD for each MAPF instance, where the runtime of EECBS can be 100 times larger than that of EECBS-GFD. As shown in Table 3.4, the

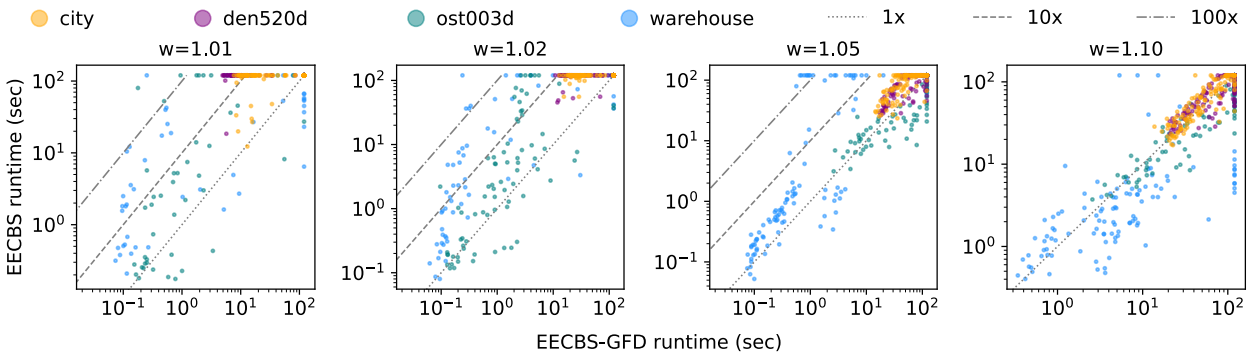


Figure 3.11: Runtimes (in seconds) of EECBS versus EECBS-GFD for each bounded-suboptimality factor w over all MAPF instances. The x- and y-coordinates of each dot represent the runtimes of EECBS-GFD and EECBS, respectively. All the x- and y-coordinates are on logarithmic scales. Dots in yellow, purple, green, and blue represent the MAPF instances on the `city`, `den520d`, `ost003d`, and `warehouse` graphs, respectively.

	$w = 1.01$		$w = 1.02$		$w = 1.05$		$w = 1.10$	
	EECBS	GFD	EECBS	GFD	EECBS	GFD	EECBS	GFD
`city`	0.01	0.51	0.00	0.61	0.05	0.81	0.35	0.61
`den520d`	0.00	0.40	0.01	0.53	0.16	0.59	0.31	0.50
`ost003d`	0.07	0.31	0.17	0.47	0.36	0.14	0.51	0.14
`warehouse`	0.10	0.26	0.04	0.43	0.15	0.66	0.58	0.28
Total	0.04	0.37	0.05	0.51	0.18	0.55	0.44	0.38

Table 3.4: MAPF instance comparisons in terms of runtimes of EECBS and EECBS-GFD over all MAPF instances. For each bounded-suboptimality factor w, the columns "EECBS" contain the percentages of MAPF instances where EECBS has lower runtimes than EECBS-GFD, the columns "GFD" contain the percentages of MAPF instances where EECBS-GFD has lower runtimes than EECBS, and the row "Total" contains the percentage over all MAPF instances.

percentages where EECBS-GFD has a lower runtime than EECBS are typically higher than those of the other way around.

To show the effectiveness of GFD, we evaluate the efficiency of resolving conflicts of EECBS and EECBS-GFD. Suppose that EECBS (with or without Flex Distribution) expands *EXP* CT nodes during a search that successfully finds a solution within the 120-second runtime limit. We define the *conflict resolution efficiency* as

$$\text{conflict resolution efficiency} = \frac{|\mathcal{X}(N_0)|}{EXP}, \tag{3.26}$$

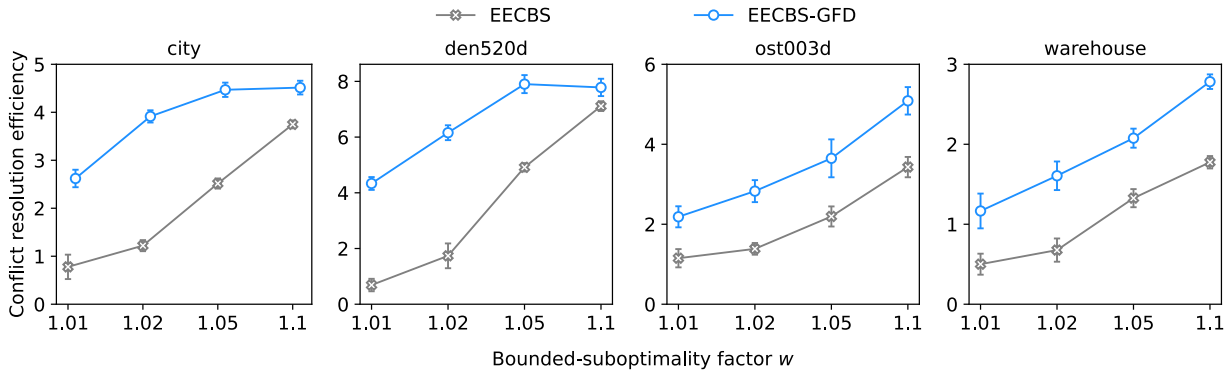


Figure 3.12: Average conflict resolution efficiencies of EECBS and EECBS-GFD for each bounded-suboptimality factor w over their respective MAPF instances that they successfully solve within the 120-second runtime limit on each graph. The vertical bars indicate the 95% confidence intervals.

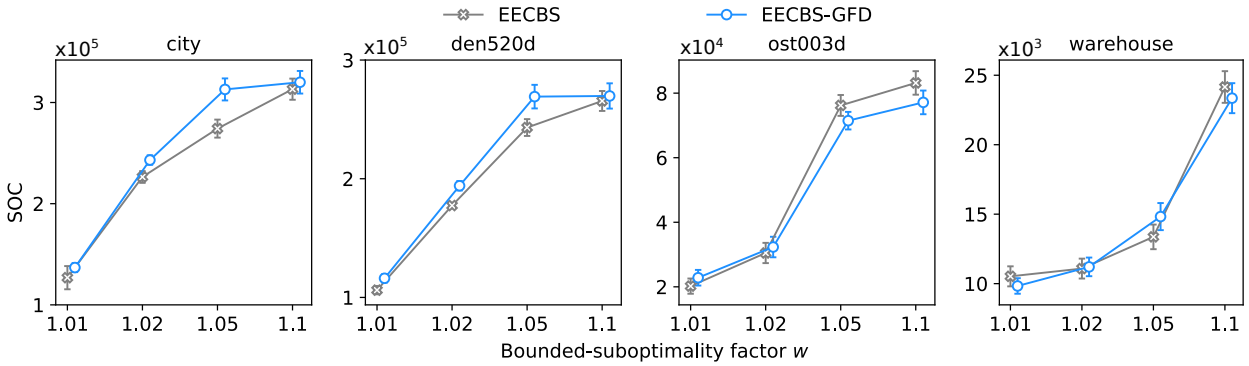


Figure 3.13: Average SOCs of EECBS and EECBS-GFD for each bounded-suboptimality factor w over their respective MAPF instances that they successfully solve within the 120-second runtime limit on each graph. The vertical bars indicate the 95% confidence intervals.

where N_0 is the root CT node of the CT. Thus, the conflict resolution efficiency is the average number of conflicts that EECBS (with or without Flex Distribution) has resolved per CT node expansion. As shown in Figure 3.12, EECBS-GFD has higher conflict resolution efficiencies than EECBS, indicating the effectiveness of GFD in resolving conflicts.

To show the solution qualities resulting from GFD, we compare the SOCs of the solutions found by EECBS and EECBS-GFD. As shown in Figure 3.13, the solutions found by EECBS-GFD have larger average SOCs than those found by EECBS. Together with Figure 3.12, it shows that EECBS-GFD often trades off solution quality for a better conflict resolution efficiency than EECBS. Together with Figures 3.9 and 3.10,

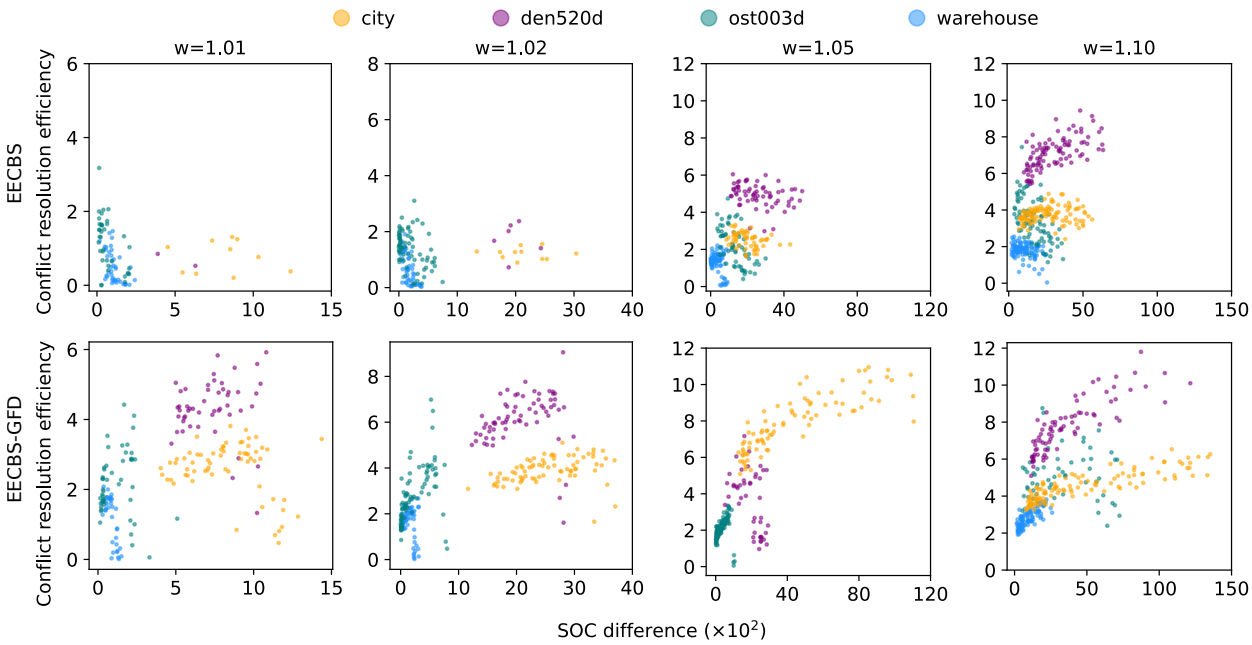


Figure 3.14: Conflict resolution efficiencies versus SOC difference of EECBS and EECBS-GFD with each bounded-suboptimality factor w over their respective MAPF instances that they successfully solve within the 120-second limit on each graph.

it shows that EECBS-GFD often trades off solution quality for a better runtime than EECBS. This trade-off is acceptable since the solutions found by EECBS-GFD remain bounded-suboptimal.

Given a MAPF instance that is solved by EECBS (with or without Flex Distribution) within the 120-second runtime limit, we define the *SOC difference* as

$$\text{SOC difference} = C(\ddot{N}) - C(N_0), \tag{3.27}$$

where N_0 is the root CT node and $\ddot{N}$ is the selected CT node whose list of paths is a solution. As shown in Figure 3.14, EECBS-GFD tends to find solutions with higher conflict resolution efficiency and larger SOC differences than EECBS, indicating that EECBS-GFD trades off solution quality for a better conflict resolution efficiency than EECBS.

Given a MAPF instance with the SOC C^* of an optimal solution, suppose that EECBS (with or without Flex Distribution) finds a solution with SOC C within the 120-second runtime limit. We define the *actual*

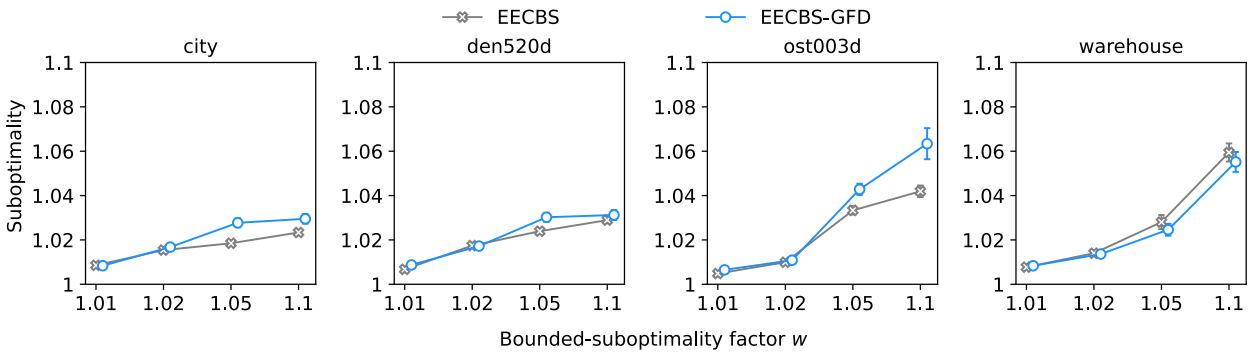


Figure 3.15: Average empirical suboptimalities of EECBS and EECBS-GFD for each bounded-suboptimality factor w over their respective MAPF instances that they successfully solve within the 120-second runtime limit on each graph. The vertical bars indicate the 95% confidence intervals.

suboptimality as C/C^*. However, since finding an optimal solution for MAPF is NP-hard, its SOC C^* is often unknown. Thus, we define the *empirical suboptimality* as

$$\text{Suboptimality} = \frac{C}{LB}, \tag{3.28}$$

where LB is a lower bound on the cost of an optimal solution that EECBS (with or without Flex Distribution) maintains, i.e., $0 \leq LB \leq C^*$. Since EECBS (with or without Flex Distribution) is a bounded-suboptimal MAPF algorithm, both the actual and empirical suboptimalities must be at most w. Also, since $0 \leq LB \leq C^*$, the actual suboptimality satisfies

$$1 \leq \frac{C}{C^*} \leq \frac{C}{LB}. \tag{3.29}$$

Thus, if the empirical suboptimality is close to 1, then the solution found by EECBS (with or without Flex Distribution) has a SOC close to that of an optimal solution.

As shown in Figure 3.15, the average empirical suboptimalities of EECBS and EECBS-GFD are much smaller than the bounded-suboptimality factors. As shown in Figure 3.15, Figure 3.16, and Table 3.5, EECBS tends to find solutions with lower empirical suboptimalities than EECBS-GFD. Since EECBS-GFD uses GFD

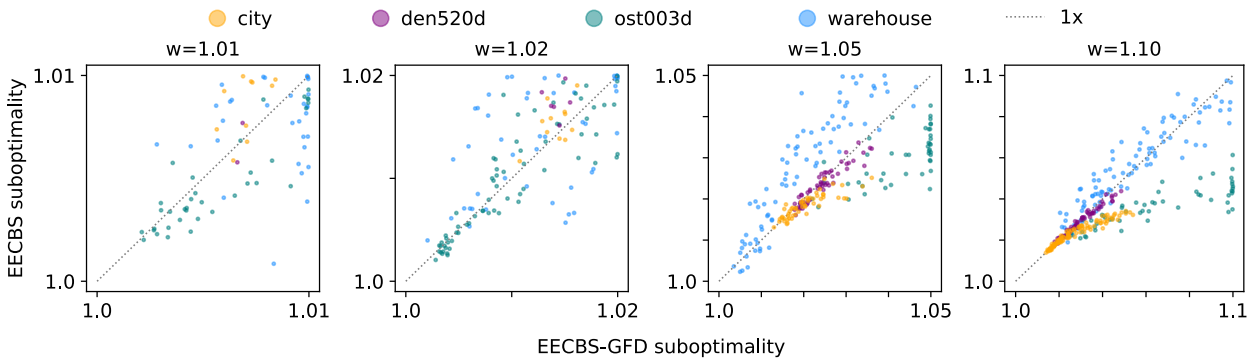


Figure 3.16: Empirical suboptimalities of EECBS versus EECBS-GFD for each bounded-suboptimality factor w over their respective MAPF instances that they successfully solve within the 120-second runtime limit. The x- and y-coordinates of each dot represent the empirical suboptimalities of EECBS-GFD and EECBS, respectively. Dots in yellow, purple, green, and blue represent the MAPF instances on the `city`, `den520d`, `ost003d`, and `warehouse` graphs, respectively.

	$w = 1.01$		$w = 1.02$		$w = 1.05$		$w = 1.10$	
	EECBS	GFD	EECBS	GFD	EECBS	GFD	EECBS	GFD
`city`	2	8	3	8	44	10	90	13
`den520d`	1	1	0	6	35	19	50	21
`ost003d`	29	7	39	30	44	1	55	0
`warehouse`	20	12	20	28	10	68	37	57
Total	52	28	62	72	133	98	232	91

Table 3.5: MAPF instance comparisons of the empirical suboptimalities of EECBS and EECBS-GFD on the same MAPF instances in Figure 3.16. For each bounded-suboptimality factor w, the columns "EECBS" contain the number of MAPF instances where EECBS has lower empirical suboptimalities than EECBS-GFD, the columns "GFD" contain the number of MAPF instances where EECBS-GFD has lower empirical suboptimalities than EECBS, and the row "Total" contains the sum of the number of MAPF instances over each graph.

to increase thresholds to improve conflict-resolution efficiency relative to EECBS, it results in a worse suboptimality.

3.9 Summary

We proposed EECBS with Flex Distribution. Rather than requiring each path to be individually bounded-suboptimal, we can increase the threshold of the low-level Focal Search with the help of Flex Distribution to accelerate EECBS. We also showed how Condition (C2) of bypassing conflicts can be changed to use it with EECBS with Flex Distribution. We proved that EECBS with Flex Distribution is bounded-suboptimal

and solution-complete. That is, it is guaranteed to find a bounded-suboptimal solution for a MAPF instance if a solution exists. Additionally, we introduced a Flex Distribution mechanism called Greedy Flex Distribution (GFD), which increases the threshold of the low-level Focal Search by the maximum allowed flex. Our empirical evaluation shows that GFD can improve the efficiency of EECBS, especially on large-scale MAPF instances. EECBS with GFD has a higher success rate for the 120-second runtime limit than the original EECBS for solving MAPF instances on large maps with different numbers of agents, especially for a small bounded suboptimality factor $w = 1.01$. For the 120-second runtime limit and the bounded-suboptimality factor $w = 1.01$, using GFD raises the success rate of EECBS over all MAPF instances from 0.18 to 0.396. Compared to EECBS, we showed that EECBS with GFD trades off solution quality for better runtime and conflict-resolution efficiency, which is acceptable since the solutions found by EECBS with GFD are guaranteed to be bounded-suboptimal.

Algorithm 3.1 Low-level Focal Search with Flex Distribution

procedure FocalSearchWithFlex(s_i, l_i, $\Psi_i(N)$, $\dot{P}_i(N)$, $lb_i(\hat{N})$)
  ▷ *$\hat{N}$ is the parent CT node of CT node N, so $lb_i(\hat{N}) = 0$ if N is the root CT node.* ◁
  ▷ $\dot{P}_i(N) = \{p_j(N) \mid j \in [k] \setminus \{i\}\}$. ◁
  $T_i(N) \leftarrow$ Compute the time horizon from constraints $\Psi_i(N)$ and paths $\dot{P}_i(N)$ ▷ *See Section 2.4.2.*
  $\Delta_i(N) \leftarrow$ Compute the distributed flex from paths $\dot{P}_i(N)$
  Determine the priority function $\tilde{f}_i(n)$ of v-t node n ▷ *$\tilde{f}_i(n) = f_i(n)$ unless mentioned otherwise.*
  Generate the root v-t node n_0 with state representation $(s_i, 0)$
  $g_i(n_0) \leftarrow 0$; $h_i(n_0) \leftarrow h_i(s_i)$; $f_i(n_0) \leftarrow g_i(n_0) + h_i(n_0)$; $\tilde{h}_i(n_0) \leftarrow \tilde{h}_i(s_i)$; $\tilde{f}_i(n_0) \leftarrow g_i(n_0) + \tilde{h}_i(n_0)$; $x_i(n_0) \leftarrow 0$
  $f_{\min,i}(N) \leftarrow f_i(n_0)$
  $\tau_i(N) \leftarrow w \cdot \max\{lb_i(\hat{N}), f_{\min,i}(N)\} + \Delta_i(N)$
  OPEN_L,FOCAL_L, $\text{CLOSED}_L \leftarrow$ empty list
  InsertNode(n_0, $\tau_i(N)$, OPEN_L, FOCAL_L)
  while OPEN_L **not** empty **do**
    $n_f \leftarrow$ v-t node with the minimum f_i-value in OPEN_L
    if $f_{\min,i}(N) < f_i(n_f)$ **then**
      $f_{\min,i}(N) \leftarrow f_i(n_f)$
      $\tau_{new} \leftarrow w \cdot \max\{lb_i(\hat{N}), f_{\min,i}(N)\} + \Delta_i(N)$
      UpdateFocal($\tau_i(N)$, τ_{new}, OPEN_L, FOCAL_L)
      $\tau_i(N) \leftarrow \tau_{new}$
    $\hat{n} \leftarrow$ v-t node with the minimum x_i-value in FOCAL_L
    Remove v-t node $\hat{n}$ from OPEN_L and FOCAL_L
    Insert $\hat{n}$ into CLOSED_L
    if IsTargetReached($\hat{n}, l_i, \Psi_i(N)$) **then**
      $lb_i \leftarrow \max\{lb_i(\hat{N}), f_{\min,i}(N)\}$
      $p_i \leftarrow$ Extract the path by back-tracking v-t node $\hat{n}$
      $c_i \leftarrow$ cost of path p_i
      return (p_i, c_i, lb_i)
    neighbors $\leftarrow$ FindNeighbors($\hat{n}$) ▷ *Find the neighboring states.*
    for $(v, t) \in$ *neighbors* **do**
      if (v, t) **not** satisfies constraints $\Psi_i(N)$ **then**
        continue
      $n \leftarrow$ GenerateChildNode($\hat{n}$, v, t, $\dot{P}_i(N)$)
      $\bar{n} \leftarrow$ FindNode(n, $T_i(N)$, CLOSED_L, OPEN_L)
      if $\bar{n}$ is null **then**
        InsertNode(n, $\tau_i(N)$, OPEN_L, FOCAL_L)
        continue
      if IsDominant(n, $\bar{n}$) **then**
        UpdatePriority(n, $\bar{n}$, $\tau_i(N)$, CLOSED_L, OPEN_L, FOCAL_L)
  return No path

Chapter 4

Mechanisms for Fractional Flex Distribution

In this chapter, we first discuss the limitations of Greedy Flex Distribution (GFD). When finding a path for an agent a_i in a CT node N with a positive maximum allowed flex, due to the increased threshold, EECBS-GFD can result in fewer conflicts but a larger SOC $C(N)$ than EECBS. Also, the low-level Focal Search with a positive distributed flex from GFD can include and expand v-t nodes with high f-values in FOCAL$_L$ without increasing the lower bound $lb_i(N)$. In this case, the SOLB $LB(N)$ of EECBS-GFD can be smaller than that of EECBS, which can result in a smaller increase of LB during the search of EECBS-GFD. Both of these effects can result in $C(N) > w \cdot LB$, i.e., the generated CT node N not being globally bounded-suboptimal, and thus it may not be expanded by EECBS-GFD. Instead of resolving conflicts in the CT node N with few conflicts, which may lead to a solution, EECBS-GFD can expand the other CT nodes with a smaller SOC but more conflicts.

To alleviate this issue, we distribute only a fraction of the maximum allowed flex. GFD is then a special case in which this fraction equals 1.0. One intuitive way to determine the amount of flex is to treat the fraction as a random variable, leading to our proposed Random Flex Distribution (RFD). Our empirical evaluation shows that the success rates of EECBS with RFD can be higher than those of EECBS for the 120-second runtime limit.

To further improve the efficiency of EECBS, we observe that the subtleties in finding a path for an agent during the low-level Focal Search lie in minimizing the number of conflicts with the paths of the other agents while satisfying the constraints in a CT node. Thus, we propose two more mechanisms: the Conflict-Based Flex Distribution (CFD) and the Delay-Based Flex Distribution (DFD). CFD uses the number of conflicts among the paths in a CT node to determine the amount of flex. On top of CFD, DFD uses the number of additional timesteps needed to find a path that satisfies the constraints in a CT node. Also, we propose the Mixed-Strategy Flex Distribution (MFD), which combines GFD, CFD, and DFD in a hierarchical manner. Our contributions are:

- We discuss the limitations of using GFD.
- We propose several mechanisms to find a path for an agent. We show that EECBS with each of our Flex Distribution mechanisms is still guaranteed to find a bounded-suboptimal solution to a MAPF instance if a solution exists.
- We evaluate our design choices and our Flex Distribution mechanisms empirically in multiple scenarios, showing that EECBS-MFD has higher success rates than EECBS and EECBS-GFD for the 120-second runtime limit. Our empirical result shows that for the 120-second runtime limit and the bounded-suboptimality factor $w = 1.01$, using MFD results in a more than threefold increase in the success rate of EECBS over all MAPF instances on all graphs, raising it from 0.18 to 0.598. In particular, while EECBS and EECBS with GFD have the success rates of 0.08 and 0.52, respectively, over all MAPF instances on the `city` graph (which is the largest graph in our empirical evaluation) for the bounded-suboptimality $w = 1.01$, EECBS with MFD has the success rate of 0.816.

4.1 Limitations of Greedy Flex Distribution (GFD)

We identified limitations of GFD on both the high and low levels of EECBS.

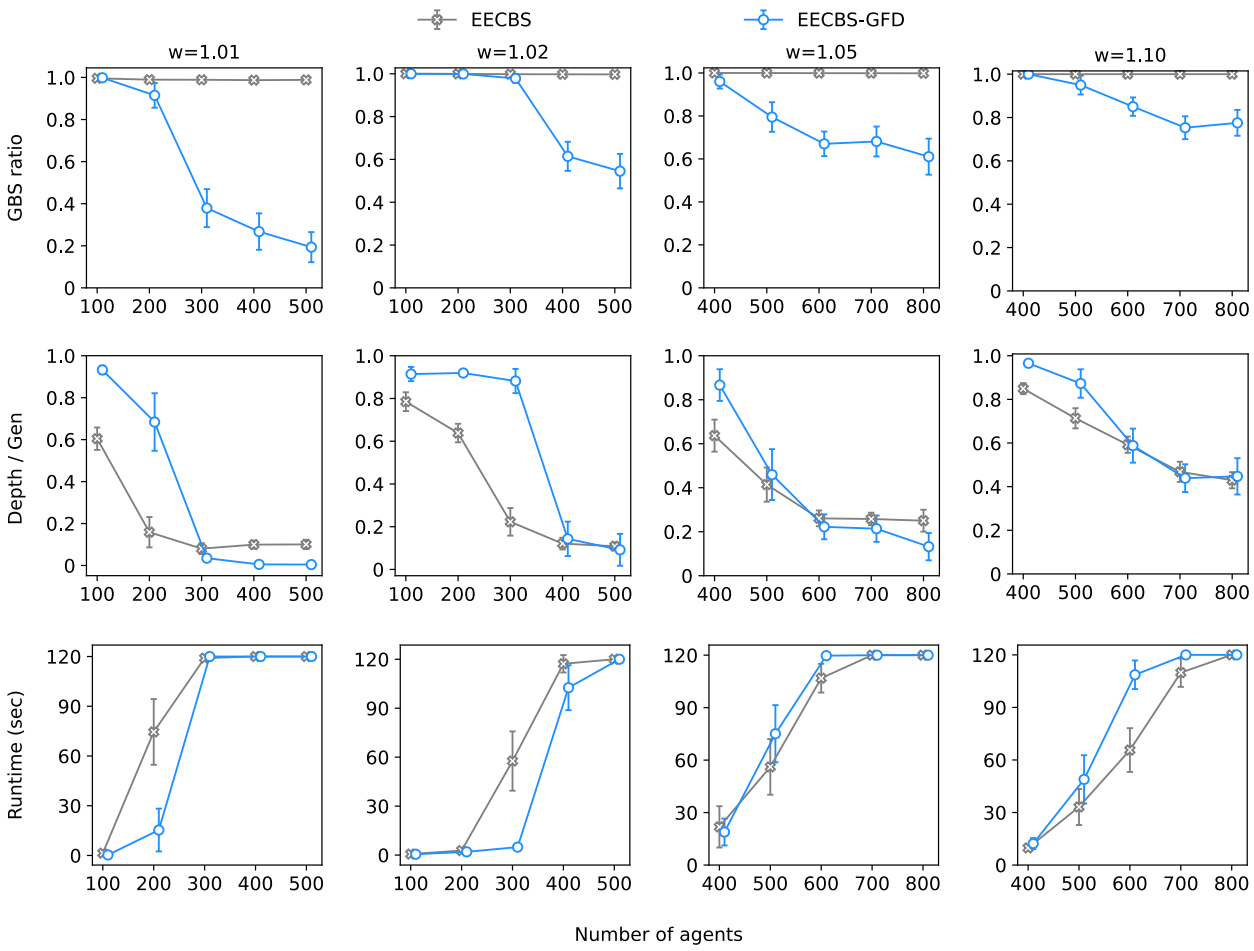


Figure 4.1: Average GBS ratios (see Equation 4.1), average resultant CT depth ratios (see Equation 4.2), and average runtimes (in seconds) of EECBS and EECBS-GFD for the 120-second runtime limit and each bounded-suboptimality factor w over all MAPF instances for each number of agents on the `ost003d` graph. If a MAPF algorithm fails to find a solution for a MAPF instance within the 120-second runtime limit, we set its runtime to 120 seconds. The vertical bars indicate the 95% confidence intervals.

4.1.1 High-Level Limitation

For EECBS (with or without Flex Distribution), the deeper a CT node is in the CT during the search, the fewer conflicts we expect it to have. Thus, we prefer a high ratio of CT depth to the number of CT nodes. With a positive maximum allowed flex, EECBS-GFD increases the threshold when finding a path for an agent in a CT node N. However, the low-level Focal Search can then use much of the maximum allowed flex, resulting in a huge increase in the SOC $C(N)$. In this case, even though each CT node remains locally bounded-suboptimal, CT node N may not be globally bounded-suboptimal and thus may not be selected

for expansion in the next few iterations. Then, EECBS-GFD may switch among CT nodes on different branches of CT that are globally bounded-suboptimal, rather than resolving conflicts along the branch that contains CT node N, thereby increasing the breadth of the CT. When EECBS switches among CT nodes on different branches of CT during the search, we say that it expands CT nodes more in a *breadth-first manner*. On the other hand, when EECBS resolves conflicts along the branch that contains the CT node N during the search, we say that it expands CT nodes more in a *depth-first manner*.

We define the *Globally Bounded-Suboptimal ratio* (GBS ratio) as

$$\text{GBS ratio} = \frac{GBS}{GEN}, \tag{4.1}$$

where *GBS* is the number of CT nodes that are globally bounded-suboptimal with respect to the lower bound LB on the SOC of an optimal solution when they are generated, and *GEN* is the number of CT nodes generated during the search. The higher the GBS ratio, the higher the percentage of CT nodes that are globally bounded-suboptimal when they are generated. The first row of Figure 4.1 shows the average GBS ratios of EECBS and EECBS-GFD for each bounded-suboptimality factor over all MAPF instances for each number of agents on the `ost003d` graph. As the bounded-suboptimality factor w increases, $w \cdot LB$ increases, and more CT nodes are globally bounded-suboptimal when they are generated, resulting in an increase of the GBS ratio as the bounded-suboptimality factor increases. On the other hand, as the number of agents increases, the maximum allowed flex can also increase since it is the sum of the flex from the paths of the other agents. Thus, EECBS-GFD can generate more CT nodes with high SOCs that are not globally bounded-suboptimal when they are generated, resulting in a decrease of the GBS ratio.

If EECBS (with or without Flex Distribution) finds a solution within the runtime limit, then we define the *resultant CT node* as the CT node whose list of paths is the solution. Otherwise, we define it as the deepest expanded CT node (where the depth is measured in the number of edges). The *resultant CT depth* is

the number of CT nodes between the root CT node and the resultant CT node. We then define the *resultant CT depth ratio* as

$$\text{resultant CT depth ratio} = \frac{DEP}{GEN}, \tag{4.2}$$

where *DEP* is the resultant CT depth and *GEN* is the number of CT nodes generated during the search. The second row of Figure 4.1 shows the average resultant CT depth ratios of EECBS and EECBS-GFD for each bounded-suboptimality factor over all MAPF instances for each number of agents on the `ost003d` graph. As the number of agents increases for each bounded-suboptimality factor, the average resultant CT depth ratio of EECBS-GFD decreases and can become lower than that of EECBS. This indicates that EECBS-GFD expands CT nodes more in a breadth-first manner than EECBS, especially when the number of agents is large.

For each bounded-suboptimality factor, the decrease of the GBS ratio as the number of agents increases indicates that EECBS-GFD expands CT more in a breadth-first manner, resulting in a decrease of the resultant CT depth and the number of generated CT nodes. That is, rather than resolving conflicts along a branch, EECBS-GFD switches among CT nodes on different branches of CT more often as the number of agents increases. Thus, the average runtime increases as the number of agents increases for each bounded-suboptimality factor, resulting in an increase in runtime. For MAPF instances with larger numbers of agents in the `ost003d` and `warehouse` graphs, the average runtimes of EECBS-GFD can even be higher than that of EECBS, as shown in the third row of Figure 4.1.

4.1.2 Low-Level Limitation

Suppose that EECBS expands CT node $\hat{N}$ and generates one of its child CT nodes N with the constraints $\Psi_i(N) = \Psi_i(\hat{N}) \cup \Psi_i$ for agent a_i, where Ψ_i is the set of constraints used to resolve the selected conflict during the expansion of CT node $\hat{N}$. Also, suppose that the bounded-suboptimality factor $w > 1$, the

maximum allowed flex $\Delta_{\max,i}(N) > 0$, and the cost $c_i^*(N)$ of a minimum-cost path that satisfies the constraints $\Psi_i(N)$ satisfies

$$w \cdot lb_i(\hat{N}) < c_i^*(N) \leq w \cdot lb_i(\hat{N}) + \Delta_{\max,i}(N), \tag{4.3}$$

which can occur if the new constraints Ψ_i substantially increase the cost of a minimum-cost path. In this case, if the low-level Focal Search finds a path for agent a_i without Flex Distribution, then its cost $c_i(N)$ satisfies

$$w \cdot lb_i(\hat{N}) < c_i^*(N) \leq c_i(N) \leq \tau_i(N) = w \cdot \max\{lb_i(\hat{N}), f_{\min,i}(N)\}, \tag{4.4}$$

which implies $lb_i(\hat{N}) < f_{\min,i}(N)$. Since $lb_i(\hat{N}) = \max\{lb_i(\hat{\hat{N}}), f_{\min,i}(\hat{N})\}$, where $\hat{\hat{N}}$ is the parent CT node of $\hat{N}$, $f_{\min,i}(\hat{N}) < f_{\min,i}(N)$, which indicates that the low-level Focal Search without Flex Distribution has to expand the v-t node n with the minimum $f_i(n)$ to find a path. Also, since

$$lb_i(\hat{N}) < f_{\min,i}(N) \equiv \max\{lb_i(\hat{N}), f_{\min,i}(N)\} = lb_i(N), \tag{4.5}$$

it holds that $lb_i(\hat{N}) < lb_i(N)$, which results in $LB(\hat{N}) < LB(N)$. In this case, we say that there is a *SOLB improvement* from CT node $\hat{N}$ to CT node N when EECBS adds the constraints Ψ_i to the set of constraints $\Psi_i(N)$ (i.e., $\Psi_i(N) = \Psi_i(\hat{N}) \cup \Psi_i$).

On the other hand, if the low-level Focal Search finds the path with GFD, due to the increase of the threshold $\tau_i(N) = w \cdot \max\{lb_i(\hat{N}), f_{\min,i}(N)\} + \Delta_{\max,i}(N)$, it is possible to find a path whose cost $c_i(N)$ satisfies

$$c_i(N) \leq w \cdot lb_i(\hat{N}) + \Delta_{\max,i}(N). \tag{4.6}$$

That is, $f_{\min,i}(N) \leq lb_i(\hat{N})$ may hold during the search of the low-level Focal Search with GFD, and thus $lb_i(N) = \max\{lb_i(\hat{N}), f_{\min,i}(N)\}$ may be equal to $lb_i(\hat{N})$, which results in $LB(N) = LB(\hat{N})$. In this

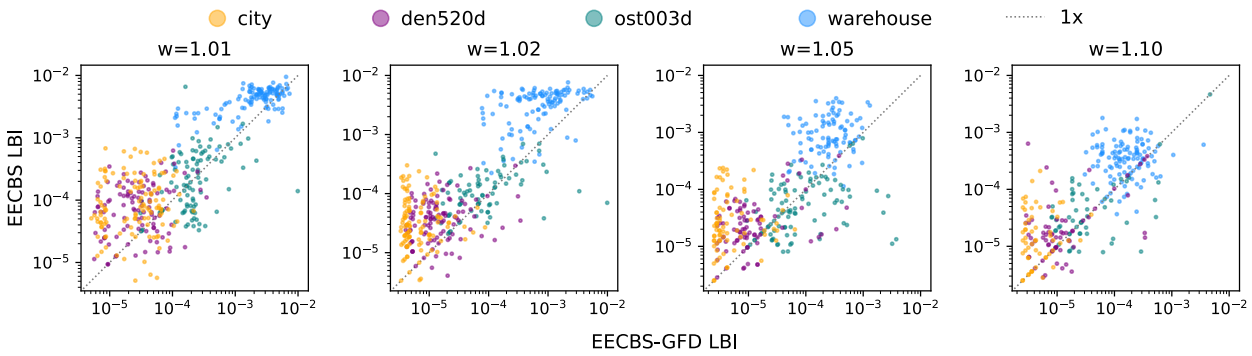


Figure 4.2: LBIs of EECBS versus EECBS-GFD for each bounded-suboptimality factor w over all MAPF instances on each graph. The x- and y-coordinates of each dot represent the LBIs of EECBS-GFD and EECBS, respectively. All the x- and y-coordinates are on logarithmic scales. Dots in yellow, purple, green, and blue represent the MAPF instances in `city`, `den520d`, `ost003d`, and `warehouse` graphs, respectively.

	$w = 1.01$		$w = 1.02$		$w = 1.05$		$w = 1.10$	
	EECBS	GFD	EECBS	GFD	EECBS	GFD	EECBS	GFD
`city`	0.82	0.16	0.88	0.12	0.90	0.10	0.88	0.12
`den520d`	0.76	0.19	0.86	0.14	0.87	0.13	0.78	0.22
`ost003d`	0.54	0.35	0.81	0.19	0.46	0.54	0.69	0.31
`warehouse`	0.85	0.15	0.94	0.06	0.94	0.06	0.82	0.18
Total	0.79	0.21	0.87	0.13	0.80	0.20	0.79	0.21

Table 4.1: MAPF instance comparisons of LBIs of EECBS and EECBS-GFD over all MAPF instances on each graph. For each bounded-suboptimality factor w, the columns "EECBS" contain the percentages of MAPF instances where EECBS has higher LBIs than EECBS-GFD, the columns "GFD" contain the percentages of MAPF instances where EECBS has higher LBIs than EECBS-GFD, and the row "Total" contains the percentages over all MAPF instances.

case, we say that there is no SOLB improvement from CT node $\hat{N}$ to CT node N when EECBS-GFD adds the constraints Ψ_i to the set of constraints $\Psi_i(N)$. This suggests that EECBS-GFD may require more effort to improve its LB than EECBS. We say that EECBS-GFD may have a worse LB improvement than EECBS.

To evaluate the LB improvement of EECBS and EECBS-GFD empirically, we then define the *relative LB Improvement* (LBI) as

$$\text{LBI} = \frac{LB^* - F(N_0)}{F(N_0)}, \tag{4.7}$$

where LB^* is the lower bound on the SOC of an optimal solution when EECBS (or, respectively, EECBS-GFD) terminates, and $F(N_0)$ is the F-value of the root CT node N_0, which is the LB value when the search starts. We compare the LBIs of EECBS-GFD and EECBS over all MAPF instances mentioned in

Section 3.8.1. As shown in Figure 4.2 and Table 4.1, there are more MAPF instances where EECBS has higher LBIs than EECBS-GFD than the other way around.

4.1.3 Summary of the Limitations of Greedy Flex Distribution (GFD)

To summarize the limitations of GFD, although EECBS-GFD typically runs faster than EECBS (see Section 3.8.2), the increase in threshold due to GFD may cause EECBS-GFD to expand CT nodes more in a breadth-first manner than EECBS. Also, EECBS-GFD may have a worse LB improvement than EECBS since there may be no SOLB improvement when it expands a CT node $\hat{N}$ and generates its child CT node N. Both of these effects may cause the generated CT node N not to be globally bounded-suboptimal, and thus it may not be expanded by EECBS-GFD in the next few iterations. In general, we would like to perform EECBS search with a high GBS ratio, i.e., by expanding CT nodes in a depth-first manner, where it can resolve conflicts efficiently along a branch by adding constraints, rather than in a breadth-first manner, where it explores many branches of the CT.

4.1.4 Key Observations on the Low-Level Focal Search

First, it is important for EECBS with Flex Distribution to reduce the amount of distributed flex so that a CT node can be globally bounded-suboptimal in the next few iterations, thereby expanding CT nodes in a depth-first manner. Second, when finding a path for an agent, it is important that the low-level Focal Search not only finds a path that satisfies the constraints of a CT node but also one that causes few conflicts with the paths of the other agents. These two observations help us design new Flex Distribution mechanisms.

4.2 Fractional Flex Distribution

If the maximum allowed flex is non-negative, then EECBS with Flex Distribution prevents determining too much distributed flex to an agent by determining only a fraction of the maximum allowed flex. On the

other hand, if the maximum allowed flex is negative, then it uses GFD to ensure that the generated CT node is locally bounded-suboptimal (see Theorem 3.4) and EECBS with Flex Distribution is still guaranteed to find a bounded-suboptimal solution to a MAPF instance if a solution exists (see Theorem 3.7). Suppose that EECBS with Flex Distribution expands CT node $\hat{N}$ and generates one of its child CT nodes N. It then needs to find a path for agent a_i that satisfies its constraints $\Psi_i(N) = \Psi_i(\hat{N}) \cup \Psi_i$, where Ψ_i is the set of constraints used to resolve the selected conflict during the CT node expansion. In this case, the distributed flex becomes

$$\Delta_i(N) = \begin{cases} \rho \cdot \Delta_{\max,i}(N), & \text{if } \Delta_{\max,i}(N) \geq 0 \\ \Delta_{\max\,.i}(N), & \text{otherwise,} \end{cases} \tag{4.8}$$

where $0 \leq \rho \leq 1$. We propose two ways for treating ρ as

- a constant user-specified hyperparameter, called *Fixed-Fractional Flex Distribution* (FFD), or
- a random variable, called *Random-Fractional Flex Distribution* (RFD).

FFD sets the distributed flex to a constant fraction of the maximum allowed flex. On the other hand, RFD sets the distributed flex to a random fraction of the maximum allowed flex. Empirically, EECBS with FFD (EECBS-FFD) and EECBS with RFD (EECBS-RFD) can have higher success rates than EECBS-GFD for the 120-second runtime limit (see Figure 4.6). We view FFD and RFD as stepping stones for our subsequent Flex Distribution mechanisms.

4.3 Conflict-Based Flex Distribution (CFD)

Based on Fractional Flex Distribution, we propose a heuristic approach called *Conflict-Based Flex Distribution* (CFD), which uses flex to avoid conflicts. Suppose that the EECBS with CFD (EECBS-CFD) expands a CT node $\hat{N}$ and generates one of its child CT nodes N that needs to find a path for agent a_i that satisfies its constraints $\Psi_i(N) = \Psi_i(\hat{N}) \cup \Psi_i$, where Ψ_i is the set of constraints used to resolve the selected

conflict during the CT node expansion. In this case, if the maximum allowed flex is non-negative (i.e., $\Delta_{\max,i}(N) \geq 0$), then EECBS-CFD determines flex proportionally to the number of conflicts that the path of agent a_i in CT node $\hat{N}$ has with the paths of the other agents $\{a_j \mid j \in [k] \setminus \{i\}\}$ in CT node $\hat{N}$. Otherwise, i.e., if the maximum allowed flex is negative (i.e., $\Delta_{\max,i}(N) < 0$), EECBS-CFD uses GFD to ensure that the CT node N remains locally bounded-suboptimal. Thus, the distributed flex $\Delta_i(N)$ via CFD is

$$\Delta_i(N) = \begin{cases} \rho_i(\hat{N}) \cdot \Delta_{\max,i}(N), \text{where } \rho_i(\hat{N}) = \frac{|\mathcal{X}_i(\hat{N})|}{|\mathcal{X}(\hat{N})|}, & \text{if } \Delta_{\max,i}(N) \geq 0 \\ \Delta_{\max\,.i}(N), & \text{otherwise,} \end{cases} \tag{4.9}$$

where $|\mathcal{X}_i(\hat{N})|$ is the number of conflicts that the path of agent a_i in CT node $\hat{N}$ has with the paths of the other agents $\{a_j \mid j \in [k] \setminus \{i\}\}$ in CT node $\hat{N}$, and $|\mathcal{X}(\hat{N})|$ is the number of conflicts among all pairs of paths in CT node $\hat{N}$ (i.e., $|\mathcal{X}(\hat{N})| = 0.5 \cdot \sum_{j=1}^{k} |\mathcal{X}_j(\hat{N})|$). That is, rather than treating the fraction as either a constant user-specified hyperparameter or a random variable, CFD uses the number of conflicts as a heuristic. For example, given a MAPF instance with three agents $\{a_1, a_2, a_3\}$ and a CT node $\hat{N}$ with conflicts $\langle a_1, a_2, v_1, t_1\rangle$, $\langle a_1, a_3, v_2, t_2\rangle$ and $\langle a_2, a_3, v_3, t_3\rangle$, suppose that EECBS-CFD selects conflict $\langle a_1, a_2, v_1, t_1\rangle$ and generates one of the child CT nodes N with the additional constraint $\langle a_1, v_1, t_1\rangle$. In this case, $|\mathcal{X}_i(\hat{N})| = 2$ and $|\mathcal{X}(\hat{N})| = 3$. If the maximum allowed flex is non-negative, then EECBS-CFD will only distribute $\rho_i(\hat{N}) = |\mathcal{X}_i(\hat{N})|/|\mathcal{X}(\hat{N})| = 2/3$ of it to find a path for agent a_1.

The rationale of CFD is to allow EECBS-CFD to implicitly resolve conflicts when it finds a path with the help of distributed flex, rather than explicitly resolving conflicts by using constraints. Compared to GFD, which uses as much flex as possible to implicitly resolve the conflicts, CFD saves some (non-negative) flex to implicitly resolve, in the future iteration, the conflicts between the paths of agents. On the other hand, if the maximum allowed flex is negative when EECBS-CFD finds a path for agent a_i in CT node N, then it uses GFD since one or more of the other agents $\{a_j \mid j \in [k] \setminus \{i\}\}$ have already used flex from

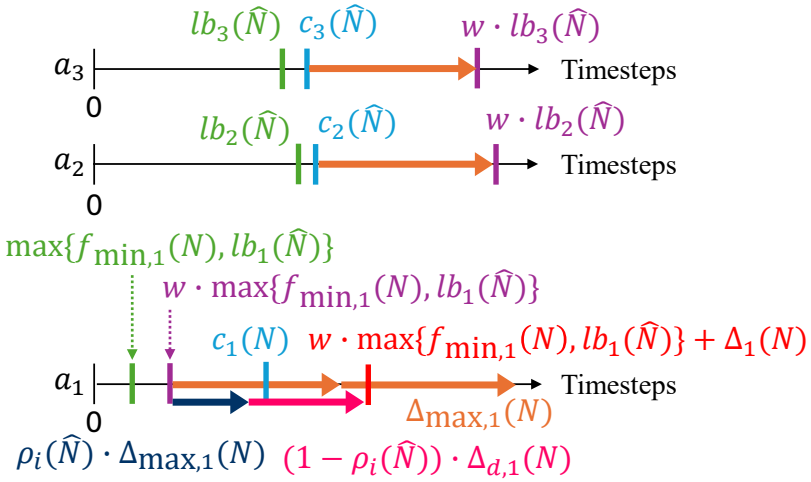


Figure 4.3: An example of how DFD increases the threshold with a positive maximum allowed flex from the flex of the paths of agents a_2 and a_3. Suppose that EECBS-DFD expands CT node $\hat{N}$ and generates one of its child CT nodes N, which requires finding a path for agent a_1. The green bars are the lower bounds, the cyan bars are the costs of the paths, the purple bars are the thresholds that are the bounded-suboptimality factor w larger than the lower bounds, and the red bar is the threshold increased by DFD. The orange arrows are the flex of the paths of agents a_2 and a_3, and the blue and pink arrows are the distributed flex by using DFD.

the path of agent a_i when their paths were found. Thus, EECBS-CFD should reduce the threshold $\tau_i(N)$ to ensure that $C(N) \leq w \cdot LB(N)$, which indicates that CT node N is still locally bounded-suboptimal after a path for agent a_i has been found. CFD improves the efficiency of EECBS with Flex Distribution, as shown in our empirical evaluation in Section 4.7.

4.4 Delay-Based Flex Distribution (DFD)

Based on CFD, we can also use flex to enable the low-level Focal Search to find a path that satisfies the constraints more quickly. Suppose that EECBS with Flex Distribution expands CT node $\hat{N}$ and generates one of its child CT nodes N. It then needs to find a path for agent a_i that satisfies the constraints $\Psi_i(N) = \Psi_i(\hat{N}) \cup \Psi_i$. The additional constraints Ψ_i may require the low-level Focal Search to find a path for agent a_i of higher cost than the path of agent a_i in CT node $\hat{N}$. Given a path that violates a constraint for an agent, we define the *delay* as the increase in the cost to find a new path that satisfies the constraint.

Accordingly, when the low-level Focal Search finds a path for agent a_i that satisfies the constraints $\Psi_i(N)$, we propose *Delay-Based Flex Distribution* (DFD) that determines flex $\Delta_{d,i}(N)$ according to the sum of the estimated delays from constraints $\Psi_i(N)$. That is,

$$\Delta_{d,i}(N) = \min(\Delta_{\max,i}(N)\ , \sum_{\psi_i \in \Psi_i(N)} d_{\psi_i}), \tag{4.10}$$

where d_{ψ_i} is the estimated delay needed to find a path that satisfies a constraint $\psi_i \in \Psi_i(N)$. Also, DFD requires the estimated sum of the delays to be non-negative, i.e.,

$$\sum_{\psi_i \in \Psi_i(N)} d_{\psi_i} \geq 0. \tag{4.11}$$

According to Equation 4.10, the $\Delta_{d,i}(N)$ is at most the maximum allowed flex $\Delta_{\max,i}(N)$. Also, since the sum of estimated delays is required to be non-negative, if $\Delta_{\max,i}(N) \geq 0$, then $\Delta_{d,i}(N) \geq 0$. Furthermore, DFD also incorporates a Flex Distribution mechanism similar to CFD, which results in

$$\Delta_i(N) = \begin{cases} \Delta_{d,i}(N) + \rho_i(\hat{N}) \cdot (\Delta_{\max,i}(N) - \Delta_{d,i}(N)), & \text{if } \Delta_{\max,i}(N) \geq 0 \\ \Delta_{\max,i}(N), & \text{otherwise,} \end{cases} \tag{4.12}$$

where $\rho_i(\hat{N}) = |\mathcal{X}_i(\hat{N})|/|\mathcal{X}(\hat{N})|$ is the same as that in Equation 4.9. The rationale is to distribute $\Delta_{d,i}(N)$ of maximum allowed flex for the delays caused by the constraints and then distribute the fraction ρ_i of the remaining allowed flex $\Delta_{\max,i}(N) - \Delta_{d,i}(N)$ for implicitly resolving the conflicts of the paths of agent a_i with the paths of the other agents $\{a_j \mid j \in [k] \setminus \{i\}\}$. Figure 4.3 shows an example of the variable relationships when the maximum allowed flex is positive.

Similar to EECBS-CFD, if the maximum allowed flex is negative when EECBS with DFD (EECBS-DFD) finds a path for agent a_i in CT node N, then it uses GFD to ensure that CT node N remains locally bounded-suboptimal.

Theorem 4.1. *EECBS-CFD and EECBS-DFD are guaranteed to find a bounded-suboptimal solution to a MAPF instance if a solution exists.*

Proof. For a CT node N, since the set of conflicts that the path of agent a_i has with the paths of the other agents $\{a_j \mid j \in [k] \setminus \{i\}\}$ is a subset of all the conflicts over each pair of paths in CT node N,

$$0 \leq \rho_i(\hat{N}) = \frac{|\mathcal{X}_i(\hat{N})|}{|\mathcal{X}(\hat{N})|} \leq 1. \tag{4.13}$$

If $\Delta_{\max,i}(N) \geq 0$, then the distributed flex of CFD satisfies $0 \leq \Delta_i(N) \leq \Delta_{\max,i}(N)$ according to Inequality 4.13. In terms of DFD, since the value of $\Delta_{d,i}(N)$ is at most the maximum allowed flex according to Equation 4.10, the first row of Equation 4.12 is equivalent to

$$\begin{aligned} \Delta_i(N) &= \Delta_{d,i}(N) + \rho_i \cdot (\Delta_{\max,i}(N) - \Delta_{d,i}(N)) \\ &= (1 - \rho_i) \cdot \Delta_{d,i}(N) + \rho_i \cdot \Delta_{\max,i}(N) \\ &\leq \Delta_{\max,i}(N), \end{aligned} \tag{4.14}$$

indicating that the distributed flex $\Delta_i(N)$ of DFD is at most $\Delta_{\max,i}$. Also, since $\Delta_{\max,i}(N) \geq 0$, $\Delta_{d,i} \geq 0$ according to Equation 4.10 and Inequality 4.11. Thus, according to the first row of Equation 4.12, the distributed flex of DFD satisfies $\Delta_i(N) \geq 0$. Together with Inequality 4.14, it holds that $0 \leq \Delta_i(N) \leq \Delta_{\max,i}(N)$. Inequality 4.14 can also be interpreted as the sum of the distributed flex $\rho_i \cdot \Delta_{\max,i}$ from CFD and $(1 - \rho_i) \cdot \Delta_{d,i}(N)$ from the estimated delays. That is, CFD is a special case of DFD where $\Delta_{d,i}(N) = 0$. On the other hand, if $\Delta_{\max,i}(N) < 0$, then both CFD and DFD use GFD to determine the distributed flex, i.e., $\Delta_i(N) = \Delta_{\max,i}(N) < 0$. Thus, the distributed flex of CFD and DFD both satisfy

Inequality 3.15. Thus, according to Theorem 3.7, EECBS-CFD and EECBS-DFD are guaranteed to find a bounded-suboptimal solution to a MAPF instance if a solution exists. □

We propose rule-based strategies for estimating the delay from a constraint. For each vertex or edge constraint, we assume that it can be satisfied by using a wait action. Thus, the delay from a vertex or edge constraint is 1. Li et al. [31, 34] propose symmetry breaking for EECBS that uses the geometry of the graph and the paths of the agents to reason about additional conflicts. Of those, we consider the *corridor conflict* and the *target conflict*. Since EECBS uses symmetry breaking during the search to improve efficiency, we can obtain the estimated delay caused by these conflicts without a large runtime overhead.

Given a graph $G = (V, E)$ of a MAPF instance, a corridor $Cr = Cr_0 \cup \{v_b, v_e\}$ is defined as a chain of connected vertices $Cr_0 \subseteq V$, each of degree 2, together with two endpoints $\{v_b, v_e\} \subseteq V$ connected to Cr_0. The length $|Cr|$ of the corridor Cr is the number of vertices in Cr_0 plus 1. A corridor conflict occurs if two agents, a_i and a_j, traverse the same corridor in opposite directions, resulting in a vertex or edge conflict. Suppose that agents a_i and a_j follow their paths in CT node $\hat{N}$, and a corridor conflict occurs when agent a_i traverses the corridor from vertex v_b to vertex v_e and agent a_j traverses the corridor in the opposite direction from vertex v_e to vertex v_b. To resolve this corridor conflict, Li et al. [31] use *range constraints* when EECBS generates the two child CT nodes of CT node $\hat{N}$. A *range constraint* prevents agent a_i from being at its exit endpoint v_e during timesteps $[0, t_{\min,j}]$, where $t_{\min,j}$ is the minimum number of timesteps needed for agent a_j to be at its exit endpoint v_b while satisfying the constraints $\Psi_j(\hat{N})$.* Due to the range constraint, agent a_i can only enter the corridor (i.e., be at its entry endpoint v_b) at or after timestep $t_{\min,j} + 1$. Thus, the minimum number of timesteps that agent a_i can be at its exit endpoint v_e is $t_{\min,j} + 1 + |Cr|$. DFD estimates the delay from such a range constraint ψ_i as $d_{\psi_i} = t_{\min,j} + 1 + |Cr| - t_{e,i}$, where $t_{e,i}$ is the timestep when agent a_i is at its exit endpoint v_e when following its path in CT node $\hat{N}$.

*A similar range constraint is added to the constraints of agent a_j when EECBS generates the other child CT node.

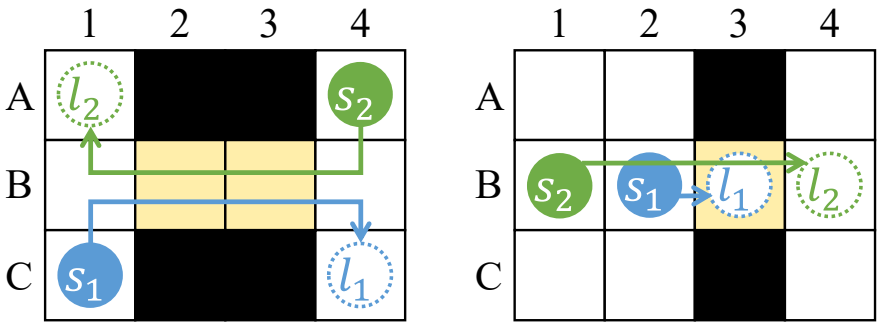


Figure 4.4: Examples of (a) a corridor conflict and (b) a target conflict between agents a_1 and a_2.

Figure 4.4 (a) shows an example of a corridor conflict between two agents a_1 and a_2 moving through a corridor $Cr = Cr_0 \cup \{v_b, v_e\}$ that contains a chain of connected vertices $Cr_0 = [B2, B3]$ and the endpoints $v_b = B1$ and $v_e = B4$. The length of the corridor is $|Cr| = 2 + 1 = 3$. Suppose that there are no constraints for both agents a_1 and a_2 at this point. Thus, agents a_1 and a_2 are traversing in opposite directions through the corridor Cr on their respective minimum-cost paths $[C1, B1, B2, B3, B4, C4]$ and $[A4, B4, B3, B2, B1, A1]$. These paths result in a corridor conflict in the form of the edge conflict $\langle a_1, a_2, B2, B3 \rangle$. The exit endpoints of the paths of agents a_1 and a_2 are $v_e = B4$ and $v_b = B1$, respectively. Since there are no constraints, the minimum timestep when agent a_1 is at its exit endpoint $B4$ is $t_{e,1} = 4$. To resolve the corridor conflict, EECBS generates two child CT nodes, each with a range constraint. One of the child CT nodes has the range constraint ψ_1 that prevents agent a_1 from being at its exit endpoint $B4$ during timesteps $[0, t_{\min,2}]$, where $t_{\min,2} = 4$ is the minimum timestep needed for agent a_2 to move from its start vertex $A4$ to its exit endpoint $B1$ since there are no constraints. Thus, when EECBS-DFD finds a path for agent a_1 that satisfies its constraints in this child CT node, DFD estimates the delay from the range constraint as $d_{\psi_1} = t_{\min,2} + 1 + |Cr| - t_{e,1} = 4 + 1 + 3 - 4 = 4$. Similarly, for the other child CT node with the range constraint for agent a_2, DFD estimates the delay from the range constraint also as 4.

A target conflict occurs when one agent is at the target vertex of another agent, although the other agent has already waited there permanently. Suppose that agents a_i and a_j follow their paths in CT node $\hat{N}$, and a target conflict occurs when agent a_j is at the target vertex l_i of agent a_i at timestep t when agent a_i has already there since timestep t_i permanently, i.e., $t_i = c_i(\hat{N}) \leq t$. To resolve this target conflict, Li et al. [31] use *length constraints* when EECBS generates two child CT nodes. One child CT node N has the length constraint that agent a_i must be at its target vertex l_i at or before timestep t and wait there permanently. That is, the cost of the path of agent a_i in CT node N should be $c_i(N) \leq t$. The other agents $\{a_m \mid m \in [k] \setminus \{i\}\}$ (including agent a_j) must not be at vertex l_i at or after timestep t. DFD estimates the delay from this length constraint as 0, since any delay may cause any of the agents to be at vertex l_i even later. The other child CT node N' of CT node $\hat{N}$ has the length constraint that agent a_i must be at its target vertex l_i after timestep t and wait there permanently. That is, the cost of the path of agent a_i in CT node N' should be $c_i(N') > t$. DFD estimates the delay from this length constraint for agent a_i as $d_{\psi_i} = t + 1 - c_i(\hat{N})$.

Figure 4.4 (b) shows an example of a target conflict that occurs between agents a_1 and a_2 when agent a_2 is at the target vertex $l_1 = B3$ of agent a_1, although agent a_1 has already waited there permanently since timestep $t_1 = 1$. The paths of agents a_1 and a_2 are $[B2, B3]$ and $[B1, B2, B3, B4]$, respectively. To resolve this target conflict, EECBS generates two child CT nodes with length constraints, respectively. One CT node has the length constraint that agent a_1 must be at its target vertex $B3$ at or before timestep 2 and wait there permanently, i.e., the cost of the path for agent a_1 should be at most 2. Agent a_2 must not be at vertex $B3$ at or after timestep 2, and DFD estimates the delay from this length constraint for agents a_1 and a_2 as 0.[†] The other child CT node has the length constraint that agent a_1 must be at its target vertex $B3$ and wait there after timestep 2 permanently, i.e., the cost of the path for agent a_1 should be at least $2 + 1 = 3$. In this case, DFD estimates the delay from this length constraint for agent a_1 as $3 - 1 = 2$.

[†]Since agent a_2 needs at least 2 timesteps to be at vertex $B3$, EECBS cannot find a path for agent a_2 that satisfies the length constraint. Thus, this CT node is pruned.

4.5 Mixed-Strategy Flex Distribution (MFD)

Although both CFD and DFD reduce the amount of distributed flex, there may still be situations where agents use too much flex, thereby increasing the SOC of a CT node. Thus, we propose a Mixed-Strategy Flex Distribution (MFD) based on CFD and DFD.

Suppose that EECBS expands a CT node $\hat{N}$ and generates one of its child CT nodes N with constraint $\Psi_i(N) = \Psi_i(\hat{N}) \cup \Psi_i$ for agent a_i, where Ψ_i is the set of constraints used to resolve the selected conflict during the CT node expansion. If the maximum allowed flex $\Delta_{\max,i}(N)$ is negative, then MFD uses GFD (i.e., determining $\Delta_i(N) = \Delta_{\max,i}(N) < 0$) to ensure that the CT node N is locally bounded-suboptimal after the low-level Focal Search finds a path for agent a_i that satisfies the constraints $\Psi_i(N)$.

If the maximum allowed flex is non-negative (i.e., $\Delta_{\max,i}(N) \geq 0$), MFD first uses DFD to determine the distributed flex $\Delta_i(N)$ via Equation 4.12. Then, MFD assumes that the low-level Focal Search uses all distributed flex, resulting in the found path for agent a_i having cost $c_i(N) = w \cdot lb_i(\hat{N}) + \Delta_i(N)$. Based on this assumption, MFD checks whether the SOC of CT node N is globally bounded-suboptimal according to Definition 3.5. That is, if

$$\sum_{j \in [k] \setminus \{i\}} c_j(N) + w \cdot lb_i(\hat{N}) + \Delta_i(N) \leq w \cdot LB, \tag{4.15}$$

then MFD uses the distributed flex $\Delta_i(N)$. Otherwise, MFD switches to CFD to determine the distributed flex via Equation 4.9, i.e., it sets $\Delta_{d,i}(N) = 0$ in Equation 4.12. Again, MFD checks whether Inequality 4.15 holds. If so, then MFD uses the distributed flex $\Delta_i(N)$. Otherwise, MFD tries to reduce the maximum allowed flex $\Delta_{\max,i}(N)$ based on the CT node N_F with the minimum F-value in CLEANUP$_H$, i.e, the CT node N_F whose $F(N_F) = LB$. For the paths of the other agents $\{a_j \mid j \in [k] \setminus \{i\}\}$, if their SOLB $\sum_{j \in [k] \setminus \{i\}} lb_j(N)$ satisfies

$$\sum_{j \in [k] \setminus \{i\}} lb_j(N_F) < \sum_{j \in [k] \setminus \{i\}} lb_j(N) \tag{4.16}$$

and their SOC $\sum_{j \in [k] \setminus \{i\}} c_j(N)$ satisfies

$$\sum_{j \in [k] \setminus \{i\}} c_j(N) < w \cdot \sum_{j \in [k] \setminus \{i\}} lb_j(N_F), \tag{4.17}$$

then MFD modifies the maximum allowed flex $\Delta_{\max,i}(N)$ from Equation 3.8 (see Definition 3.8) to

$$\tilde{\Delta}_{\max,i}(N) = w \cdot \sum_{j \in [k] \setminus \{i\}} lb_j(N_F) - \sum_{j \in [k] \setminus \{i\}} c_j(N). \tag{4.18}$$

Inequality 4.16 ensures that the new maximum allowed flex $\tilde{\Delta}_{\max,i}(N)$ is smaller than the original maximum allowed flex $\Delta_{\max,i}(N)$ that is derived from Equation 3.8. Inequality 4.17 ensures that the new maximum allowed flex $\tilde{\Delta}_{\max,i}(N)$ remains positive. Also, if both Inequalities 4.16 and 4.17 hold, then

$$\sum_{j \in [k] \setminus \{i\}} c_j(N) < w \cdot \sum_{j \in [k] \setminus \{i\}} lb_j(N_F) < w \cdot \sum_{j \in [k] \setminus \{i\}} lb_j(N), \tag{4.19}$$

implying that the maximum allowed flex $\Delta_{\max,i}(N)$ must satisfy

$$\Delta_{\max,i}(N) = w \cdot \sum_{j \in [k] \setminus \{i\}} lb_j(N) - \sum_{j \in [k] \setminus \{i\}} c_j(N) > 0. \tag{4.20}$$

That is, Inequalities 4.16 and 4.17 cannot be both satisfied when $\Delta_{\max,i}(N) = 0$. After MFD modifies the maximum allowed flex, it uses CFD (i.e., the first row of Equation 4.9) to determine the distributed flex $\Delta_i(N)$, resulting in $0 \leq \Delta_i(N) \leq \tilde{\Delta}_{\max,i}(N) < \Delta_{\max,i}(N)$. If the low-level Focal Search with MFD finds a path for agent a_i in CT node N that satisfies the constraints $\Psi_i(N)$ and terminates, then the cost of the

path satisfies $c_i(N) \leq w \cdot lb_i(N) + \Delta_i(N)$. If $lb_i(N) = lb_i(N_F)$ occurs during the search, the SOC $C(N)$ of CT node N then satisfies

$$\begin{aligned} C(N) &= \sum_{j \in [k] \setminus \{i\}} c_j(N) + c_i(N) \\ &\leq \sum_{j \in [k] \setminus \{i\}} c_j(N) + w \cdot lb_i(N) + \Delta_i(N) \\ &= \sum_{j \in [k] \setminus \{i\}} c_j(N) + w \cdot lb_i(N_F) + \Delta_i(N) \\ &\leq \sum_{j \in [k] \setminus \{i\}} c_j(N) + w \cdot lb_i(N_F) + \tilde{\Delta}_{\max,i}(N) \\ &\leq \sum_{j \in [k] \setminus \{i\}} c_j(N) + w \cdot lb_i(N_F) + \sum_{j \in [k] \setminus \{i\}} (w \cdot lb_j(N_F) - c_j(N)) = w \cdot LB, \end{aligned} \tag{4.21}$$

which indicates that the CT node N is globally bounded-suboptimal when it is generated and thus has a chance to be selected in the next few iterations. If at least one of the Inequalities 4.16 and 4.17 does not hold, then MFD uses zero distributed flex (i.e., $\Delta_i(N) = 0$). Since MFD reduces the distributed flex $\Delta_i(N)$ while still maintaining $0 \leq \Delta_i(N) \leq \Delta_{\max,i}(N)$ if the maximum allowed flex $\Delta_{\max,i}(N)$ is non-negative and uses GFD otherwise, according to Theorem 3.7, EECBS with MFD is guaranteed to find a bounded-suboptimal solution to a MAPF instance if a solution exists (i.e., bounded-suboptimal and solution-complete).

In short, the rationale of MFD is that if the maximum allowed flex $\Delta_{\max,i}(N) < 0$, then it uses GFD to ensure that CT node N remains locally bounded-suboptimal. Otherwise (i.e., $\Delta_{\max,i}(N) \geq 0$), then MFD starts by determining the distributed flex via DFD and reduces it in a hierarchical manner when the cost of the path of agent a_i that uses all the distributed flex results in CT node N not being globally bounded-suboptimal. That is, MFD determines the distributed flex such that CT node N is likely to be globally bounded-suboptimal when it is generated and thus can be expanded in the next few iterations. Figure 4.5 shows how MFD determines the distributed flex $\Delta_i(N)$ for finding a path for agent a_i satisfying

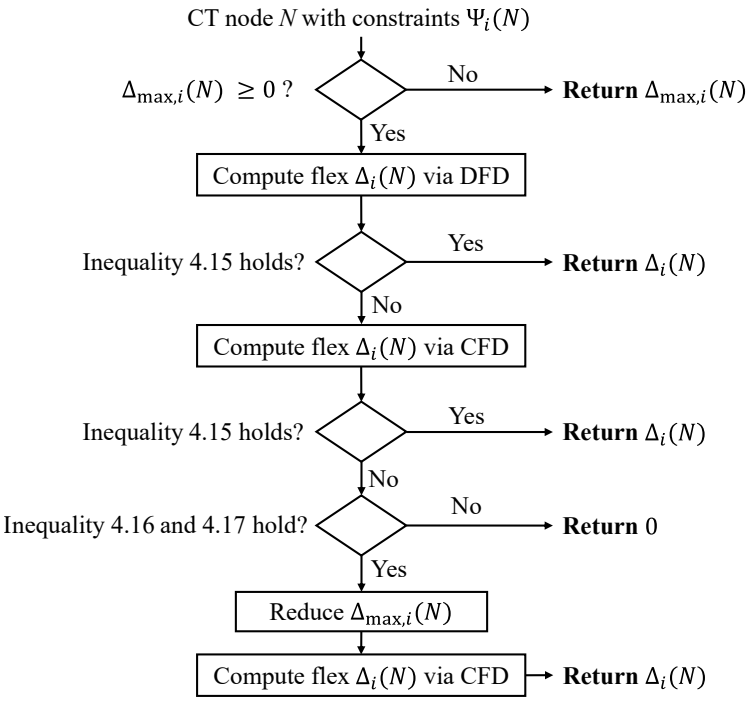


Figure 4.5: The flow chart of determining the distributed flex $\Delta_i(N)$ via MFD. The rectangular blocks represent functions, and the diamond-shaped blocks represent conditions.

constraints $\Psi_i(N)$ in CT node N. The input is the CT node N with the constraints $\Psi_i(N)$, and the output is the distributed flex.

4.6 Low-Level Focal-A* (FA*) for Congested MAPF Instances

Since EECBS-GFD has a worse LB improvement than EECBS (see Section 4.1.2), we propose a new low-level heuristic search algorithm that combines Focal Search and A*, called the low-level *Focal-A** (FA*).

When the low-level FA* finds a path for an agent a_i that satisfies the constraints $\Psi_i(N)$ in a CT node N, it maintains the same lists for v-t nodes as the low-level Focal Search. That is, when the low-level FA* finds a path for agent a_i that satisfies the constraints $\Psi_i(N)$ in CT node N, it maintains

- CLOSED$_L$, that contains all expanded v-t nodes,
- OPEN$_L$, that contains all generated but not yet expanded v-t nodes (i.e., the frontier of the search tree of v-t nodes), sorted in increasing order of their f_i-values, and

- FOCAL$_L$, that contains the v-t nodes in OPEN$_L$ whose f_i-values are no larger than τ_i, sorted in increasing order of their number of conflicts (i.e., the x_i-values).

The core idea of the low-level FA* is finding a path by expanding v-t nodes from FOCAL$_L$ and then increasing the lower bound $f_{\min,i}(N)$ by expanding v-t nodes from OPEN$_L$. Thus, the low-level FA* consists of two steps:

(i) It repeatedly selects v-t nodes with the minimum number of conflicts in FOCAL$_L$ for expansion until a path $p_i(N)$ with its cost $c_i(N)$ that satisfies the constraints $\Psi_i(N)$ is found, and

(ii) it then iteratively selects v-t nodes with the minimum f_i-value in OPEN$_L$ for expansion until either a minimum-cost path that satisfies the constraints $\Psi_i(N)$ is found or the selected v-t node has its f_i-value at least the cost of the path found in Step (i), indicating that $c_i(N)$ is the cost of a minimum-cost path that satisfies the constraints $\Psi_i(N)$.

That is, the low-level FA* first runs the low-level Focal Search until a path is found and then switches to the low-level A* to increase the lower bound $f_{\min,i}(N)$. When the low-level FA* terminates, the cost of a minimum-cost path that satisfies the constraints $\Psi_i(N)$ is either the f_i-value of its last selected v-t node in Step (ii) or the cost of the path found in Step (i). Compared to the low-level Focal Search that returns the path $p_i(N)$ and the lower bound $lb_i(N)$, the low-level FA* returns the path $p_i(N)$ found in Step (i) and the cost found in Step (ii) as a lower bound on the cost of a minimum-cost path that satisfies the constraints $\Psi_i(N)$ since it is, in fact, equal to the cost of a minimum-cost path that satisfies the constraints $\Psi_i(N)$. That is, the low-level FA* runs its search on the same search tree as the low-level Focal Search, but it additionally expands v-t nodes from OPEN$_L$ to replace the lower bound of the low-level Focal Search with the cost of a minimum-cost path that satisfies the constraints $\Psi_i(N)$.

Algorithm 4.1 (with its tool functions in Algorithm 4.2) shows the pseudo-code of the low-level FA* with Flex Distribution. The blue text marks the differences between it and the low-level Focal Search with

Flex Distribution. In Step (i), the low-level FA* uses a boolean variable $isPathFound$ to check if a path has been found during the search and, if so, uses the variables p_i, c_i, and lb_i to store the path, the cost, and the lower bound, respectively [Line 6]. If the low-level FA* finds a path while expanding v-t nodes in FOCAL$_L$, then it sets variable $isPathFound$ to true (i.e., a path has been found). In this case, the low-level FA* returns the path with its cost and lower bound and terminates [Line 33] if one of the following conditions holds. The first condition is that $lb_i = c_i$. Then, the found path is a minimum-cost path. The second condition is that the constraints $\Psi_i(N) = \emptyset$. Then, since $h_i(s_i)$ is the cost of a minimum-cost path from the start vertex s_i to the target vertex l_i, $lb_i = f_{\min,i}(N) = f_i(n_0) = h_i(s_i)$, and it cannot be improved by expanding v-t nodes in OPEN$_L$. Otherwise, the low-level FA* starts Step (ii). It discards FOCAL$_L$ and only operates OPEN$_L$ during the following search. If the low-level FA* finds another path while expanding v-t nodes in OPEN$_L$, then this path is the minimum-cost path that satisfies the constraints $\Psi_i(N)$ and thus the low-level FA* can update the lower bound lb_i accordingly [Line 30]. Otherwise, if the minimum f_i-value in OPEN$_L$ reaches the cost c_i of the found path in Step (i), then this indicates that the found path (by expanding the v-t node in FOCAL$_L$) is a minimum-cost path. In this case, the low-level FA* returns the path with the lower bound equal to its cost [Line 22].

4.7 Empirical Evaluation

The configuration setting for the empirical evaluation in this section is the same as that introduced in Section 3.8.1 (see Table 3.2). We use the 120-second runtime limit unless mentioned otherwise. Apart from EECBS and EECBS-GFD, we implement different Flex Distribution mechanisms, as shown in Table 4.2. We evaluate the success rates of EECBS and EECBS with different Flex Distribution mechanisms for the 120-second runtime limit. Then, we focus on the efficiency and the solution quality comparison between EECBS, EECBS-GFD, and EECBS-MFD. We also provide insights and a case study on why EECBS-MFD is

MAPF Algorithm abbreviation	Full name
EECBS	Explicit Estimation Conflict-Based Search
EECBS-GFD	EECBS with Greedy Flex Distribution
EECBS-FFD	EECBS with Fixed-Fractional Flex Distribution
EECBS-RFD	EECBS with Random-Fractional Flex Distribution
EECBS-CFD	EECBS with Conflict-Based Flex Distribution
EECBS-DFD	EECBS with Delay-Based Flex Distribution
EECBS-MFD	EECBS with Mixed-Strategy Flex Distribution

Table 4.2: EECBS and its variants with different Flex Distribution mechanisms.

more efficient than EECBS-GFD. Lastly, we evaluate the efficiency of low-level FA* compared to low-level Focal Search on congested MAPF instances.

4.7.1 Performance Comparison

Figure 4.6 shows the success rates of EECBS and EECBS with different Flex Distribution mechanisms for the 120-second runtime limit, each bounded-suboptimality factor w, and each number of agents over all MAPF instances on each graph. In general, for the small bounded-suboptimality factors $w = \{1.01, 1.02\}$, EECBS-GFD has higher success rates than EECBS. However, as the bounded-suboptimality factor increases (e.g., with $w = 1.10$ on the `den520d` and the `ost003d` graphs), EECBS-GFD can have lower success rates than EECBS. On the other hand, with the bounded-suboptimality factors $w = \{1.01, 1.02, 1.05, 1.10\}$, EECBS-CFD and EECBS-DFD typically have higher success rates than EECBS-GFD. Furthermore, since MFD combines DFD and CFD hierarchically, the success rates of EECBS-MFD typically converge to the ones of either EECBS-CFD or EECBS-DFD and can even be higher than both, especially for MAPF instances on the `city` graph.

Figure 4.7, Table 4.3, and Table 4.4 show the success rates of EECBS, EECBS-GFD, and EECBS-MFD for the 120-second runtime limit and each bounded-suboptimality factor w over all MAPF instances on each graph. For the 120-second runtime limit and the bounded-suboptimality factor $w = 1.01$, using MFD results in a more than threefold increase in the success rate of EECBS over all MAPF instances on all graphs, raising it from 0.18 to 0.598. Also, the success rate of EECBS-MFD (0.598) is higher than

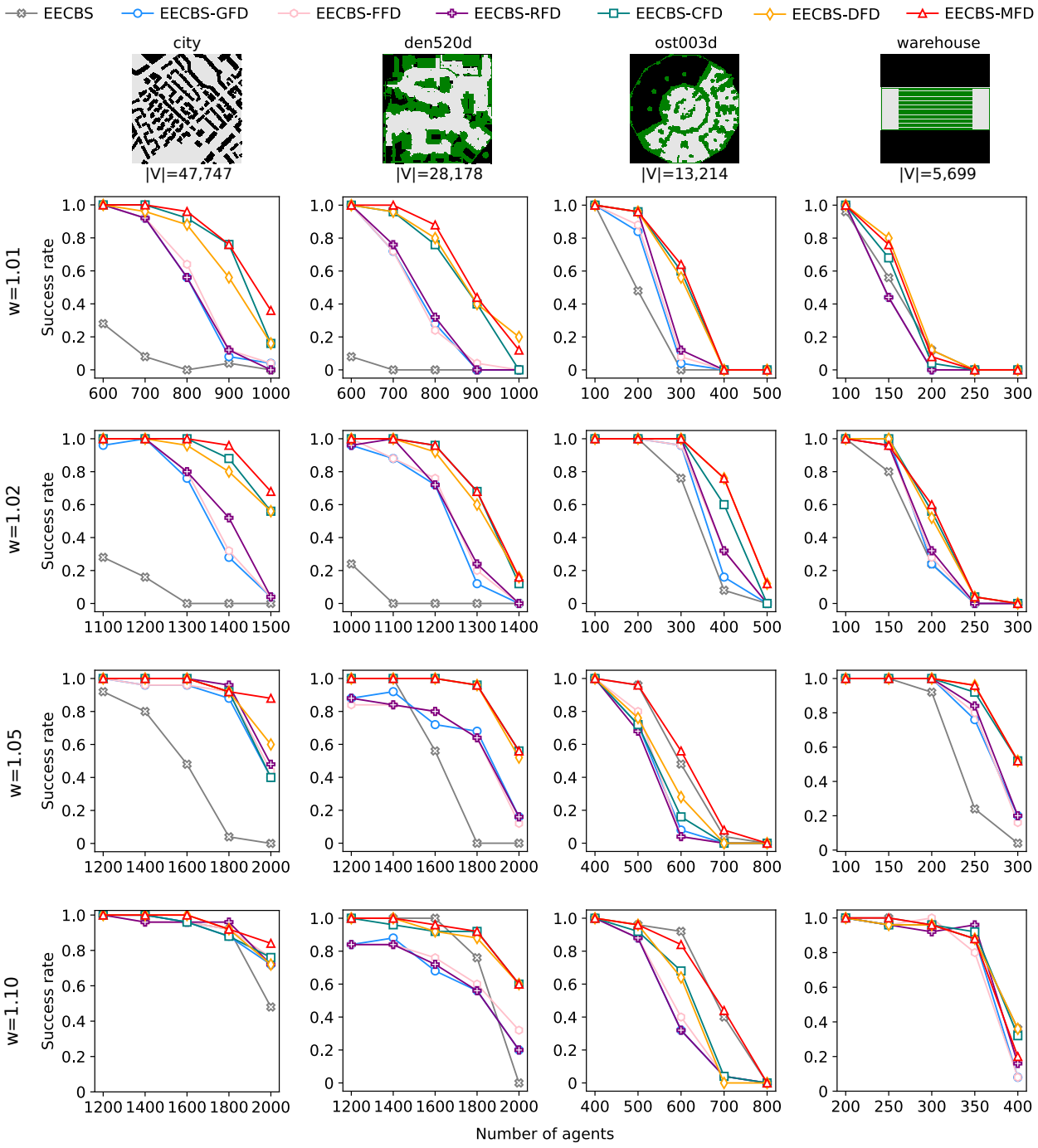


Figure 4.6: Success rates of EECBS and EECBS with different Flex Distribution mechanisms for the 120-second runtime limit, each bounded-suboptimality factor w, and each number of agents over all MAPF instances on each graph. $|V|$ indicates the number of vertices of each graph.

that of EECBS-GFD (0.396). As shown in Tables 4.3 and 4.4, with the bounded-suboptimality factors $w = \{1.01, 1.02, 1.05\}$, EECBS-MFD has higher success rates than EECBS and EECBS-GFD over all MAPF

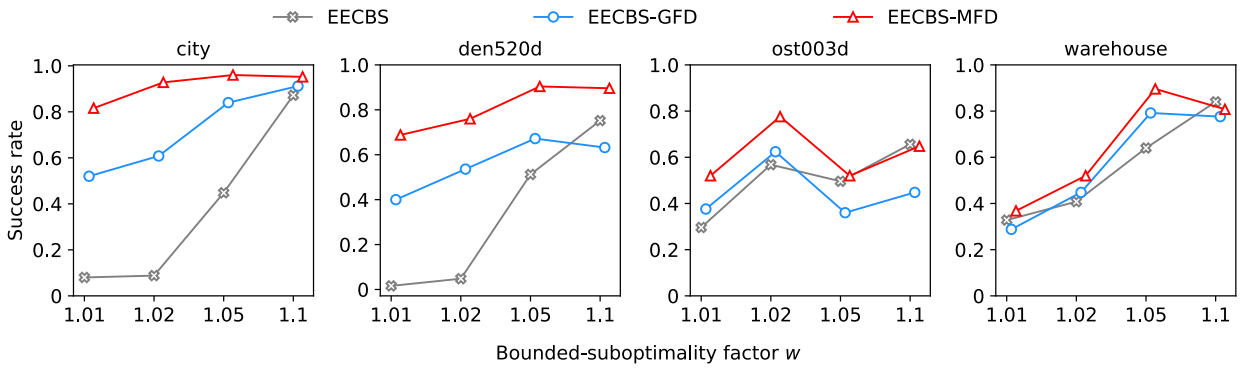


Figure 4.7: Success rates of EECBS, EECBS-GFD, and EECBS-MFD for the 120-second runtime limit and each bounded-suboptimality factor w over all MAPF instances on each graph.

	$w = 1.01$		$w = 1.02$		$w = 1.05$		$w = 1.10$	
	EECBS	MFD	EECBS	MFD	EECBS	MFD	EECBS	MFD
`city`	0.080	0.816	0.088	0.928	0.448	0.960	0.872	0.952
`den520d`	0.016	0.688	0.048	0.760	0.512	0.904	0.752	0.896
`ost003d`	0.296	0.520	0.568	0.776	0.496	0.520	0.656	0.648
`warehouse`	0.328	0.368	0.408	0.520	0.640	0.896	0.840	0.808
Total	0.180	0.598	0.278	0.746	0.524	0.820	0.780	0.826

Table 4.3: Success rates of EECBS and EECBS-MFD for the 120-second runtime limit over all MAPF instances. For each bounded-suboptimality factor w, the columns "EECBS" contain the success rates of EECBS, the columns "MFD" contain the success rates of EECBS-MFD, and the row "Total" contains the success rates over all MAPF instances.

instances on each graph. In particular, while EECBS and EECBS-GFD have the success rates of 0.08 and 0.52, respectively, over all MAPF instances on the `city` graph (which is the largest graph in our empirical evaluation) for the bounded-suboptimality factor $w = 1.01$, EECBS-MFD has the success rate of 0.816. This indicates that EECBS-MFD finds bounded-suboptimal solutions more quickly than EECBS and EECBS-GFD, especially with small bounded-suboptimality factors. As the bounded-suboptimality factor reaches $w = 1.10$, EECBS-MFD has higher success rates than EECBS and EECBS-GFD over all MAPF instances on large graphs, such as the `city` and `den520d` graphs, and is still competitive over all MAPF instances on the `ost003d` and `warehouse` graphs. Additionally, Figure 4.8 shows the average runtimes of EECBS, EECBS-GFD, and EECBS-MFD. Typically, the higher the success rates, the lower the average runtimes.

	$w = 1.01$		$w = 1.02$		$w = 1.05$		$w = 1.10$	
	GFD	MFD	GFD	MFD	GFD	MFD	GFD	MFD
`city`	0.520	0.816	0.608	0.928	0.840	0.960	0.912	0.952
`den520d`	0.400	0.688	0.536	0.760	0.672	0.904	0.632	0.896
`ost003d`	0.376	0.520	0.624	0.776	0.360	0.520	0.448	0.648
`warehouse`	0.288	0.368	0.448	0.520	0.792	0.896	0.776	0.808
Total	0.396	0.598	0.554	0.746	0.666	0.820	0.692	0.826

Table 4.4: Success rates of EECBS-GFD and EECBS-MFD for the 120-second runtime limit over all MAPF instances. For each bounded-suboptimality factor w, the columns "GFD" contain the success rates of EECBS-GFD, the columns "MFD" contain the success rates of EECBS-MFD, and the row "Total" contains the success rates over all MAPF instances.

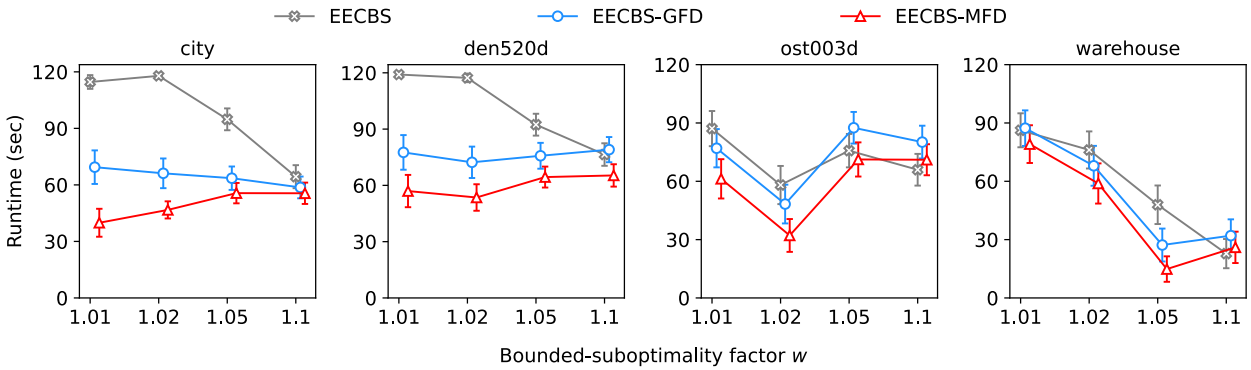


Figure 4.8: Average runtimes (in seconds) of EECBS, EECBS-GFD, and EECBS-MFD for the 120-second runtime limit and each bounded-suboptimality factor w over all MAPF instances on each graph. The vertical bars indicate the 95% confidence intervals.

Figure 4.9 shows the runtimes of EECBS versus EECBS-MFD for each bounded-suboptimality factor w and the 120-second runtime limit over all MAPF instances on each graph. EECBS-MFD typically has lower runtimes than EECBS and can even run 100 times faster. As shown in Table 4.5, there are more MAPF instances for which EECBS-MFD has lower runtimes than EECBS than the other way around. For each bounded-suboptimality factor w, EECBS-MFD has lower runtimes than EECBS on at least 50% of all MAPF instances (see the "Total" row). With small bounded-suboptimality factors $w = \{1.01, 1.02\}$, there is no MAPF instance where EECBS has a lower runtime than EECBS-MFD on the large `city` and `den520d` graphs. As the bounded-suboptimality factor w reaches 1.10, this advantage diminishes. As shown in Table 4.5, with the bounded-suboptimality $w = 1.10$, EECBS-MFD still has lower runtimes than EECBS on 55% of all MAPF instances.

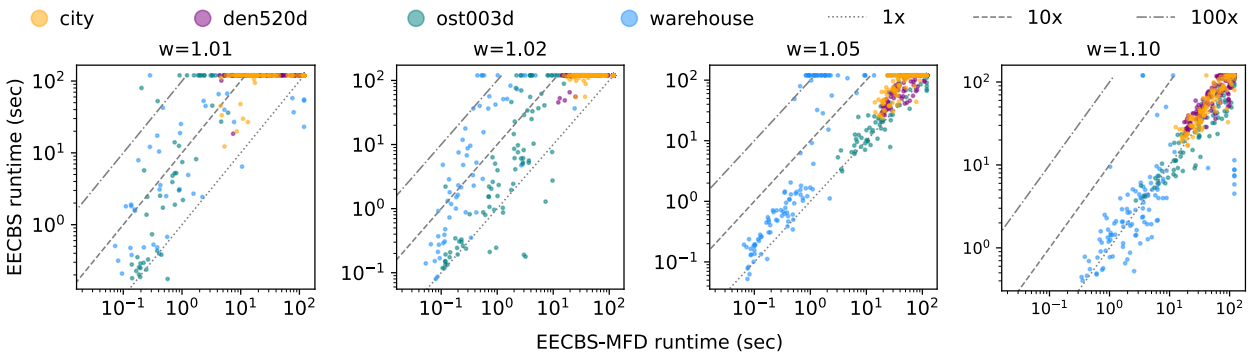


Figure 4.9: Runtimes (in seconds) of EECBS versus EECBS-MFD for the 120-second runtime limit and each bounded-suboptimality factor w over all MAPF instances on each graph. The x- and y-coordinates of each dot represent the runtimes of EECBS-MFD and EECBS, respectively. All the x- and y-coordinates are on logarithmic scales. Dots in yellow, purple, green, and blue represent the MAPF instances in the `city`, `den520d`, `ost003d`, and `warehouse` graphs, respectively.

	$w = 1.01$		$w = 1.02$		$w = 1.05$		$w = 1.10$	
	EECBS	MFD	EECBS	MFD	EECBS	MFD	EECBS	MFD
`city`	0.00	0.82	0.00	0.93	0.01	0.95	0.17	0.80
`den520d`	0.00	0.69	0.00	0.76	0.07	0.83	0.20	0.71
`ost003d`	0.03	0.49	0.09	0.69	0.19	0.37	0.46	0.23
`warehouse`	0.06	0.34	0.01	0.51	0.05	0.85	0.41	0.46
Total	0.02	0.58	0.02	0.72	0.08	0.75	0.31	0.55

Table 4.5: MAPF instance comparisons of the runtimes of EECBS and EECBS-MFD over all MAPF instances on each graph. For each bounded-suboptimality factor w, the columns "EECBS" contain the percentages of MAPF instances where EECBS has lower runtimes than EECBS-MFD, the columns "MFD" contain the percentages of MAPF instances where EECBS-GFD has lower runtimes than EECBS, and the row "Total" contains the percentage over all MAPF instances.

Figure 4.10 shows the runtimes of EECBS-GFD versus EECBS-MFD for each bounded-suboptimality factor w over all MAPF instances. EECBS-GFD can have 10 times higher runtimes than EECBS-MFD. With the bounded-suboptimality factors $w = \{1.05, 1.10\}$, EECBS-GFD can have 100 times higher runtimes than EECBS-MFD on the `warehouse` graph. As shown in Table 4.6, for each bounded-suboptimality factor, the percentages of MAPF instances where EECBS-MFD has lower runtimes than EECBS-GFD are higher than those of the other way around over all graphs.

To show the effectiveness of GFD and MFD in accelerating EECBS, we evaluate the conflict resolution efficiency (see Equation 3.26) of EECBS, EECBS-GFD, and EECBS-MFD. As shown in Figure 4.11, EECBS-MFD has a higher conflict resolution efficiency than EECBS, indicating its good effectiveness in resolving

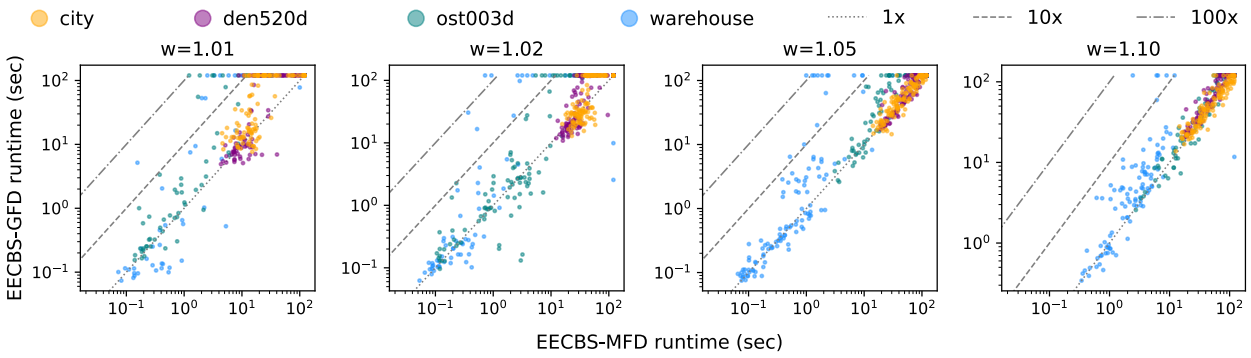


Figure 4.10: Runtimes (in seconds) of EECBS-GFD versus EECBS-MFD for the 120-second runtime limit and each bounded-suboptimality factor w over all MAPF instances on each graph. The x- and y-coordinates of each dot represent the runtime of EECBS-MFD and EECBS-GFD, respectively. All the x- and y-coordinates are on logarithmic scales. Dots in yellow, purple, green, and blue represent the MAPF instances in the `city`, `den520d`, `ost003d`, and `warehouse` graphs, respectively.

	$w = 1.01$		$w = 1.02$		$w = 1.05$		$w = 1.10$	
	GFD	MFD	GFD	MFD	GFD	MFD	GFD	MFD
`city`	0.13	0.69	0.38	0.54	0.27	0.69	0.42	0.56
`den520d`	0.20	0.49	0.34	0.42	0.36	0.54	0.04	0.86
`ost003d`	0.06	0.46	0.28	0.50	0.10	0.42	0.14	0.50
`warehouse`	0.18	0.18	0.19	0.34	0.34	0.56	0.22	0.60
Total	0.14	0.46	0.30	0.45	0.27	0.55	0.20	0.63

Table 4.6: MAPF instance comparisons of the runtimes of EECBS-GFD and EECBS-MFD over all MAPF instances on each graph. For each bounded-suboptimality factor w, the columns "GFD" contain the percentages of MAPF instances where EECBS-GFD has lower runtimes than EECBS-MFD, the columns "MFD" contain the percentages of MAPF instances where EECBS-MFD has lower runtimes than EECBS-GFD, and the row "Total" contains the percentage over all MAPF instances.

conflicts. Also, EECBS-MFD has a higher conflict resolution efficiency than EECBS-GFD with the bounded-suboptimality factors $w = \{1.05, 1.10\}$ on the `ost003d` graph, and otherwise has a competitive conflict resolution efficiency. Due to the hierarchical framework (see Section 4.5), MFD can determine a distributed flex smaller than GFD. Even so, EECBS-MFD and EECBS-GFD have similar conflict resolution efficiency for each bounded-suboptimality factor on each graph. Figure 4.12 shows the conflict resolution efficiencies versus the SOC differences of EECBS, EECBS-GFD, and EECBS-MFD. Both EECBS-GFD and EECBS-MFD tend to have higher conflict resolution efficiencies and larger SOC differences than EECBS, indicating that

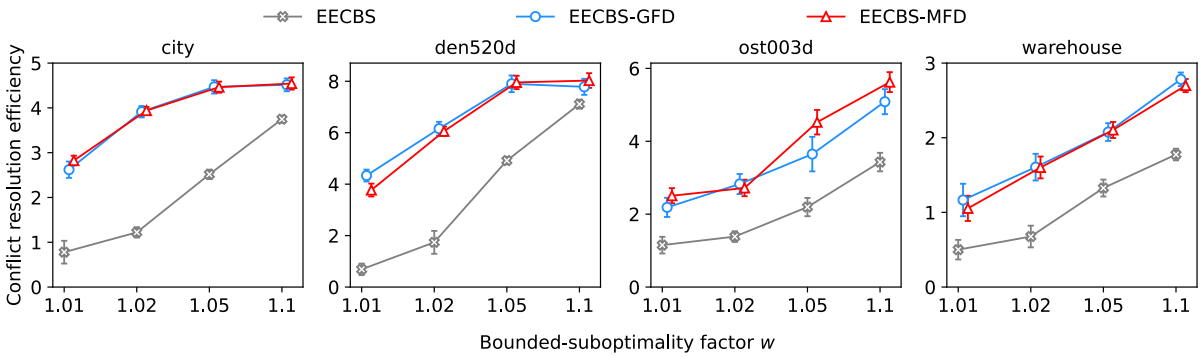


Figure 4.11: Average conflict resolution efficiencies of EECBS, EECBS-GFD, and EECBS-MFD for each bounded-suboptimality factor w over their respective MAPF instances that they successfully solve within the 120-second runtime limit on each graph. The vertical bars indicate the 95% confidence intervals.

	$w = 1.01$		$w = 1.02$		$w = 1.05$		$w = 1.10$	
	EECBS	MFD	EECBS	MFD	EECBS	MFD	EECBS	MFD
`city`	1	9	1	10	45	11	93	14
`den520d`	1	1	0	6	43	21	66	27
`ost003d`	25	11	35	35	54	2	76	1
`warehouse`	26	12	15	36	12	68	29	69
Total	53	33	51	87	154	102	264	111

Table 4.7: MAPF instance comparisons of the empirical suboptimalities of EECBS and EECBS-MFD over all MAPF instances that are successfully solved by both MAPF algorithms within the 120-second runtime limit on each graph. For each bounded-suboptimality factor w, the columns "EECBS" contain the number of MAPF instances where EECBS has a lower empirical suboptimality than EECBS-MFD, the columns "MFD" contain the number of MAPF instances where EECBS-MFD has a lower empirical suboptimality than EECBS, and the row "Total" contains the sum of the numbers of MAPF instances over each graph.

they achieve better conflict resolution efficiencies via worse solution qualities than EECBS. Also, EECBS-MFD solves more MAPF instances than EECBS-GFD, which results in more dots. This indicates that MFD is more effective than GFD in trading off solution quality for conflict resolution efficiency.

In terms of solution quality, Figure 4.13 shows the SOCs of the solutions found by EECBS, EECBS-GFD, and EECBS-MFD. The solutions found by EECBS-MFD typically have higher average SOCs than those found by EECBS and EECBS-GFD, especially for MAPF instances on large graphs, such as the `city` and `den520d` graphs. However, since the solutions found by these MAPF algorithms are guaranteed to be bounded-suboptimal, trading off solution quality for conflict resolution efficiency is considered reasonable.

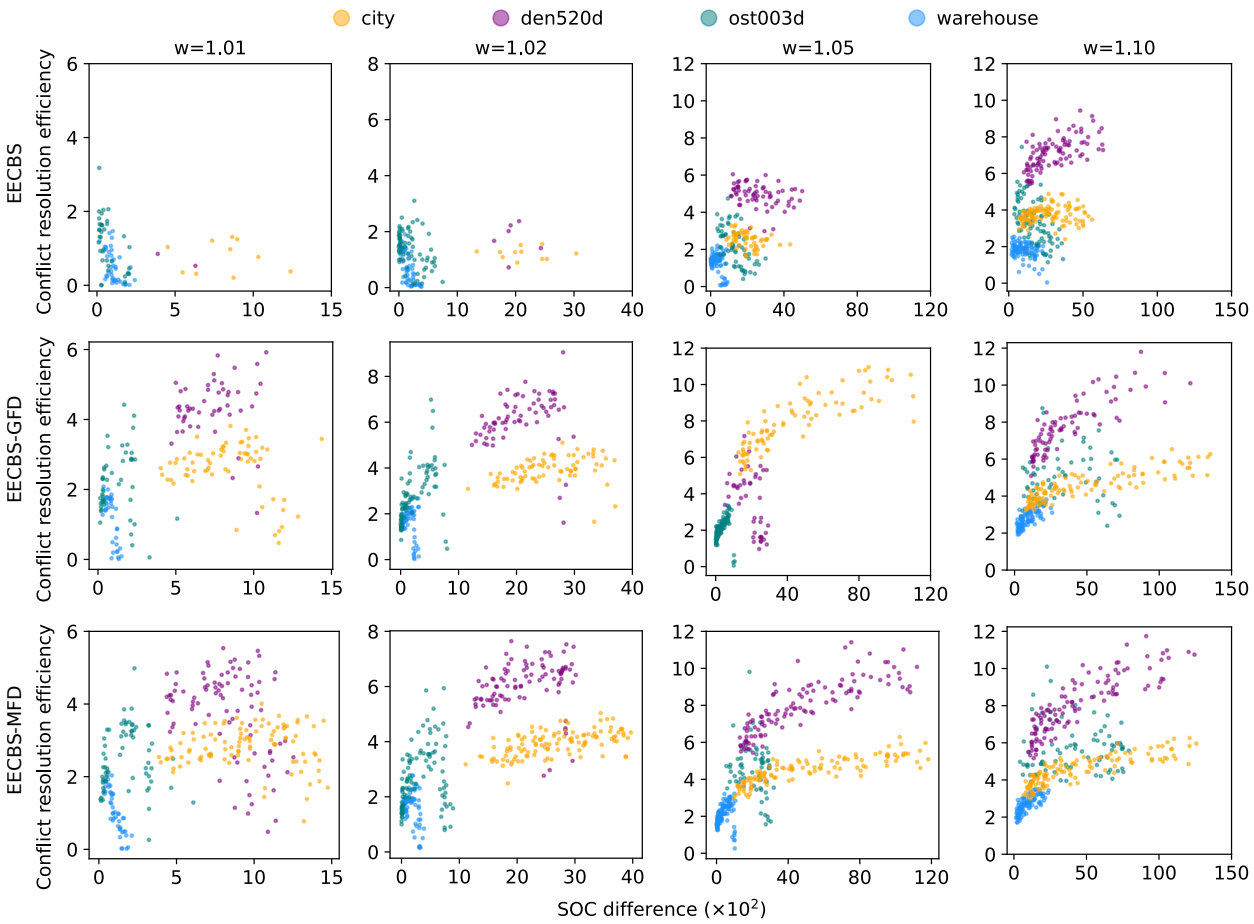


Figure 4.12: Conflict resolution efficiencies versus SOC differences of EECBS, EECBS-GFD, and EECBS-MFD for each bounded-suboptimality factor w over their respective sets of MAPF instances that are successfully solved by each MAPF algorithm within the 120-second runtime limit on each graph.

Figure 4.14 shows the average empirical suboptimalities of EECBS, EECBS-GFD, and EECBS-MFD, indicating that the solutions they find have the average empirical suboptimalities much smaller than the bounded-suboptimality factors. We also compare the empirical suboptimalities of EECBS-MFD versus EECBS and EECBS-MFD versus EECBS-GFD. For each pair of MAPF algorithms, we compare their empirical suboptimalities among the MAPF instances that are solved by both of them within the 120-second runtime limit. The top row of Figure 4.15 shows the empirical suboptimality of EECBS-MFD versus EECBS. Table 4.7 shows the number of MAPF instances where EECBS-MFD has a lower empirical suboptimality than EECBS and those where the other way around. Although EECBS-MFD uses flex to increase the

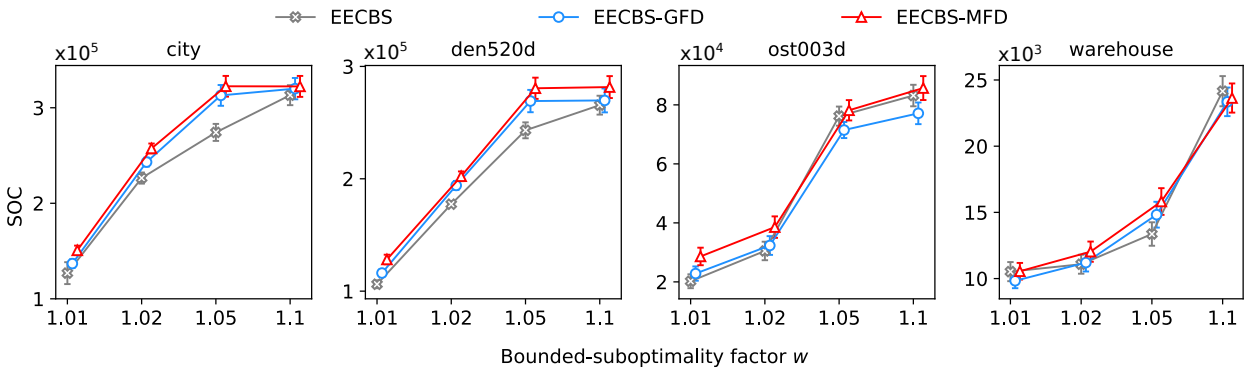


Figure 4.13: Average SOCs of EECBS, EECBS-GFD, and EECBS-MFD for each bounded-suboptimality factor w over their respective sets of MAPF instances that are successfully solved by each MAPF algorithm within the 120-second runtime limit on each graph. The vertical bars indicate the 95% confidence intervals.

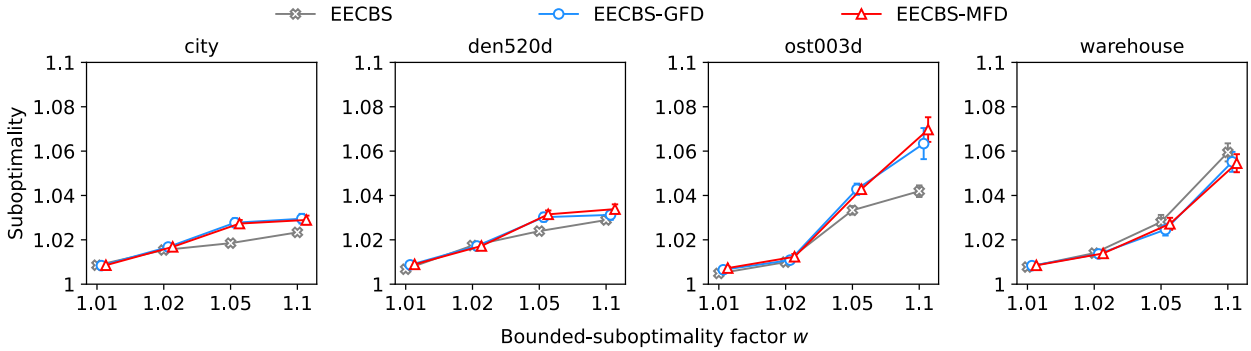


Figure 4.14: Average empirical suboptimalities of EECBS, EECBS-GFD, and EECBS-MFD for each bounded-suboptimality factor w over their respective sets of MAPF instances that are successfully solved by each MAPF algorithm within the 120-second runtime limit on each graph. The vertical bars indicate the 95% confidence intervals.

threshold when finding a path for an agent, it still has a chance to solve MAPF instances with lower empirical suboptimality than EECBS, especially when solving MAPF instances with the bounded-suboptimality factors $w = \{1.02, 1.05, 1.10\}$ on the `warehouse` graph. The bottom row of Figure 4.15 shows the empirical suboptimality of EECBS-MFD versus EECBS-GFD. Table 4.8 shows the number of MAPF instances where EECBS-MFD has a lower empirical suboptimality than EECBS-GFD and those where the other way around. Since MFD reduces the distributed flex during the search, there are more MAPF instances where EECBS-MFD has lower empirical suboptimalities than EECBS-GFD than the other way around.

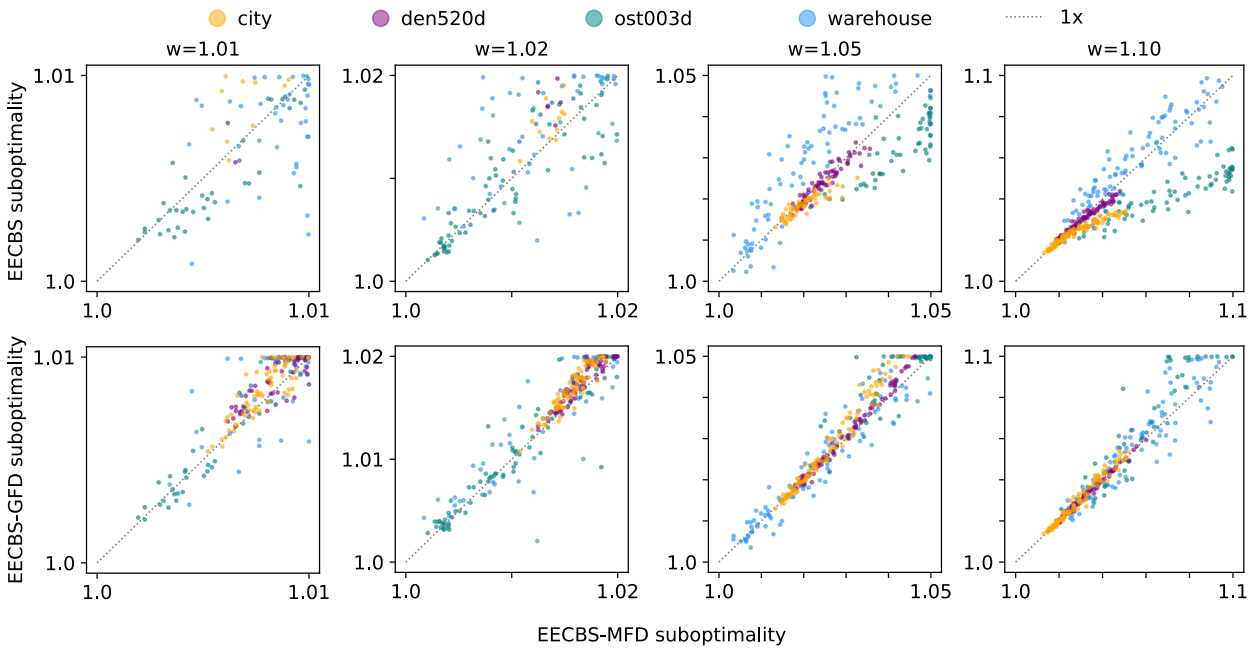


Figure 4.15: Empirical suboptimalities of EECBS (or, respectively, EECBS-GFD) versus empirical suboptimalities of EECBS-MFD for each bounded-suboptimality factor w over all MAPF instances that are successfully solved by both MAPF algorithms within the 120-second runtime limit on each graph. The x- and y-coordinates of each dot represent the empirical suboptimalities of EECBS-MFD and EECBS (or, respectively, EECBS-GFD), respectively. Dots in yellow, purple, green, and blue represent the MAPF instances on `city`, `den520d`, `ost003d`, and `warehouse` graphs, respectively.

	$w = 1.01$		$w = 1.02$		$w = 1.05$		$w = 1.10$	
	GFD	MFD	GFD	MFD	GFD	MFD	GFD	MFD
`city`	14	51	6	70	32	73	41	70
`den520d`	10	38	5	62	19	65	41	38
`ost003d`	20	25	26	50	8	37	17	39
`warehouse`	12	23	18	36	50	49	33	63
Total	56	137	55	218	109	224	132	210

Table 4.8: MAPF instance comparisons of the empirical suboptimalities of EECBS-GFD and EECBS-MFD over all MAPF instances that are successfully solved by both MAPF algorithms within the 120-second runtime limit on each graph. For each bounded-suboptimality factor w, the columns "GFD" contain the number of MAPF instances where EECBS-GFD has a lower empirical suboptimality than EECBS-MFD, the columns "MFD" contain the number of MAPF instances where EECBS-MFD has a lower empirical suboptimality than EECBS-GFD, and the row "Total" contains the sum of the numbers of MAPF instances over each graph.

4.7.2 Empirical Insights

As shown in the top row of Figure 4.16, by distributing flex fractionally, EECBS-MFD overcomes the high-level limitation (i.e., switching among CT nodes on different branches of CT rather than resolving conflicts

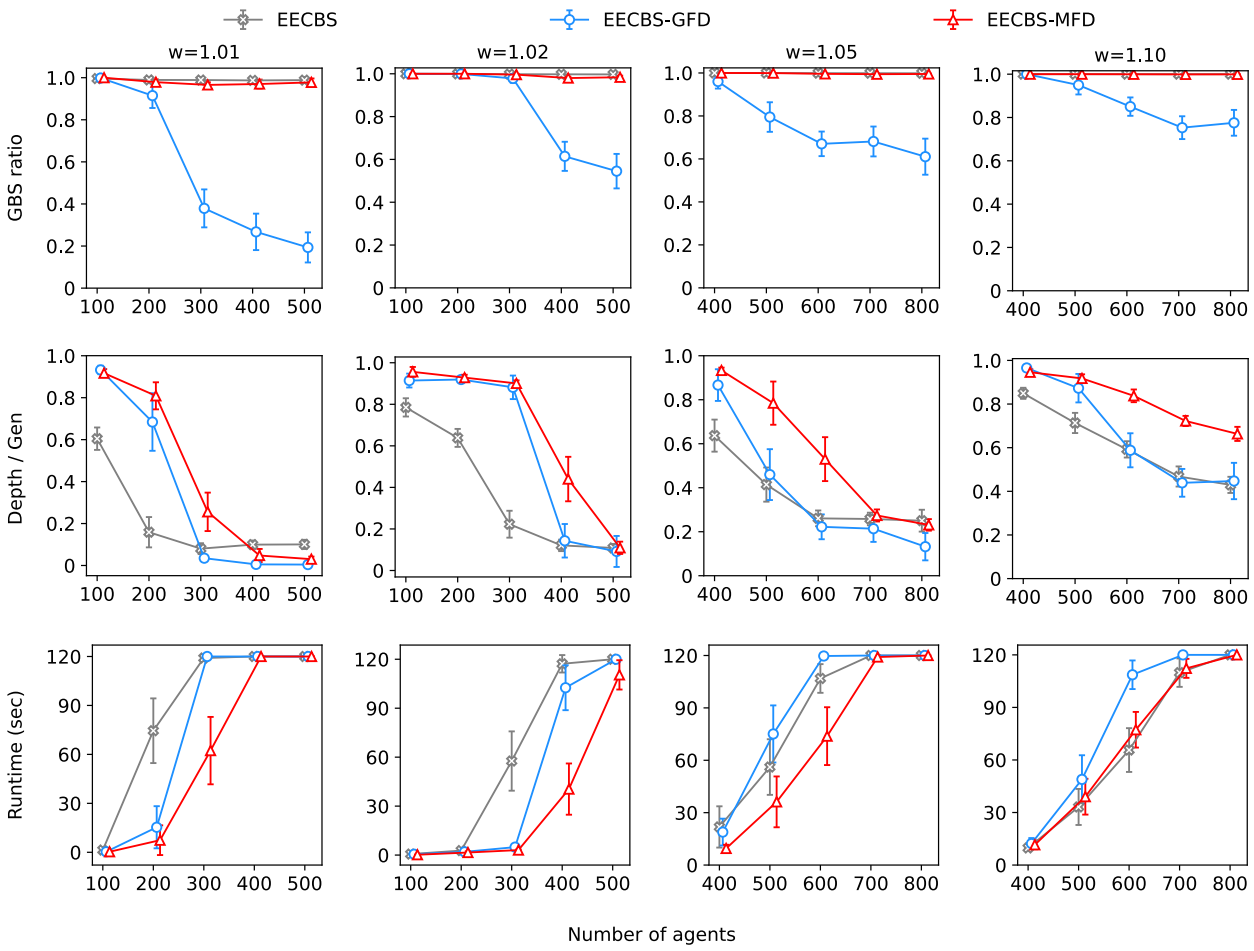


Figure 4.16: Average GBS ratios (see Equation 4.1), average resultant CT depth ratios (see Equation 4.2), and average runtimes (in seconds) of EECBS, EECBS-GFD, and EECBS-MFD for each bounded-suboptimality factor w over all MAPF instances for each number of agents on the `ost003d` graph. For MAPF instances that are not solved by a MAPF algorithm within the 120-second runtime limit, we set the runtime to 120 seconds. The vertical bars indicate the 95% confidence intervals.

along a branch), resulting in its GBS ratios being higher than EECBS-GFD and competitive with EECBS. As shown in the middle row of Figure 4.16, EECBS-MFD has higher resultant CT depth ratios than EECBS and EECBS-GFD, indicating that EECBS-MFD expands CT nodes more in a depth-first manner than EECBS and EECBS-GFD. Thus, as shown in the bottom row of Figure 4.16, EECBS-MFD has smaller average runtimes than EECBS and EECBS-GFD, indicating that EECBS-MFD is more efficient than EECBS and EECBS-GFD in finding bounded-suboptimal solutions.

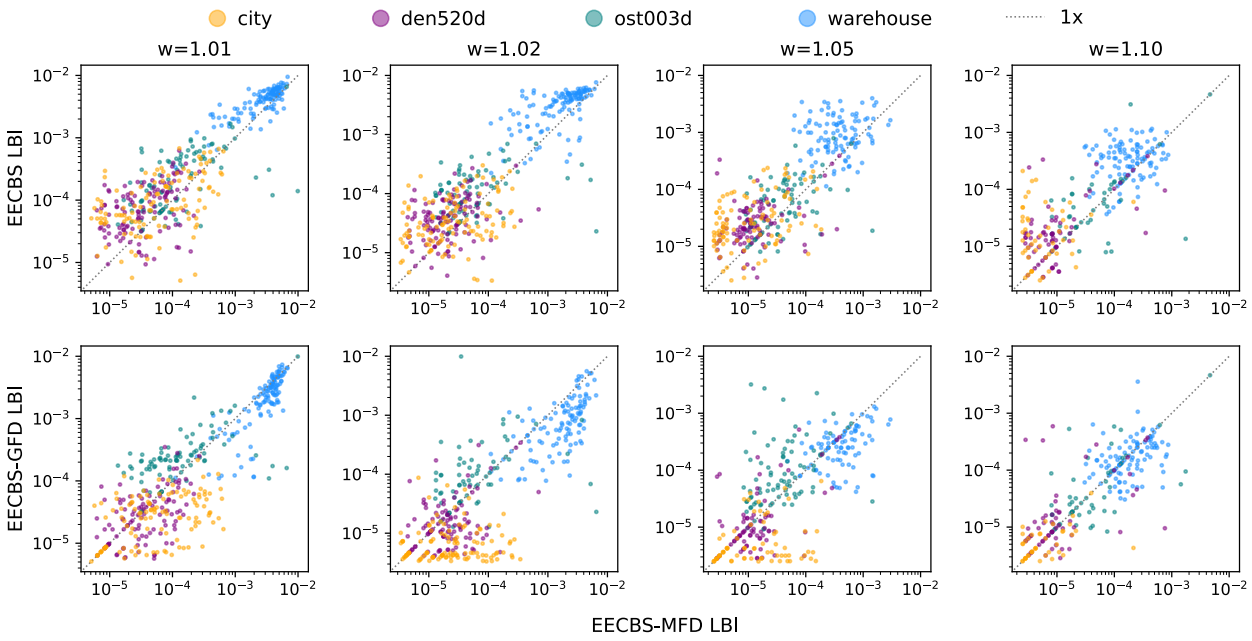


Figure 4.17: LBIs of EECBS (or, respectively, EECBS-GFD) versus LBIs of EECBS-MFD for each bounded-suboptimality factor w over all MAPF instances. The x- and y-coordinates of each dot represent the LBIs of EECBS-MFD and EECBS (or, respectively, EECBS-GFD), respectively. All the x- and y-coordinates are on logarithmic scales. Dots in yellow, purple, green, and blue represent the MAPF instances in the `city`, `den520d`, `ost003d`, and `warehouse` graphs, respectively.

	$w = 1.01$		$w = 1.02$		$w = 1.05$		$w = 1.10$	
	EECBS	MFD	EECBS	MFD	EECBS	MFD	EECBS	MFD
`city`	0.65	0.34	0.56	0.37	0.76	0.18	0.66	0.13
`den520d`	0.78	0.20	0.71	0.20	0.76	0.18	0.59	0.20
`ost003d`	0.74	0.15	0.71	0.10	0.56	0.24	0.32	0.20
`warehouse`	0.86	0.14	0.86	0.14	0.78	0.18	0.69	0.22
Total	0.76	0.21	0.71	0.20	0.72	0.20	0.57	0.19

Table 4.9: MAPF instance comparisons of LBIs of EECBS and EECBS-MFD over all MAPF instances on each graph. For each bounded-suboptimality factor w, the columns "EECBS" contain the percentages of MAPF instances where EECBS has higher LBIs than EECBS-MFD, the columns "MFD" contain the percentages of MAPF instances where EECBS-MFD has higher LBIs than EECBS, and the row "Total" contains the percentage over all MAPF instances.

Figure 4.17 and Tables 4.9 and 4.10 show the LBIs of EECBS-MFD compared to those of EECBS and EECBS-GFD. As shown in Table 4.9, since EECBS can only increase the lower bound on the cost of a minimum-cost path in order to increase the threshold, there are more MAPF instances where EECBS has higher LBIs than EECBS-MFD. As shown in Table 4.10, by reducing the distributed flex, there are more MAPF instances where EECBS-MFD has higher LBIs than EECBS-GFD for the bounded-suboptimality

	$w = 1.01$		$w = 1.02$		$w = 1.05$		$w = 1.10$	
	GFD	MFD	GFD	MFD	GFD	MFD	GFD	MFD
`city`	0.20	0.64	0.11	0.74	0.20	0.57	0.29	0.25
`den520d`	0.30	0.54	0.23	0.60	0.25	0.54	0.36	0.34
`ost003d`	0.62	0.15	0.38	0.21	0.66	0.18	0.35	0.21
`warehouse`	0.19	0.75	0.12	0.79	0.19	0.58	0.38	0.46
Total	0.33	0.52	0.21	0.59	0.33	0.46	0.34	0.31

Table 4.10: MAPF instance comparisons of LBIs of EECBS-GFD and EECBS-MFD over all MAPF instances. For each bounded-suboptimality factor w, the columns "GFD" contain the percentages of MAPF instances where EECBS-GFD has higher LBIs than EECBS-MFD, the columns "MFD" contain the percentages of MAPF instances where EECBS-MFD has higher LBIs than EECBS-GFD, and the row "Total" contains the percentage over all MAPF instances.

$w = 1.01$	k	100	200	300	400	500
	EECBS	0.86	0.25	0.10	**0.16**	**0.16**
	GFD	**1.00**	0.72	0.05	0.01	<0.01
	MFD	**1.00**	**0.91**	**0.44**	0.05	0.06

$w = 1.05$	k	400	500	600	700	800
	EECBS	0.96	0.81	0.54	**0.46**	**0.41**
	GFD	0.93	0.64	0.30	0.30	0.23
	MFD	**1.00**	**0.95**	**0.66**	0.42	0.36

$w = 1.02$	k	100	200	300	400	500
	EECBS	0.99	0.87	0.50	0.18	0.18
	GFD	**1.00**	**1.00**	0.97	0.18	0.13
	MFD	**1.00**	**1.00**	**0.99**	**0.55**	**0.22**

$w = 1.10$	k	400	500	600	700	800
	EECBS	**1.00**	0.95	0.90	0.78	0.70
	GFD	**1.00**	0.96	0.75	0.68	0.66
	MFD	**1.00**	**1.00**	**0.99**	**0.98**	0.94

Table 4.11: Average progress of EECBS with different Flex Distribution mechanisms but each bounded-suboptimality factor w over all MAPF instances for each number of agents k on the `ost003d` graph. Rows "GFD", "FFD", "RFD", "CFD", "DFD", and "MFD" contain the average progress of EECBS-GFD, EECBS-FFD, EECBS-RFD, EECBS-CFD, EECBS-DFD, and EECBS-MFD, respectively. Numbers in bold indicate the highest progress over all MAPF instances for each number of agents k and each bounded-suboptimality factor w.

factors $w = \{1.01, 1.02, 1.05\}$, which indicates that EECBS-MFD overcomes the low-level limitation (i.e., worse LB improvements) from EECBS-GFD. As the bounded-suboptimality factor w increases to 1.10, since it provides thresholds that are large enough to find a path for an agent, both EECBS-MFD and EECBS-GFD reach similar LBIs.

We define the *progress* of EECBS (with or without Flex Distribution) as the ratio of the resultant depth of the CT and the number of expanded CT nodes. The deeper the CT node, the fewer conflicts it is expected to have. Thus, the higher the progress, the more "focused" EECBS (with or without Flex Distribution) is in resolving conflicts. In the extreme, the progress approaches 1 if EECBS (with or without Flex Distribution) expands CT nodes only along the branch of the CT from the root CT node to the resultant CT node.

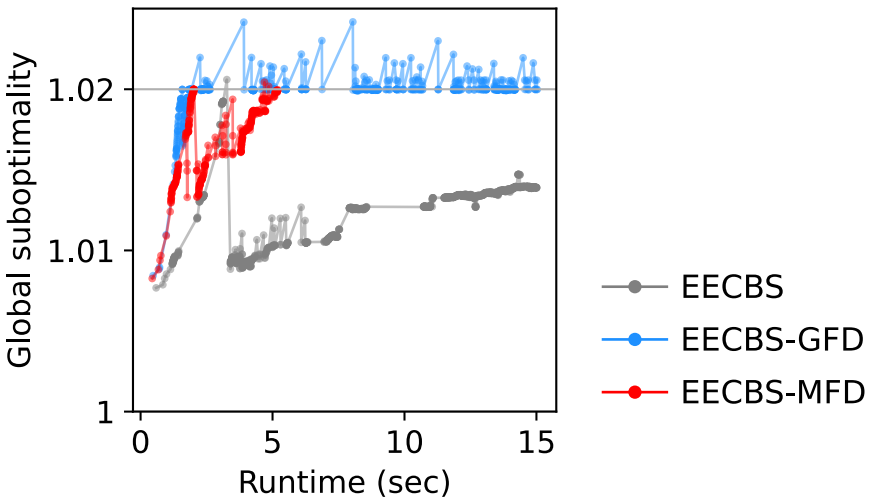


Figure 4.18: Global suboptimalities at the time of generation of each CT node generated by EECBS, EECBS-GFD, and EECBS-MFD as a function of the time it was generated. The x- and y-coordinates of each dot represent the time at which a CT node is generated and its global suboptimality at that time, respectively. Dots in gray, blue, and red represent CT nodes generated by EECBS, EECBS-GFD, and EECBS-MFD, respectively. The number of generated CT nodes for EECBS, EECBS-GFD, and EECBS-MFD is 1312, 608, and 350, respectively.

Table 4.11 shows the progress of EECBS, EECBS-GFD, and EECBS-MFD, where EECBS-MFD has a higher progress than EECBS-GFD. When the number of agents is low, EECBS-MFD has higher progress than EECBS and EECBS-GFD. As the number of agents increases, the progress of EECBS-GFD decreases due to the greedy mechanism for Flex Distribution, whereas the progress of EECBS-MFD remains high. For the bounded-suboptimal factors and the numbers of agents where EECBS has the highest progress, the corresponding success rates are similar between EECBS and EECBS-MFD (see Figure 4.6).

In general, EECBS-MFD overcomes the high-level and the low-level limitations of EECBS-GFD and thus improves the efficiency of EECBS.

4.7.3 Case Study

We let EECBS, EECBS-GFD, and EECBS-MFD, each with the bounded-suboptimality factor $w = 1.02$ and a 15-second runtime limit, solve a MAPF instance with 400 agents on the `ost003d` graph. EECBS and EECBS-GFD reach the 15-second runtime limit when they try to solve this MAPF instance. Figure 4.18 shows the

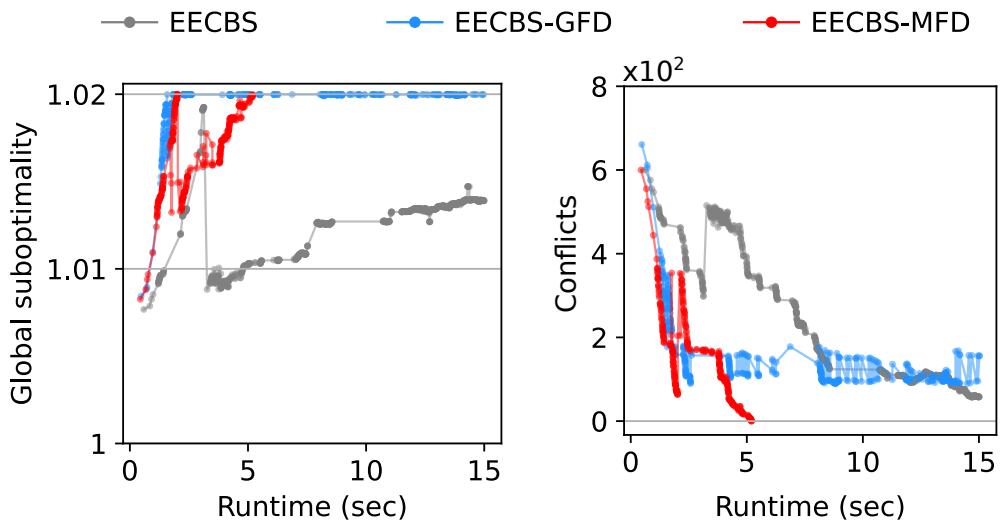


Figure 4.19: Global suboptimalities and number of conflicts at the time of expansion of each CT node expanded by EECBS, EECBS-GFD, and EECBS-MFD as functions of the time it was expanded. The x-coordinate represents the time at which a CT node is expanded, and the y-coordinates represent the global suboptimality and the number of conflicts at the time each CT node is expanded. The number of expanded CT nodes for EECBS, EECBS-GFD, and EECBS-MFD is 843, 395, and 305, respectively.

global suboptimality $C(N)/LB$ (see Definition 3.4) at the time of generation of each CT node N, where the LB-value is recorded when CT node N is generated. Since EECBS requires an individually bounded-suboptimal path for each agent in a CT node, it tends to generate CT nodes with low SOCs, resulting in low global suboptimalities. On the other hand, since EECBS-GFD determines the distributed flex as the maximum allowed flex, it tends to generate CT nodes with high global suboptimalities. Moreover, many CT nodes generated by EECBS-GFD are not globally bounded-suboptimal during their generation. Thus, EECBS-GFD will not expand them unless LB increases during the search. Compared to EECBS-GFD, EECBS-MFD typically distributes less flex when generating a CT node, which strikes a balance between the amount of distributed flex and the number of CT nodes that are globally bounded-suboptimal during their generation. Also, EECBS-MFD finds a solution with a runtime of about 5 seconds, so there is no red dot afterwards in Figure 4.18. Figure 4.19 shows the global suboptimality and the number of conflicts of each expanded CT node as functions of the time it was expanded. The CT nodes expanded by EECBS-MFD

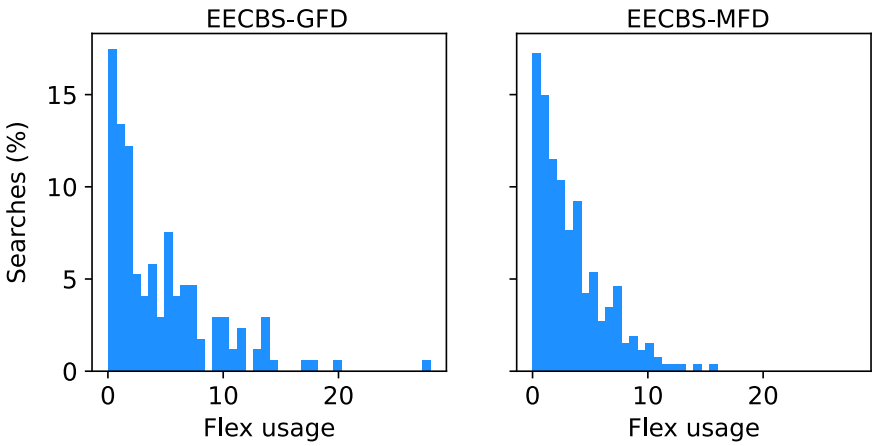


Figure 4.20: Percentage of low-level searches of EECBS-GFD and EECBS-MFD as a function of the flex usage. The x-coordinate indicates the flex usage of a low-level search, and the y-coordinate indicates the percentage of the low-level searches.

have similar numbers of conflicts but smaller global suboptimalities than those expanded by EECBS-GFD, allowing EECBS-MFD to find a solution within the 15-second runtime limit.

When the low-level Focal Search finds a path with the cost $c_i(N)$ for an agent a_i and a lower bound $lb_i(N)$ on the cost of a minimum-cost path in a CT node N and terminates, if the flex of this path is negative (i.e., $w \cdot lb_i(N) - c_i(N) < 0$), then we say that the *flex usage* is

$$\text{flex usage} = c_i(N) - w \cdot lb_i(N). \tag{4.22}$$

Figure 4.20 shows the percentage of low-level searches of EECBS-GFD and EECBS-MFD as a function of the flex usage when it finds a path for an agent, ignoring the searches when they find paths with $w \cdot lb_i(N) - c_i(N) \geq 0$. Compared to EECBS-GFD, EECBS-MFD has a smaller percentage of low-level searches with high flex usage. For example, none of the low-level searches of EECBS-MFD has flex usage exceeding 20.

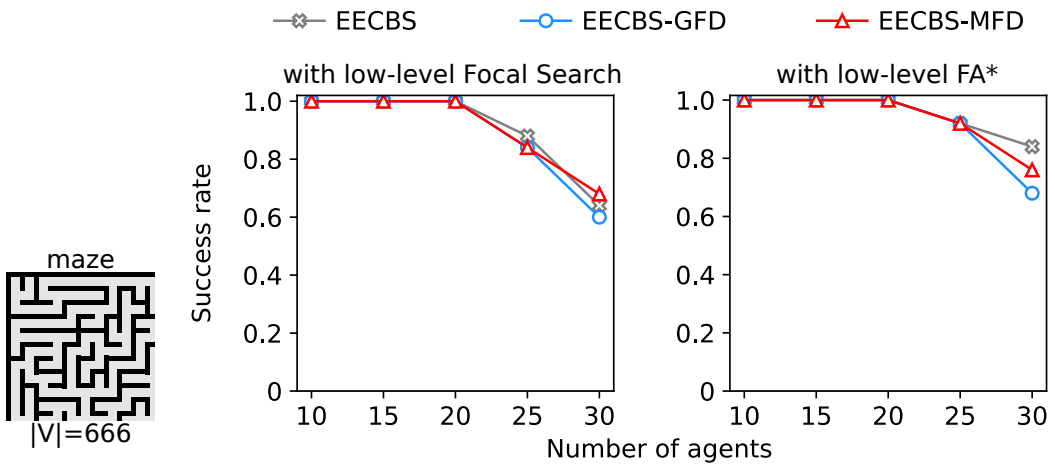


Figure 4.21: Success rates of EECBS, EECBS-GFD, and EECBS-MFD, with the low-level Focal Search and the low-level FA*, respectively, for the 120-second runtime limit and the bounded-suboptimality factor $w = 1.05$ over all MAPF instances for each number of agents on the `maze` graph. $|V|$ indicates the number of vertices of the `maze` graph.

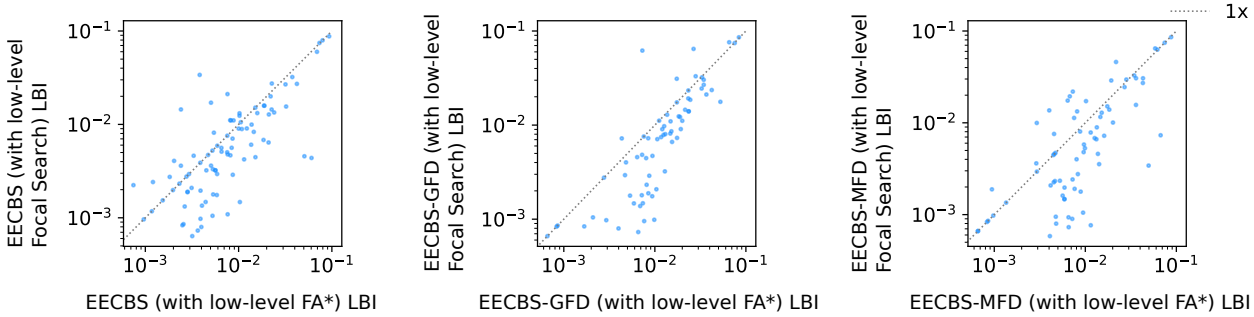


Figure 4.22: LBIs of EECBS (or, respectively, EECBS-GFD and EECBS-MFD) with the low-level Focal Search, versus LBIs of EECBS (or, respectively, EECBS-GFD and EECBS-MFD) with the low-level FA* for the 120-second runtime limit and the bounded-suboptimality factor $w = 1.05$ over all MAPF instances on the `maze` graph. The x- and y-coordinates of each dot represent the LBIs of EECBS-MFD (or, respectively, EECBS-GFD and EECBS-MFD) with the low-level Focal Search and EECBS-MFD (or, respectively, EECBS-GFD and EECBS-MFD) with the low-level FA*, respectively. All the x- and y-coordinates are on logarithmic scales.

4.7.4 Evaluation of the Low-Level FA*

To evaluate the low-level FA*, we test EECBS, EECBS-GFD, and EECBS-MFD on 25 MAPF instances for the 120-second runtime limit, the bounded-suboptimality factor $w = 1.05$, and each number of agents $k = \{10, 15, 20, 25, 30\}$ on the `maze` graph (*maze-32-32-2*) of size 32×32. As shown in Figure 4.21, all success rates improve when the low-level Focal Search is replaced with the low-level FA*. With the low-level FA*, the success rate of EECBS is higher than that of EECBS-MFD, which, in turn, is higher than that

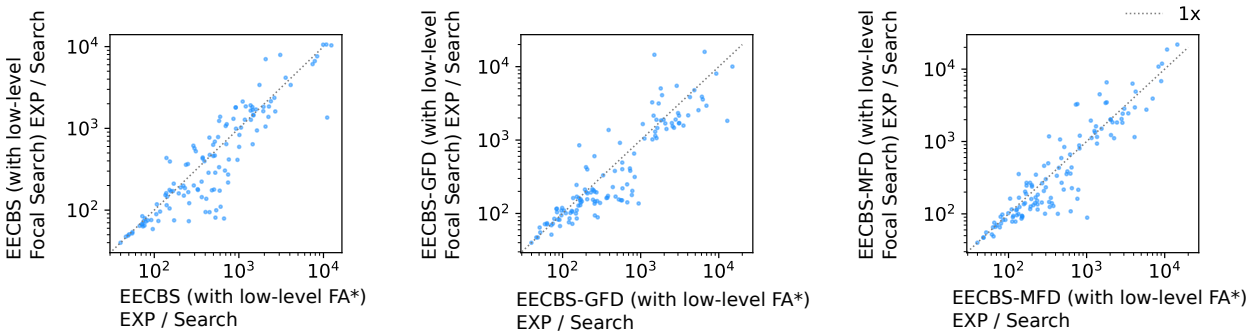


Figure 4.23: Average numbers of v-t node expansions per search (EXP/Search) of EECBS (or, respectively, EECBS-GFD and EECBS-MFD) with the low-level Focal Search, versus average numbers of v-t node expansions per search of EECBS (or, respectively, EECBS-GFD and EECBS-MFD) with the low-level FA* for the 120-second runtime limit and the bounded-suboptimality factor $w = 1.05$ over all MAPF instances on the `maze` graph. The x- and y-coordinates of each dot represent the LBIs of EECBS-MFD (or, respectively, EECBS-GFD and EECBS-MFD) with the low-level Focal Search and EECBS-MFD (or, respectively, EECBS-GFD and EECBS-MFD) with the low-level FA*, respectively. All the x- and y-coordinates are on logarithmic scales.

of EECBS-GFD. As shown in Figure 4.22, EECBS, EECBS-GFD, and EECBS-MFD reach higher LBIs and thus expand CT nodes more in a depth-first manner when they use the low-level FA* than when they use the low-level Focal Search. As shown in Figure 4.23, EECBS, EECBS-GFD, and EECBS-MFD reach higher average numbers of v-t node expansions per search when they use the low-level FA* than when they use the low-level Focal Search since the low-level FA* expands v-t nodes from OPEN$_L$ after it finds a path, while the low-level Focal Search terminates once it finds a path. That is, the low-level FA* trades off expanding more v-t nodes (from OPEN$_L$) for a higher LBI value.

We also test EECBS, EECBS-GFD, and EECBS-MFD with the low-level FA* on 25 MAPF instances for the 120-second runtime limit, the bounded-suboptimality factor $w = 1.05$, and each number of agents $k = \{1200, 1400, 1600, 1800, 2000\}$ on the `city` graph. As shown in Figure 4.24, all success rates decrease when the low-level Focal Search is replaced with the low-level FA*. As shown in Figure 4.25, although EECBS, EECBS-GFD, and EECBS-MFD reach higher LBIs when they use the low-level FA*, their LBI values on the `city` graph are approximately between 10^{-5} and 10^{-3}, which are lower than their LBI values when using the low-level FA* on the `maze` graph, which are approximately between 10^{-3} and 10^{-1} (see

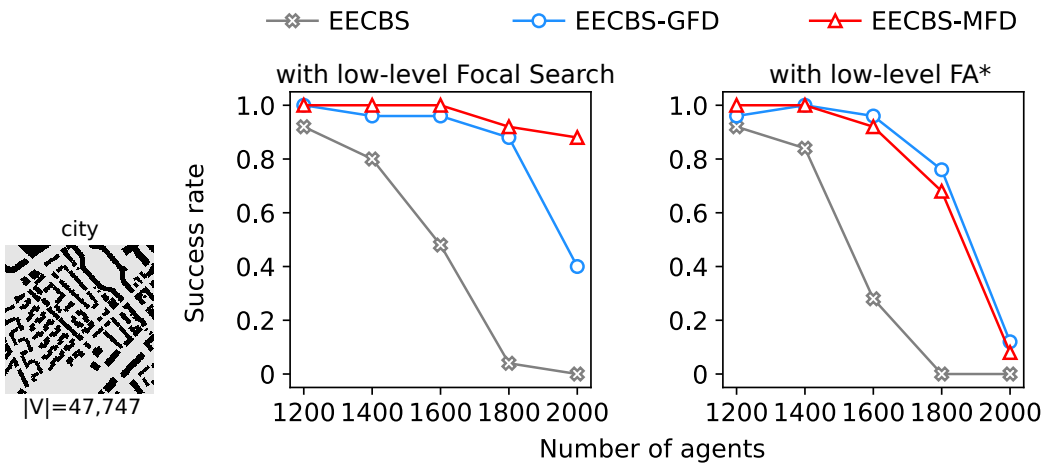


Figure 4.24: Success rates of EECBS, EECBS-GFD, and EECBS-MFD, with the low-level Focal Search and the low-level FA*, respectively, for the 120-second runtime limit and the bounded-suboptimality factor $w = 1.05$ over all MAPF instances for each number of agents on the `maze` graph. $|V|$ indicates the number of vertices of the `maze` graph.

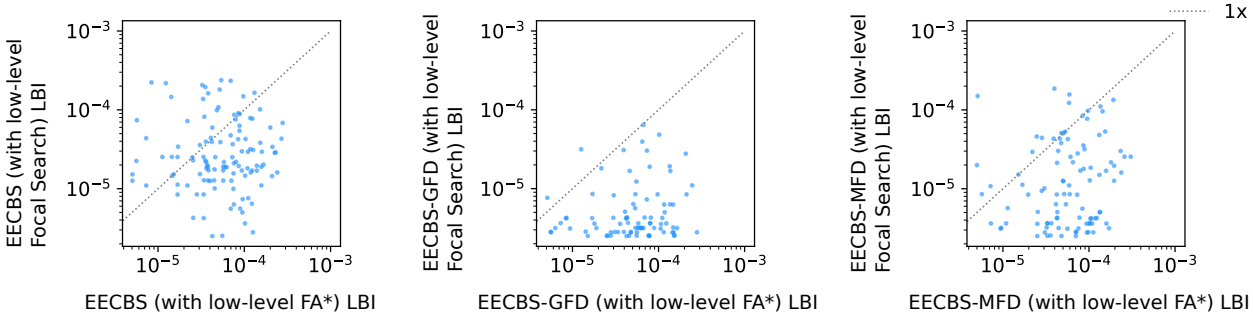


Figure 4.25: LBIs of EECBS (or, respectively, EECBS-GFD and EECBS-MFD) with the low-level Focal Search, versus LBIs of EECBS (or, respectively, EECBS-GFD and EECBS-MFD) with the low-level FA* for the 120-second runtime limit and the bounded-suboptimality factor $w = 1.05$ over all MAPF instances on the `city` graph. The x- and y-coordinates of each dot represent the LBIs of EECBS-MFD (or, respectively, EECBS-GFD and EECBS-MFD) with the low-level Focal Search and EECBS-MFD (or, respectively, EECBS-GFD and EECBS-MFD) with the low-level FA*, respectively. All the x- and y-coordinates are on logarithmic scales.

Figure 4.22). As shown in Figure 4.26, EECBS, EECBS-GFD, and EECBS-MFD have much higher average numbers of v-t node expansions per search when they use the low-level FA* than when they use the low-level Focal Search. In general, for large-scale MAPF instances, although the low-level FA* improves the LBIs of EECBS, EECBS-GFD, and EECBS-MFD, it also expands many more v-t nodes than the low-level Focal Search, which slows down the search and thus decreases the success rates for the 120-second runtime limit.

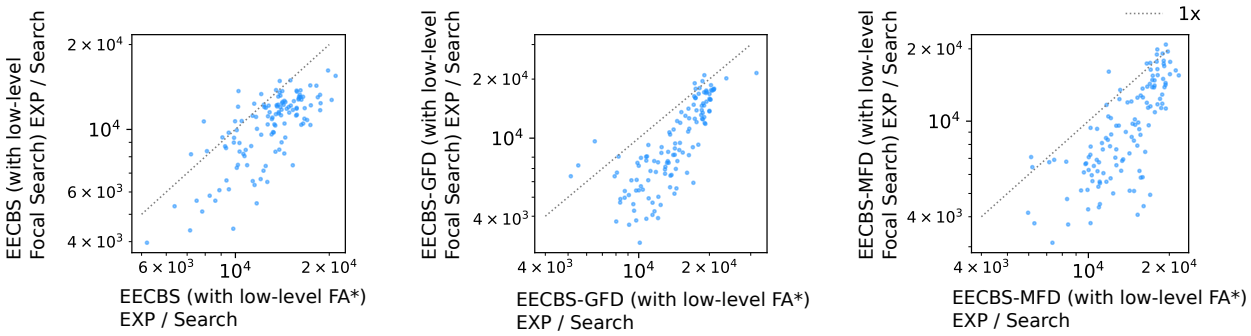


Figure 4.26: Average numbers of v-t node expansions per search (EXP/Search) of EECBS (or, respectively, EECBS-GFD and EECBS-MFD) with the low-level Focal Search, versus average numbers of v-t node expansions per search of EECBS (or, respectively, EECBS-GFD and EECBS-MFD) with the low-level FA* for the 120-second runtime limit and the bounded-suboptimality factor $w = 1.05$ over all MAPF instances on the `city` graph. The x- and y-coordinates of each dot represent the LBIs of EECBS-MFD (or, respectively, EECBS-GFD and EECBS-MFD) with the low-level Focal Search and EECBS-MFD (or, respectively, EECBS-GFD and EECBS-MFD) with the low-level FA*, respectively. All the x- and y-coordinates are on logarithmic scales.

4.8 Summary

EECBS with Greedy Flex Distribution (GFD) runs faster than EECBS, while it is still guaranteed to find a bounded-suboptimal solution to a MAPF instance if a solution exists. However, GFD imposes two limitations: (1) EECBS with GFD may switch among CT nodes on different branches of CT that are globally bounded-suboptimal, rather than resolving conflicts along the branch, and (2) EECBS with GFD may have a worse LB improvement than EECBS. To overcome these limitations, we proposed *Conflict-Based Flex Distribution* (CFD), which determines the distributed flex in proportion to the number of collisions. We also proposed *Delay-Based Flex Distribution* (DFD), which determines the distributed flex by estimating the increase in the cost (i.e., delays) to find a new path that satisfies the constraints. Furthermore, we proposed *Mixed-Strategy Flex Distribution* (MFD), which combines DFD and CFD in a hierarchical framework. We proved that EECBS with MFD, like EECBS and EECBS with GFD, is guaranteed to find a bounded-suboptimal solution to a MAPF instance if a solution exists (i.e., bounded-suboptimal and solution-complete). Our experiments showed that EECBS with different Flex Distribution mechanisms, especially MFD, solves MAPF instances more quickly than EECBS and EECBS with GFD. Compared to

EECBS and EECBS with GFD, EECBS with MFD solves more MAPF instances within the 120-second runtime limit and achieves competitive average empirical suboptimalities. For the 120-second runtime limit and the bounded-suboptimality factor $w = 1.01$, using MFD results in a more than threefold increase in the success rate of EECBS over all MAPF instances on all graphs, raising it from 0.18 to 0.598. Also, the success rate of EECBS-MFD (0.598) is higher than that of EECBS-GFD (0.396). Additionally, we proposed the low-level Focal-A* (FA*) for solving congested MAPF instances. When finding a path for an agent that satisfies constraints, the low-level FA* uses the low-level Focal Search to find a path and then uses the low-level A* to improve the lower bound on the cost of a minimum-cost path that satisfies the constraints. We showed that using the low-level FA* can improve the efficiency of EECBS and EECBS-MFD in solving congested MAPF instances.

Algorithm 4.1 Low-Level Focal-A* (FA*) with Flex Distribution

procedure FOCAL-A*SEARCHWITHFLEX(s_i, l_i, $\Psi_i(N)$, $\dot{P}_i(N)$, $lb_i(\hat{N})$)
  ▷ $lb_i(\hat{N}) = 0$ *in the root CT node and* $\dot{P}_i(N) = \{p_j(N) \mid j \in [k] \setminus \{i\}\}$. ◁
  $T_i(N) \leftarrow$ Compute the time horizon from constraints $\Psi_i(N)$and paths $\dot{P}_i(N)$. ▷ *See Section 2.4.2.*
  $\Delta_i(N) \leftarrow$ Compute the distributed flex from paths $\dot{P}_i(N)$
  Determine the priority function $\tilde{f}_i(n)$ of v-t node n ▷ $\tilde{f}_i(n) = f_i(n)$ *unless mentioned otherwise.*
  $isPathFound \leftarrow$ false; $p_i \leftarrow$ null; $c_i \leftarrow$ null; $lb_i \leftarrow$ null
  Generate the root v-t node n_0 with state representation $(s_i, 0)$
  $g_i(n_0) \leftarrow 0$; $h_i(n_0) \leftarrow h_i(s_i)$; $f_i(n_0) \leftarrow g_i(n_0) + h_i(n_0)$; $\tilde{h}_i(n_0) \leftarrow \tilde{h}_i(s_i)$; $\tilde{f}_i(n_0) \leftarrow g_i(n_0) + \tilde{h}_i(n_0)$; $x_i(n_0) \leftarrow 0$
  $f_{\min,i}(N) \leftarrow f_i(n_0)$
  $\tau_i(N) \leftarrow w \cdot \max\{lb_i(\hat{N}), f_{\min,i}(N)\} + \Delta_i(N)$
  OPEN$_L$,FOCAL$_L$, and CLOSED$_L \leftarrow$ empty list
  INSERTNODE(n_0, $\tau_i(N)$, OPEN$_L$, FOCAL$_L$, $isPathFound$)
  while OPEN$_L$ **not** empty **do**
    $\hat{n} \leftarrow$ v-t node with the minimum f_i-value in OPEN$_L$
    if $f_{\min,i}(N) < f_i(\hat{n})$ **then**
      $f_{\min,i}(N) \leftarrow f_i(\hat{n})$
      if not $isPathFound$ **then**
        $\tau_{new} \leftarrow w \cdot \max\{lb_i(\hat{N}), f_{\min,i}(N)\} + \Delta_i(N)$
        UPDATEFOCAL($\tau_i(N)$, τ_{new}, OPEN$_L$, FOCAL$_L$)
        $\tau_i(N) \leftarrow \tau_{new}$
    if $isPathFound$ **then**
      if $c_i \leq f_i(\hat{n})$ **then return** (p_i, c_i, c_i) ▷ *Path* p_i *is the minimum-cost path.*
      Remove v-t node $\hat{n}$ from OPEN$_L$
    else
      $\hat{n} \leftarrow$ v-t node with the minimum x_i-value in FOCAL$_L$ ▷ *Overwrite v-t node* $\hat{n}$.
      Remove v-t node $\hat{n}$ from OPEN$_L$ and FOCAL$_L$
    Insert $\hat{n}$ into CLOSED$_L$
    if ISTARGETREACHED($\hat{n}$, l_i, $\Psi_i(N)$) **then**
      $lb_i \leftarrow \max\{lb_i(\hat{N}), f_{\min,i}(N)\}$
      if $isPathFound$ **then return** (p_i, c_i, lb_i)
      $p_i \leftarrow$ Extract the path by back-tracking v-t node $\hat{n}$; $c_i \leftarrow$ cost of path p_i
      $isPathFound \leftarrow$ true
      if $lb_i = c_i$ **or** $\Psi_i(N) = \emptyset$ **then return** (p_i, c_i, lb_i) ▷ *Path* p_i *is the minimum-cost path.*
      continue
    neighbors $\leftarrow$ FINDNEIGHBORS($\hat{n}$, $T_i(N)$) ▷ *Find the neighboring states.*
    for $(v, t) \in$ *neighbors* **do**
      if (v, t) **not** satisfies constraints $\Psi_i(N)$ **then continue**
      $n \leftarrow$ GENERATECHILDNODE($\hat{n}$, v, t, $\dot{P}_i(N)$)
      $\bar{n} \leftarrow$ FINDNODE(n, $T_i(N)$, CLOSED$_L$, OPEN$_L$)
      if $\bar{n}$ is null **then**
        INSERTNODE(n, $\tau_i(N)$, OPEN$_L$, FOCAL$_L$, $isPathFound$)
        continue
      if ISDOMINANT(n, $\bar{n}$) **then**
        UPDATEPRIORITY(n, $\bar{n}$, $\tau_i(N)$, CLOSED$_L$, OPEN$_L$, FOCAL$_L$, $isPathFound$)
  return No path

Algorithm 4.2 Tool Functions for the Low-Level Focal-A* Search

procedure InsertNode(n, $\tau_i(N)$, OPEN$_L$, FOCAL$_L$, $isPathFound$)
    Insert v-t node n into OPEN$_L$
    Sort v-t nodes in OPEN$_L$ in increasing order of f_i-values, breaking ties in favor of v-t nodes with smaller x_i-values
    if $f_i(n) \leq \tau_i(N)$ **and not** $isPathFound$ **then**
        Insert v-t node n into FOCAL$_L$
        Sort v-t nodes in FOCAL$_L$ in increasing order of x_i-values, breaking ties in favor of v-t nodes with smaller $\tilde{f}_i$-values

procedure UpdatePriority(n, $\bar{n}$, $\tau_i(N)$, CLOSED$_L$, OPEN$_L$, FOCAL$_L$, $isPathFound$)
    $(v, t) \leftarrow$ state representation of v-t node n
    $(v, \bar{t}) \leftarrow$ state representation of v-t node $\bar{n}$
    $\bar{f}_i \leftarrow f_i(\bar{n})$
    if $\bar{n} \in$ CLOSED$_L$ **then**
        Remove v-t node $\bar{n}$ from CLOSED$_L$
        InsertNode($\bar{n}$, $\tau_i(N)$, OPEN$_L$, FOCAL$_L$, $isPathFound$)
    else
        Update the priority of $\bar{n}$ in OPEN$_L$
        if $f_i(\bar{n}) \leq \tau_i(N)$ **and not** $isPathFound$ **then**
            if $\bar{f}_i > \tau_i(N)$ **then**
                Insert v-t node $\bar{n}$ into FOCAL$_L$
            else
                Update the priority of v-t node $\bar{n}$ in FOCAL$_L$

Chapter 5

Flow-Based Guidance Framework with Flex Distribution

Although EECBS is the state-of-the-art algorithm for finding a bounded-suboptimal solution for a MAPF instance, its scalability to large-scale MAPF instances with many agents moving in large environments is limited. One approach to accelerating EECBS is to guide the low-level Focal Search with the *highway heuristics*. This is achieved by transforming an edge in the graph into a pair of weighted directed edges, designating a subset of them as the *highways*, and increasing the costs of non-highway edges to penalize traversing them when computing the estimated cost-to-go for an agent from a vertex to its target. However, the performance of EECBS with the highway heuristic is limited to Kiva-like warehouses [15], where agents move across a region with corridors when transitioning from one free region to another.* In the context of *lifelong MAPF* [37] (i.e., a MAPF variant where agents need to visit multiple target vertices sequentially), Chen et al. [12] guide the low-level heuristic search based on the paths that agents have traversed. Zhang et al. [74] extend the highway heuristics to *guidance graph optimization* for lifelong MAPF. However, it is non-trivial to transfer these approaches from lifelong MAPF to bounded-suboptimal MAPF. In the previous chapters, we have proposed Flex Distribution to accelerate EECBS. In this chapter, we leverage EECBS with Flex Distribution and propose the *Flow-Based Guidance Framework*. It is a pre-processing technique that generates the *Flow-Based Guidance Heuristic*, which guides the low-level Focal Search while ensuring that EECBS with Flex Distribution remains bounded-suboptimal and solution-complete. The flow is the

*An example of a Kiva-like warehouse is the `warehouse` graph in Section 3.8.1.

number of agents that traverse an edge of the graph of a MAPF instance in a direction when following their paths. Given a MAPF instance, the Flow-Based Guidance Framework generates flows through *simulation* by finding paths for agents on a weighted directed graph with strategies that are designed for bounded-suboptimal MAPF. Based on the flows, the Flow-Based Guidance Framework then constructs the *Flow-Based Guidance Graph* and computes the Flow-Based Guidance Heuristic. Our contributions are as follows:

- We introduce the two-phase Guidance Framework as a general pre-processing approach for MAPF.
- We propose the Flow-Based Guidance Framework, which leverages observations from bounded-suboptimal MAPF and Flex Distribution to derive the Flow-Based Guidance Heuristic that guides the low-level Focal Search.
- Our Flow-Based Guidance Framework can efficiently output the Flow-Based Guidance Heuristic to accelerate EECBS with Flex Distribution effectively. Our empirical evaluation shows that EECBS with Flex Distribution with the Flow-Based Guidance Heuristic is more efficient than that with the state-of-the-art guidance heuristics. Also, EECBS with Flex Distribution with the Flow-Based Guidance Heuristic solves MAPF quickly on graphs not limited to Kiva-like warehouses.

5.1 Related Work for Guidance Heuristics

In this section, we review related work on generating guidance heuristics. We first introduce the highway heuristic, which, to our knowledge, is the only related work for bounded-suboptimal MAPF. Then, we introduce guidance heuristics for lifelong MAPF and (unbounded) suboptimal MAPF.

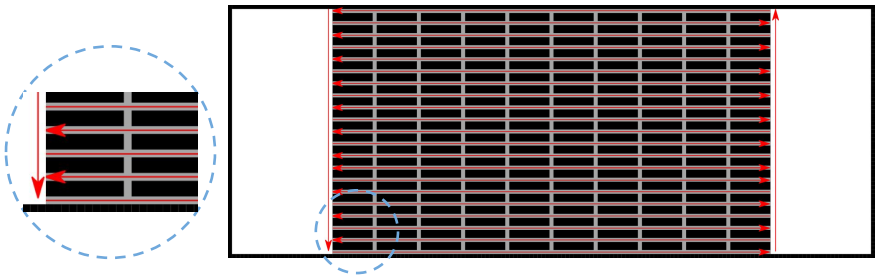


Figure 5.1: `Warehouse` graph with obstacle-free regions on two sides, marked in white, and corridors of width 1, marked in gray. The red arrows indicate the human-designed Criss Cross highway. The contents of the blue dashed circle show an enlarged portion of the graph.

5.1.1 Highway Heuristic

To improve the efficiency of bounded-suboptimal MAPF algorithms that are based on the Conflict-Based Framework, Cohen et al. [15] proposed guiding the low-level Focal Search with a set of predefined *highways*. Based on the four-neighbor undirected grid graph $G = (V, E)$ with a finite number of vertices (and edges) from a MAPF instance, Cohen et al. [15] create a weighted directed graph $G' = (V, E', W)$, where each undirected edge $e = \{u, v\} \in E$ is mapped to a pair of weighted directed edges $e_{uv} = (u, v) \in E'$ and $e_{vu} = (v, u) \in E'$ with their respective costs $w_{uv} \in W$ and $w_{vu} \in W$. The highways are defined as a subset of directed edges $E_{hwy} \subseteq E'$. The cost of each directed edge is set to 1 if it belongs to the highways, and set to w otherwise. To guide the low-level Focal Search for finding a path for an agent a_i, Cohen et al. [15] determine the *highway heuristic* $h_{hwy,i}(v)$, which is the cost of a minimum-cost path from vertex $v \in V$ to the target vertex l_i on the weighted directed graph G'. The highway heuristic is then used as the secondary heuristic $\tilde{h}_i(n) = h_{hwy,i}(v)$ for the low-level Focal Search. If multiple v-t nodes in FOCAL$_L$ have the minimum number of conflicts of all v-t nodes in FOCAL$_L$, then the low-level Focal Search selects the v-t node n with the minimum $\tilde{f}_i(n) = g_i(n) + \tilde{h}_i(n) = g_i(n) + h_{hwy,i}(v)$ among them for expansion. This biases the path found for agent a_i toward the highways.

To determine the highway, Cohen et al. [15] propose three approaches: the Criss Cross-based (CC-based), the Graph Model-based (GM-based), and the Heat Map-based (HM-based) approaches. As shown

in Figure 5.1, the CC-based approach is typically used in Kiva-like warehouses, where obstacles are placed in rows in the middle, creating horizontal corridors between them, and the two sides of the graph are obstacle-free regions. In an obstacle-free region, we say a border is all the vertices on the column that connects the horizontal corridors. The highways from the CC-based approach consist of the edges in one direction, the edges of the next horizontal corridor, and so on. The highways also consist of the edges in vertical directions between the obstacle-free regions and the region of horizontal corridors. In other words, the CC-based approach relies on a customized design based on human ingenuity.

Cohen et al. [15] also propose the GM-based and HM-based approaches to guide the low-level Focal Search on graphs not limited to Kiva-like warehouses.

Given a MAPF instance, the GM-based approach constructs a graph model that contains several variables for each vertex on the graph of the MAPF instance. Then, it constructs the highways by solving the maximum a posteriori estimation problem on the graph model, which is an optimization problem that takes the variables and "probabilistic constraints" among them as input. However, since the number of variables increases in proportion to the number of vertices, the GM-based approach suffers from a significant computational overhead when solving the maximum a posteriori estimation problem for large-scale MAPF instances. We have run the source code of the GM-based approach from Cohen et al. [15]. The 60-second runtime limit was exceeded on all graphs used in our empirical evaluation.

Given a MAPF instance with the graph $G = (V, E)$, the HM-based approach is an iterative process. It begins with a weighted directed graph with the same set of vertices as V. For each edge $e = (u, v) \in E$, the HM-based approach adds a pair of directed edges e_{uv} and e_{vu} to its weighted directed graph, where e_{uv} and e_{vu} represent the directed edge from vertex u to vertex v and from vertex v to vertex u, respectively. The cost of each directed edge is initialized to 1. At each iteration, the HM-based approach randomly selects a pair of start and target vertices (which may not belong to the ones specified in the given MAPF instance). Then, it finds a minimum-cost path on its weighted directed graph. The cost of each directed

edge e_{uv} is recalculated based on how often e_{uv} and e_{vu} have appeared in the paths found at the previous iterations, and the process repeats. When the number of iterations reaches a user-specified limit, the HM-based approach selects a subset of directed edges as the highways based on the cost of each directed edge. Empirically, we discovered that the HM-based approach can misguide the low-level Focal Search. That is, EECBS with Flex Distribution and the highway heuristic from the HM-based approach is less efficient than that without any guidance.

5.1.2 Guidance Heuristics of Other MAPF Variants

To the best of our knowledge, the highway heuristic from Cohen et al. [15] is the only work that guides the low-level heuristic search for bounded-suboptimal MAPF. Even so, it has motivated researchers to explore ideas for guiding the low-level heuristic search in the context of two other MAPF variants, namely, lifelong MAPF and suboptimal MAPF.

For lifelong MAPF, an agent receives its next target vertex once it is at its current one. In this case, Chen et al. [12] propose Traffic Flow Optimization (TFO), which coordinates the paths of the agents by updating their heuristics based on the traffic flows of the *time-independent paths*, which are paths that do not allow wait actions. Similar to the HM-based approach, TFO then constructs a weighted directed graph, where the edge costs are derived from handcrafted functions based on human knowledge that take the time-independent paths as inputs. As we apply TFO for guiding the low-level Focal Search for EECBS with Flex Distribution in the Empirical Evaluation section, since TFO is designed for lifelong MAPF, directly applying it to generate the heuristic that guides the low-level Focal Search for EECBS with Flex Distribution results in inefficiency (see Section 5.4.4). On the other hand, Zhang et al. [74] propose an approach that generates the heuristic by constructing a weighted directed graph that represents the estimated action costs associated with each vertex, called the guidance graph. While running lifelong MAPF, Zhang et al. [74] use a learning-based model to generate the future edge costs based on the current edge costs of the

guidance graph. However, similar to the GM-based approach [15], the number of variables in the guidance graph scales with the number of vertices, resulting in huge computational overhead for training the model and thus limiting scalability. In this dissertation, we target large graphs with more than 5,000 vertices, which their approach cannot handle.

For suboptimal MAPF, the objective is to find a solution with an SOC that is not necessarily optimal or bounded-suboptimal. In this case, Wang and Botea [65] propose a rule-based method that forces the agents to move in a particular direction. However, similar to the CC-based approach, this approach is graph-dependent and requires human ingenuity. On the other hand, Han and Yu [20] propose Space Utility Optimization (SUO), which guides the low-level heuristic search along paths that traverse vertices/edges with low congestion. Empirically, since SUO is designed for suboptimal MAPF without any guarantees on solution quality, directly applying it to generate the heuristic that guides the low-level Focal Search for EECBS with Flex Distribution results in inefficiency (see Section 5.4.4).

5.2 Guidance Framework for MAPF

Recently, pre-processing MAPF instances to avoid conflicts or congestion before running a MAPF algorithm has been popular. Based on the related work, we introduce a general framework, called the *Guidance Framework*, which consists of two phases: the *path-simulation phase* (P1) and the *heuristic-calculation phase* (P2). In Phase (P1), given the graph $G = (V, E)$ of a MAPF instance, the Guidance Framework constructs a weighted directed graph $G' = (V, E', W')$, called the guidance graph [74]. For each undirected edge $e = \{u, v\} \in E$, there is a pair of directed edges $e_{uv} = (u, v),\ e_{vu} = (v, u) \in E'$, which represent the directed edge from vertex u to vertex v and from vertex v to vertex u, respectively. Each directed edge e_{uv} has the cost $w_{uv} \in W'$. In Phase (P1), the Guidance Framework finds paths in the graph G' and then updates the edge costs of the directed edges based on these paths. Since the guidance graph G' differs from the graph G of the given MAPF instance, we call finding a path in G' a *simulation*, and a path found

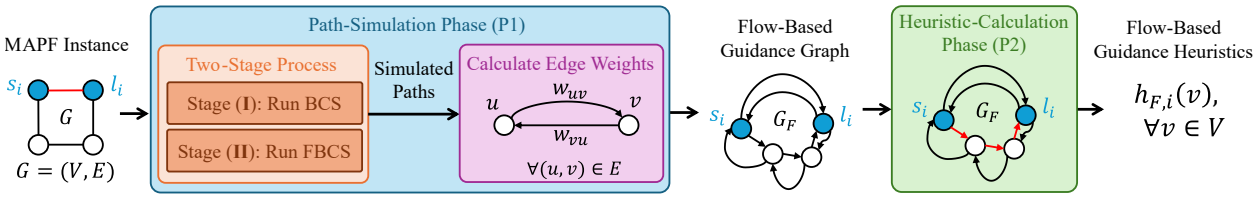


Figure 5.2: The flow chart of the Flow-Based Guidance Framework, which takes a MAPF instance on a graph $G = (V, E)$ as input and returns the Flow-Based Guidance Heuristic $h_{F,i}$ for each agent a_i on each vertex $v \in V$. The shortest paths from the start vertex s_i to the target vertex l_i of agent a_i in graphs G and G_F are marked in red.

is called a *simulated* path. In Phase (P2), the Guidance Framework generates a *guidance heuristic* for each agent a_i by computing the cost of a minimum-cost path in the guidance graph G' from each vertex $v \in V$ to the target vertex l_i. The guidance heuristic is then used as the secondary heuristic (see Section 2.4.2) of the low-level Focal Search.

The Guidance Framework generalizes some of the related work. For example, the Guidance Framework can be applied to the HM-based approach. That is, the HM-based approach [15] repeatedly selects a start vertex and a target vertex and then simulates a path from the start vertex to the target vertex on the weighted directed graph in Phase (P1). It then calculates the highway heuristic from the weighted directed graph as guidance at Phase (P2). The Guidance Framework can also be applied to approaches that solve other MAPF variants. For example, TFO [12] selects all agents in the lifelong MAPF instance and simulates their time-independent paths on a weighted directed graph in Phase (P1), and then calculates the guidance heuristic via a backward breadth-first search in Phase (P2).

5.3 Flow-Based Guidance Framework (FBGF)

Although several approaches have been proposed for lifelong and suboptimal MAPF, generating heuristics that efficiently guide agents in the context of bounded-suboptimal MAPF remains challenging, especially for large-scale MAPF instances. This is so because, unlike lifelong MAPF, bounded-suboptimal MAPF has the following features:

(F1) Agents have to wait at their target vertices permanently,

(F2) the solution has to be bounded-suboptimal, and

(F3) the MAPF algorithm has to find a solution within a given runtime limit.

Feature (F2) is the difference between bounded-suboptimal MAPF and (unbounded) suboptimal MAPF.

To guide the low-level Focal Search and accelerate EECBS with Flex Distribution, we follow our two-phase Guidance Framework and propose the *Flow-Based Guidance Framework* (FBGF) as a pre-processing technique. Figure 5.2 shows the flow chart of the Flow-Based Guidance Framework. In Phase (P1), FBGF simulates paths and constructs the *Flow-Based Guidance Graph* (FGG) $G_F = (V, E_F, W_F)$. In Phase (P2), it uses the FGG to compute the *Flow-Based Guidance Heuristic* (FH) for each agent, which maps vertices to costs-to-go.

5.3.1 Path-Simulation Phase (P1)

In Phase (P1), FBGF simulates one path for an agent one by one. To achieve Features (F1), (F2), and (F3) during the simulation, FBGF uses the following strategies:

(S1) Viewing the target vertices of the other agents as obstacles,

(S2) using a two-stage low-level *Flexible Bounded-Cost Search* (FBCS), and

(S3) sorting all agents $\{a_i \mid i \in [k]\}$ in increasing order of their $h_i(s_i)$ and limiting the number of simulated paths.

EECBS with Flex Distribution determines $h_i(s_i)$ as the cost of a minimum-cost path of an agent a_i from its start vertex s_i to its target vertex l_i, which is a lower bound on the cost of the path for agent a_i in an optimal solution.

5.3.1.1 Achieving Feature (F1)

In terms of Feature (F1), after an agent waits at its target vertex permanently, it may encounter numerous target conflicts (see Section 2.4.3.2) with the other agents, which may cause computational overhead to resolve, even when symmetry breaking [31] is used. Thus, to achieve Feature (F1), FBGF uses Strategy (S1) that guides agents to avoid target conflicts. When finding a simulated path for an agent a_i, FBGF views the target vertices $\{l_j \mid j \in [k] \setminus \{i\}\}$ of all the other agents as obstacles, known as the *target obstacles*. That is, each simulated path that FBGF uses for constructing the FGG for an agent a_i never lets agent a_i be at any target vertices except for target vertex l_i.

5.3.1.2 Achieving Feature (F2)

In terms of Feature (F2), FBGF uses Strategy (S2), i.e., the *two-stage low-level (F)BCS*. It is a two-stage process (see the orange block in Figure 5.2) that first runs the low-level Bounded-Cost Search (BCS) in Stage (I) and then runs the low-level Flexible Bounded-Cost Search (FBCS) at Stage (II). In Stage (I), FBGF sorts all k agents of the given MAPF instance and uses the low-level BCS to find a simulated path for each agent in that order. The low-level BCS uses the target obstacles from Strategy (S1) when finding a simulated path. It maintains an OPEN$_L$ list that contains all generated but not yet expanded v-t nodes n (i.e., the frontier of the search tree of the v-t nodes). However, instead of sorting the v-t nodes n in increasing order of their $f_i(n)$ (like the OPEN$_L$ in A*), the low-level BCS sorts them in increasing order of their number of conflicts $x_i(n)$, breaking ties in favor of v-t nodes with smaller h_i-values. If there are v-t nodes with the same number of conflicts and the same h_i-value, then the low-level BCS breaks ties in favor of v-t nodes with smaller values of $g_i(n)$, and then random selection for any further unresolved ties. For each v-t node n with the state representation (v, t), the low-level BCS sets $h_i(n) = h_i(v)$ as in the low-level Focal Search, which is the cost of a minimum-cost path from vertex v to the target vertex l_i. Thus, $h_i(n) = h_i(v)$ is admissible.

Algorithm 5.1 shows the pseudo-code of the low-level (F)BCS. The blue text marks the differences between the low-level BCS and the low-level FBCS. Suppose that FBGF uses a list P to store all the simulated paths that have been found by either the low-level BCS or the low-level FBCS [Line 1]. $J = \{j \in [k] \mid P[j]$ is **not** null$\}$ is the set of indices of agents whose simulated paths have been found [Line 5]. The low-level BCS begins by generating a root v-t node n_0 with the state representation $(s_i, 0)$ and adding it to OPEN$_L$. Since $h_i(n) = h_i(v)$ is an admissible heuristic for any v-t node n with state representation (v, t), $f_i(n_0) = 0 + h_i(n_0) = h_i(s_i)$ is a lower bound on the cost of the path of agent a_i in an optimal solution. At each iteration, the low-level BCS selects the v-t node $\hat{n}$ for expansion that has the minimum number of conflicts among all v-t nodes in OPEN$_L$, i.e., $\hat{n} = \arg\min_{n \in \text{OPEN}_L} x_i(n)$. That is, the low-level BCS finds a path with the cost at most the threshold τ_i that has the minimum number of conflicts with the existing simulated paths. To deploy Strategies (S1) and (S2), the low-level BCS only generates a child v-t node n' with state representation (v', t') if $v' \notin \{l_j \mid j \in [k] \setminus \{i\}\}$ and $f_i(n') \leq w \cdot h_i(s_i)$ [Line 22]. That is, the low-level BCS uses the threshold $\tau_i = w \cdot h_i(s_i)$. The low-level BCS terminates either when a simulated path is found or when OPEN$_L$ is empty. If the low-level BCS finds a simulated path, then its cost c_i is at most the bounded-suboptimality factor w larger than $h_i(s_i)$, i.e., $c_i \leq w \cdot h_i(s_i)$. Thus, the SOC of all simulated paths in list P is at most the bounded-suboptimality factor w larger than its SOLB.

Due to the target obstacles from Strategy (S1) and the bounded-cost criterion, the low-level BCS may not be able to simulate a path for an agent a_i of a cost that is at most the threshold $w \cdot h_i(s_i)$. If we use EECBS with Flex Distribution to solve a MAPF instance, then the solution is guaranteed bounded-suboptimal as long as each CT node is locally bounded-suboptimal (see Theorem 3.1). Thus, FBGF can further increase the threshold when running the low-level BCS by using the flex from the other simulated paths. The resulting simulated path, if a path exists, can still be used as the path in the root CT node of EECBS with Flex Distribution. Thus, in Stage (II), we propose the low-level *Flexible Bounded-Cost Search* (FBCS) to simulate a path for each agent for which the low-level BCS failed to find one in Stage (I). When

finding a simulated path for an agent a_i for which the low-level BCS failed to find one in Stage (I) (i.e., $P[i] = \text{null}$), the low-level FBCS increases the threshold τ_i by adding the *flex* of the simulated paths of the other agents $\{a_j \mid j \in J\}$ that have been found in either Stage (I) or Stage (II) [Line 8]. That is,

$$\tau_i = w \cdot h_i(s_i) + \sum_{j \in J} (w \cdot h_j(s_j) - c_j), \tag{5.1}$$

where c_j is the cost of the simulated path $P[j]$ and $w \cdot h_j(s_j) - c_j$ is its flex. After the low-level FBCS finds a simulated path for agent a_i, if a path exists, FBGF adds it to the simulated path list P. In this case, the SOC of all simulated paths in list P satisfies

$$\begin{aligned} \sum_{j \in J \cup \{i\}} c_j &\leq \tau_i + \sum_{j \in J} c_j \\ &= w \cdot h_i(s_i) + \sum_{j \in J} (w \cdot h_j(s_j) - c_j) + \sum_{j \in J} c_j \\ &= w \cdot \sum_{j \in J \cup \{i\}} h_j(s_j). \end{aligned} \tag{5.2}$$

Thus, the SOC over all simulated paths in list P is still at most the bounded-suboptimality factor w larger than the SOLB, i.e.,

$$\sum_{j \in J} c_j \leq w \cdot \sum_{j \in J} h_j(s_j), \text{ where } J = \{j \in [k] \mid P[j] \text{ is } \textbf{not} \text{ null}\}. \tag{5.3}$$

5.3.1.3 Achieving Feature (F3)

In terms of Feature (F3), simulating paths for all the k agents can result in a huge runtime overhead (see Table 5.6). Thus, to achieve Feature (F3), FBGF uses Strategy (S3) that sorts all k agents $\{a_i \mid i \in [k]\}$ in increasing order of their $h_i(s_i)$ and then limits the number of simulated paths to at most a user-specified *path-found threshold* $k_{\max} \leq k$. Since we follow Li et al. [34] and compute $h_i(s_i)$ for each agent a_i from

each vertex $v \in V$ to its target vertex l_i as an admissible heuristic for the low-level Focal Search, using $h_i(s_i)$ to sort the agents does not result in huge runtime overhead. Also, since $h_i(s_i)$ is the cost of a minimum-cost path of an agent a_i from its start vertex s_i to its target vertex l_i, agent a_i with low $h_i(s_i)$ often has a path that avoids target obstacles and conflicts with the simulated paths of the other agents with low cost, which results in the low-level search expanding fewer v-t nodes to find it, thereby reducing the runtime overhead of FBGF. Empirically, sorting all k agents $\{a_i \mid i \in [k]\}$ in increasing order of their $h_i(s_i)$ results in EECBS with Flex Distribution finding solutions more quickly than sorting all k agents in decreasing order of their $h_i(s_i)$ and sorting them randomly (see Figure 5.11). In general, FBGF reduces its runtime by limiting the number of simulated paths and generates effective FH by sorting the agents.

We now combine Strategies (S1), (S2), and (S3). FBGF simulates a path for each agent in a sorted sequence, where a simulated path has to avoid all target obstacles while minimizing the number of conflicts with the existing simulated paths. FBGF first uses the low-level BCS during Stage (I) to simulate a path. If the number of simulated paths reaches the path-found threshold $k_{\max} \leq k$, then FBGF stops the simulation. Otherwise, FBGF uses the low-level FBCS during Stage (II) to simulate a path for each agent for which the low-level BCS failed to find a path. FBGF stops the simulation when either the number of simulated paths reaches the path-found threshold, or it has run one low-level FBCS for each agent for which the low-level BCS failed to find a path.

Then, FBGF constructs the FGG $G_F = (V, E_F, W_F)$ and determines its edge costs based on the simulated paths. We define the *flow* of a directed edge as the number of agents traversing it when following their simulated paths. Based on the flow of a directed edge, FBGF normalizes its edge cost in the range $[1, c_p]$, where c_p is the user-specified *maximum penalty cost*. FBGF calculates the weights $w_{uv} \in W_F$ of a directed edge e_{uv}

$$w_{uv} = 1 + (c_p - 1) \cdot \frac{\Phi_{\max} - \Phi_{uv}}{k}, \tag{5.4}$$

where $\Phi_{\max}$ is the maximum flow over all simulated paths (i.e., the maximum number of agents traversing a directed edge). According to Equation 5.4, the costs of the directed edges are negatively correlated with their flows. The costs of directed edges with the maximum flow (i.e., $\Phi_{uv} = \Phi_{\max}$) are set to 1, and those with zero flow (i.e., $\Phi_{uv} = 0$) are set to c_p if $\Phi_{\max} = k$. That is, FBGF promotes directed edges with higher flow (i.e., those traversed by more agents) by giving them smaller costs.

Algorithm 5.2 shows the pseudo-code of Phase (P1) of FBGF. It begins with using BCS in Stage (I) [Lines 5-12] and FBCS in Stage (II) [Lines 13-22]. After that, it calculates the edge costs of FGG and returns FGG accordingly [Lines 25-35].

5.3.2 Heuristic-Calculation Phase (P2)

In Phase (P2), FBGF calculates the *Flow-Based Guidance Heuristic* (FH) for each agent from the FGG constructed during Phase (P1). Since the FGG is a guidance graph, we define the FH-value $h_{F,i}(v)$ of a vertex v for an agent a_i as the cost of a minimum-cost path from vertex v to the target vertex l_i in the FGG G_F. Given a vertex v, its FH-value $h_{F,i}(v)$ for an agent a_i serves as a "guidance" that indicates which vertex v to expand during the low-level Focal Search. EECBS with Flex Distribution then follows Cohen et al. [15] and uses FH as the secondary heuristic. That is, given a v-t node n with its state representation (v, t), $\tilde{h}_i(n) = \tilde{h}_i(v) = h_{F,i}(v)$. According to Equation 5.4, the costs of the directed edges in the FGG are negatively correlated with their flows. The rationale is as follows: The FGG is based on the paths found by the low-level (F)BCS, which aims to minimize the number of conflicts of a path with the existing simulated paths. Thus, the paths of agents that follow the flows obtained from the simulated paths may have fewer conflicts with each other than those with the minimum costs (see Table 5.5).

5.3.3 Bounded-Suboptimality of Flow-Based Guided Heuristic

By using the FH $h_{F,i}(v)$ of a vertex $v \in V$ as the secondary heuristic $\tilde{h}_i(n)$ of a v-t node v with state representation (v, t), i.e., $\tilde{h}_i(n) = \tilde{h}_i(v) = h_{F,i}(v)$, the low-level Focal Search of EECBS with Flex Distribution and FH still finds a path of a cost at most the threshold. Thus, each CT node remains locally bounded-suboptimal.

Lemma 5.1. *Suppose that the distributed flex satisfies Inequality 3.15 when the low-level Focal Search with Flex Distribution finds a path for an agent that satisfies the constraints. For any secondary heuristics of the low-level Focal Search, EECBS with Flex Distribution is guaranteed to find a bounded-suboptimal solution to a MAPF instance if a solution exists.*

Proof. Since the secondary heuristic is only used to break ties when two v-t nodes in FOCAL$_L$ have the same minimum number of conflicts, all v-t nodes n in FOCAL$_L$ have their $f_i(n)$ at most the threshold, resulting in a path of cost at most the threshold. Thus, each CT node remains locally bounded-suboptimal, and EECBS with Flex Distribution is thus guaranteed to find a bounded-suboptimal solution to a MAPF instance if a solution exists, according to Theorem 3.7. □

Theorem 5.1. *Suppose that FBGF finds a simulated path $P[j]$ with cost c_j in Phase (P1) for an agent a_j and $J = \{j \in [k] \mid P[j]$ is **not** null$\}$ is the set of indices of agents whose simulated paths are found by FBGF. Also, suppose that we use EECBS with Flex Distribution and FH to find a solution for a MAPF instance, where the distributed flex satisfies Inequality 3.15. For each agent a_j with a simulated path (i.e., $j \in J$), EECBS with Flex Distribution and FH uses this path as the path for agent a_j in the root CT node N_0. On the other hand, for each agent a_i without a simulated path (i.e., $i \in [k] \setminus J$), EECBS with Flex Distribution and FH uses the low-level Focal Search with zero distributed flex to find a path for agent a_j in the root CT node N_0. Then, EECBS with Flex Distribution and FH is guaranteed to find a bounded-suboptimal solution if a solution exists.*

Proof. Since $h_i(s_i)$ is admissible and the root CT node N_0 does not contain any constraint, a lower bound $lb_i(N_0)$ on the cost of a minimum-cost path of each agent a_i is $h_i(s_i)$. According to Inequality 5.3, given the set $J = \{j \in [k] \mid P[j] \text{ is } \textbf{not} \text{ null}\}$ of indices of agents whose simulated paths are found, the SOC over all these simulated paths in list P is at most the bounded-suboptimality factor w larger than the SOLB, i.e.,

$$\sum_{j \in J} c_j \leq w \cdot \sum_{j \in J} h_j(s_j) = w \cdot \sum_{j \in J} lb_j(N_0). \tag{5.5}$$

Also, when EECBS with Flex Distribution generates the root CT node N_0, for each agent a_i without a simulated path (i.e., $i \in [k] \setminus J$), the low-level Focal Search with zero distributed flex finds its path with the cost $c_i(N_0)$ satisfying

$$c_i(N_0) \leq w \cdot h_i(s_i) = w \cdot lb_i(N_0). \tag{5.6}$$

Also, Thus, the SOC $C(N_0)$ of the root CT node N_0 satisfies

$$\begin{aligned} C(N_0) &= \sum_{j \in J} c_j + \sum_{i \in [k] \setminus J} c_i(N_0) \\ &\leq w \cdot \sum_{j \in J} h_j(s_j) + w \cdot \sum_{i \in [k] \setminus J} h_i(s_i) \\ &= w \cdot \sum_{i \in [k]} h_i(s_i) \\ &= w \cdot \sum_{i \in [k]} lb_i(N_0) \\ &= w \cdot LB(N_0), \end{aligned} \tag{5.7}$$

indicating that the root CT node is N_0 locally bounded-suboptimal. Since the rest of EECBS with Flex Distribution remains unchanged, according to Theorem 3.7, it is guaranteed to find a bounded-suboptimal solution for a MAPF instance if a solution exists. □

To evaluate the efficiency of EECBS with Flex Distribution and FH, the runtime of FBGF needs to be considered. Thus, when we use EECBS with Flex Distribution to find a solution and use the simulated paths as the paths in the root CT node, we can save runtime by not running the low-level Focal Search for agents that already have the simulated paths.

5.4 Empirical Evaluation

In this section, we first introduce the configuration settings used for the empirical evaluation. Next, we describe how we implement the state-of-the-art guidance heuristics for bounded-suboptimal MAPF. Then, we use EECBS and EECBS with Mixed-Strategy Flex Distribution (EECBS-MFD) as baseline bounded-suboptimal MAPF algorithms. We first evaluate the success rates and runtimes of EECBS-MFD and FH (EECBS-MFD-FH) for FBGF with different values of the path-found threshold $k_{\max}$ and the maximum penalty cost c_p. Then, based on EECBS-MFD, we compare its efficiency with different guidance heuristics. We also provide some empirical insights on why FH is effective, along with a case study. Lastly, we evaluate the effectiveness of Strategies (S1), (S2), and (S3) by implementing different FBGF variants.

5.4.1 Configuration Settings

We use the same MAPF instances as the ones in Section 3.8.1 with the bounded-suboptimality factor $w = 1.10$ on the `city`, `den520d`, and `ost003d` graphs. The number of agents used for creating MAPF instances on each graph is shown in the $w = 1.10$ rows of Table 3.2. As for the `warehouse` graph, unlike the MAPF instances in Section 3.8.1, we follow Cohen et al. [15] and set the start and target vertices on the free regions on both sides. The number of agents used for creating MAPF instances on the `warehouse` graph ranges from 300 to 700 with an increment of 100. We use the 60-second runtime limit unless mentioned otherwise. We evaluate runtime as the sum of the runtime to generate a guidance heuristic and the runtime to run EECBS-MFD with it. We set the runtime of a MAPF algorithm on a MAPF instance to 60 seconds if the

Algorithm abbreviation	Full name
CC-based approach	Criss Cross-based highway heuristics [15]
HM-based approach	Heat Map-based highway heuristics [15]
GM-based approach	Graph Model-based highway heuristics [15]
TFO	Traffic Flow Optimization [12]
SUO	Space Utility Optimization [20]

Table 5.1: The state-of-the-art guidance heuristics evaluated for bounded-suboptimal MAPF.

MAPF algorithm fails to find a solution within the 60-second runtime limit. We implement EECBS with all enhancements in Section 3.8.1 and use Mixed-Strategy Flex Distribution (MFD) as the Flex Distribution mechanism, i.e., EECBS-MFD. We implement all MAPF algorithms in `C++` (compiled with `GCC-11.3.0`) and run the experiments with `CentOS Linux` on an `Intel Xeon-2640 v4` processor with 16 GB of memory.

5.4.2 Implementation of the State-Of-The-Art Guidance Heuristics

We first show how we scale up the CC-based approach for generating its highway heuristic. Then, we show how to integrate the state-of-the-art guidance heuristics for other MAPF variants, i.e., the Traffic Flow Optimization (TFO) for lifelong MAPF and the Space Utility Optimization (SUO) for suboptimal MAPF, into the Guidance Framework for bounded-suboptimal MAPF. Table 5.1 shows the state-of-the-art guidance heuristics used in our empirical evaluation.

5.4.2.1 Implementation of the Highway Heuristics

Cohen et al. [15] show the CC-based approach with 10 MAPF instances with 130 agents on a warehouse graph of size 53×22. We use 25 MAPF instances with up to 700 agents on the `warehouse` graph of size 63×161 from the MAPF benchmark suite [59], as shown in Figure 5.1. In terms of GM-based and HM-based approaches, we run the source code from Cohen et al. [15]. However, we omit the results of the GM-based approach since the runtime of generating its highway heuristic exceeds 120 seconds.

5.4.2.2 Implementation of Traffic Flow Optimization (TFO)

TFO [12] was originally used for lifelong MAPF. To implement TFO for EECBS-MFD, we rely on the Guidance Framework and use the low-level Focal Search to simulate a time-independent path for each agent one after the other in Phase (P1). When the low-level Focal Search expands a v-t node $\hat{n}$ with state representation (u, t) and generates a child v-t node n with state representation $(v, t+1)$, instead of increasing the g-value by 1, we update it based on the *contraflow* [12], i.e.,

$$g(n) = g(\hat{n}) + (1 + \Phi_{uv} \times \Phi_{vu}). \tag{5.8}$$

Once a time-independent path is found, we update the flows of the directed edges that the path traverses, which affects their contraflows and, consequently, the edge costs when finding paths for subsequent agents. Then, during the heuristics calculation phase, we simply set the cost of a directed edge to the flow from the simulated paths.

5.4.2.3 Implementation of Space Utility Optimization (SUO)

SUO [20] was originally used for suboptimal MAPF. We regard SUO as an online guidance to avoid congestion for the low-level Focal Search of EECBS-MFD. Before the low-level Focal Search finds a path for an agent in a child CT node, SUO constructs a heatmap $\mathcal{T}$ from the paths of the other agents. The value $\mathcal{T}(v)$ (or $\mathcal{T}(u, v)$) of a vertex v (or, respectively, a directed edge (u, v)) in the heatmap indicates the number of agents that traverse the vertex (or, respectively, thee directed edge), called the utility. The low-level Focal Search uses the utility as the secondary heuristic. When the low-level Focal Search expands a v-t node $\hat{n}$ with state representation $(u, t-1)$ and generates a child v-t node n with state representation (v, t), we follow the SUO variant called SU-I [20] and compute the secondary heuristic as

$$\tilde{h}_i(n) = h(v) + h_{\text{SU-I}}(u, v), \tag{5.9}$$

		$k_{\max}=0$		$k_{\max}=0.25k$		$k_{\max}=0.5k$		$k_{\max}=0.75k$		$k_{\max}=k$	
		SR	RT	SR	RT	SR	RT	SR	RT	SR	RT
city	$c_p=5$	0.63	42.71	0.78	33.23	0.93	24.43	0.94	19.64	0.81	28.29
	$c_p=10$	0.62	42.78	0.80	30.39	0.95	19.43	0.94	17.87	0.73	29.80
	$c_p=20$	0.69	42.45	0.90	27.40	0.95	18.79	**0.98**	**15.02**	0.81	27.92
den520d	$c_p=5$	0.46	48.52	0.53	43.90	0.65	36.56	0.74	32.46	0.51	40.29
	$c_p=10$	0.44	48.50	0.61	41.20	0.81	30.35	0.74	29.18	0.53	39.95
	$c_p=20$	0.45	48.30	0.66	38.37	0.75	29.62	**0.82**	**26.61**	0.52	39.83
ost003d	$c_p=5$	0.45	44.43	0.46	41.44	0.57	35.51	0.65	29.82	0.74	26.97
	$c_p=10$	0.42	44.46	0.50	39.12	0.62	31.65	0.71	25.00	**0.76**	25.37
	$c_p=20$	0.43	44.37	0.50	38.63	0.60	31.35	0.73	**23.01**	0.74	24.96
warehouse	$c_p=5$	0.81	18.44	0.93	11.95	0.95	9.96	0.97	7.96	**1.00**	7.70
	$c_p=10$	0.81	18.78	0.88	12.88	0.98	8.64	0.97	8.28	**1.00**	8.19
	$c_p=20$	0.81	18.70	0.90	12.71	0.90	12.32	**1.00**	**6.05**	0.96	8.44

Table 5.2: Success rates (SR) and average runtime (RT) for the 60-second runtime limit over all MAPF instances on each graph. Numbers in bold indicate the highest SR or lowest RT on each graph.

where

$$h_{\text{SU-I}}(u,v) = 0.5 \cdot \frac{\mathcal{T}(u)}{k} + 0.5 \cdot \frac{\mathcal{T}(v,u)}{k}. \tag{5.10}$$

When two v-t nodes in FOCAL$_L$ have the minimum number of conflicts, the low-level Focal Search breaks ties by selecting the one with the smaller $\tilde{h}_i$-value for expansion.

5.4.3 Hyper-Parameter Tuning for the Flow-Based Guidance Framework

We evaluate FBGF over the path-found thresholds $k_{\max} = \{0, 0.25k, 0.5k, 0.75k, k\}$ and the following maximum penalty costs $c_p = \{5, 10, 20\}$. Table 5.2 shows the success rates and the average runtimes of EECBS-MFD-FH over all MAPF instances on each graph. For each graph and each maximum penalty cost c_p, the average runtime decreases when the path-found threshold $k_{\max}$ increases from 0 to $0.75k$, which shows the effectiveness of the number of simulated paths in accelerating EECBS-MFD. However, for large graphs, such as the `city` and the `den520d` graphs, and each maximum penalty cost c_p, the average runtime increases when the path-found threshold $k_{\max}$ increases from $0.75k$ to k, which shows that finding simulated paths can result in a huge runtime overhead. On the other hand, as the maximum penalty cost c_p increases, agents tend to stick to the guided paths, which can reduce runtime when the guidance

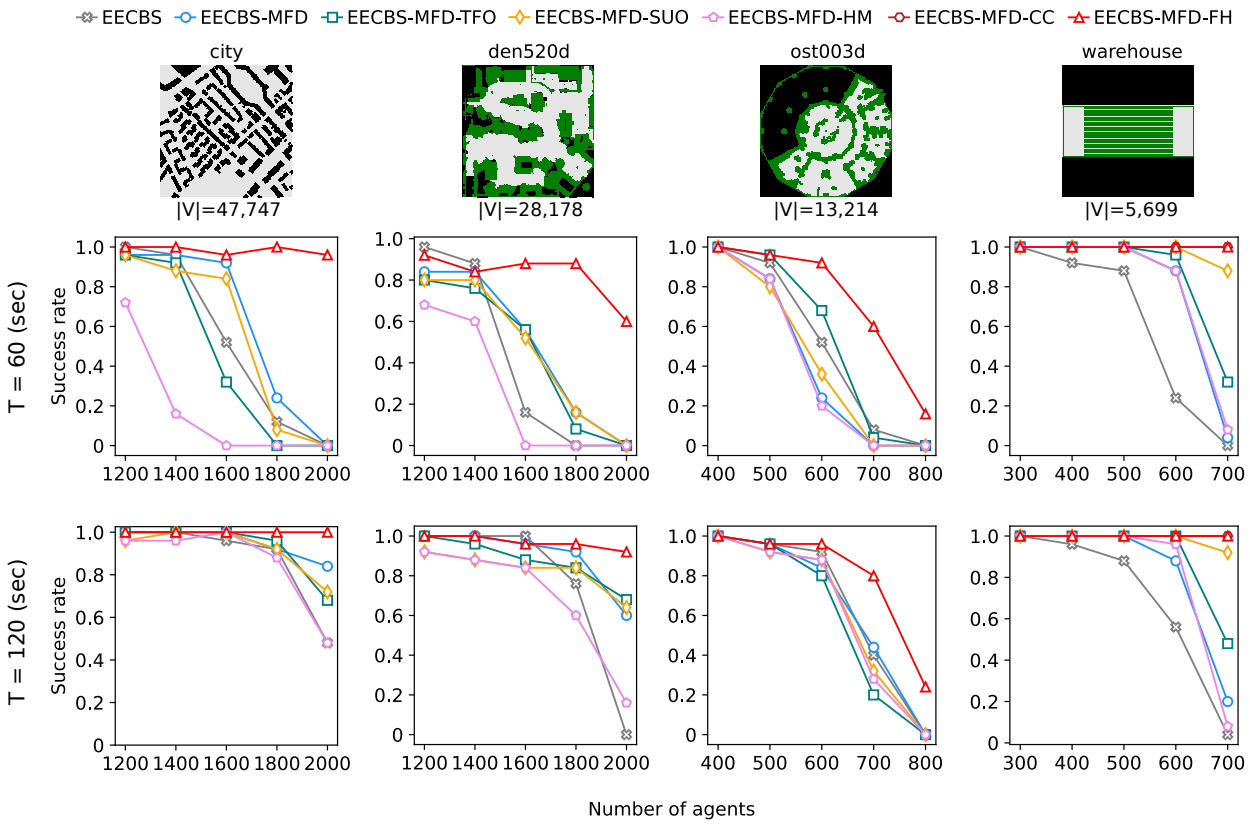


Figure 5.3: Success rates of EECBS, EECBS-MFD, and EECBS-MFD with different guidance heuristics (TFO, SUO, HM, CC, and FH) for the 60-second and 120-second runtime limits, respectively, the bounded-suboptimality factor $w = 1.10$, and each number of agents over all MAPF instances on each graph. $|V|$ indicates the number of vertices on each graph.

effectively accelerates EECBS-MFD. For example, for each graph and the path-found $k_{\max} = 0.75k$, the average runtime decreases when the maximum penalty cost c_p increases.

5.4.4 Performance Comparison

We set the path-found threshold $k_{\max} = 0.75$ and the maximum penalty cost $c_p = 20$ since they, in general, yield the highest success rates and the lowest average runtimes for the 60-second runtime limit over all MAPF instances on each graph, as shown in Table 5.2.

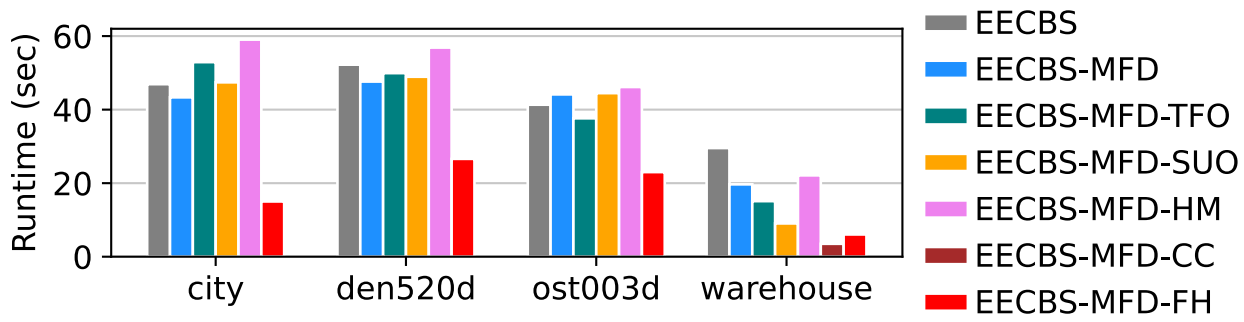


Figure 5.4: Average runtimes between the state-of-the-art guidance heuristics and our FH over all MAPF instances on each graph.

	city		den520d		ost003d		warehouse	
	avg.	std.	avg.	std.	avg.	std.	avg.	std.
EECBS	46.92	14.70	52.23	11.62	41.34	21.55	29.55	26.62
EECBS-MFD	43.44	16.23	47.63	15.47	44.14	21.12	19.68	22.67
EECBS-MFD-TFO	52.91	9.54	49.92	13.27	37.64	23.60	15.11	21.13
EECBS-MFD-SUO	47.42	14.14	48.94	14.37	44.48	21.06	9.05	11.20
EECBS-MFD-HM	59.04	**2.51**	56.87	**6.04**	46.15	**18.86**	22.12	21.44
EECBS-MFD-CC							**3.51**	**3.78**
EECBS-MFD-FH	**15.02**	14.52	**26.61**	21.54	**23.01**	24.60	6.05	7.94

Table 5.3: Average runtimes (in seconds), i.e., values from Figure 5.4, and standard deviations of EECBS, EECBS-MFD, and EECBS-MFD with different guidance heuristics (TFO, SUO, HM, CC, and FH) over all MAPF instances on each graph. Since EECBS-MFD-CC is only applicable in the `warehouse` graph, its average runtimes and standard deviations on other graphs are blank on other graphs. Numbers in bold indicate the lowest value along the column.

5.4.4.1 Comparison with the State-Of-The-Art Guidance Heuristics

Figure 5.3 shows the success rates of EECBS, EECBS-MFD, and EECBS-MFD with different guidance heuristics for the 60-second and 120-second runtime limits, respectively. While the CC-based approach can only be applied to the Kiva-like `warehouse` graph, FH achieves the same 1.00 success rate and can be applied to graphs beyond the `warehouse` graph. Also, EECBS-MFD-FH outperforms EECBS-MFD with all state-of-the-art guidance heuristics for large-scale MAPF instances on graphs other than `warehouse`. Since EECBS-MFD-CC is limited to the `warehouse` graph, we ignore it in Figure 5.4 and Table 5.3. Figure 5.4 shows the average runtimes of EECBS-MFD with different guidance heuristics over all MAPF instances on

	$T = 60$ (sec)			$T = 120$ (sec)		
	EECBS	MFD	MFD-FH	EECBS	MFD	MFD-FH
`city`	0.520	0.616	0.984	0.872	0.952	1.000
`den520d`	0.400	0.480	0.824	0.752	0.896	0.968
`ost003d`	0.504	0.416	0.728	0.656	0.648	0.792
`warehouse`	0.608	0.784	1.000	0.688	0.816	1.000
Total	0.508	0.574	0.884	0.742	0.828	0.940

Table 5.4: Success rates of EECBS, EECBS-MFD, and EECBS-MFD-FH for the 60-second and 120-second runtime limits, respectively, and the bounded-suboptimality factor $w = 1.10$ over all MAPF instances on each graph. For each runtime limit T, the columns "EECBS" contain the success rates of EECBS, the columns "MFD" contain the success rates of EECBS-MFD, the columns "MFD-FH" contain the success rates of EECBS-MFD-FH, and the row "Total" contains the success rates over all MAPF instances.

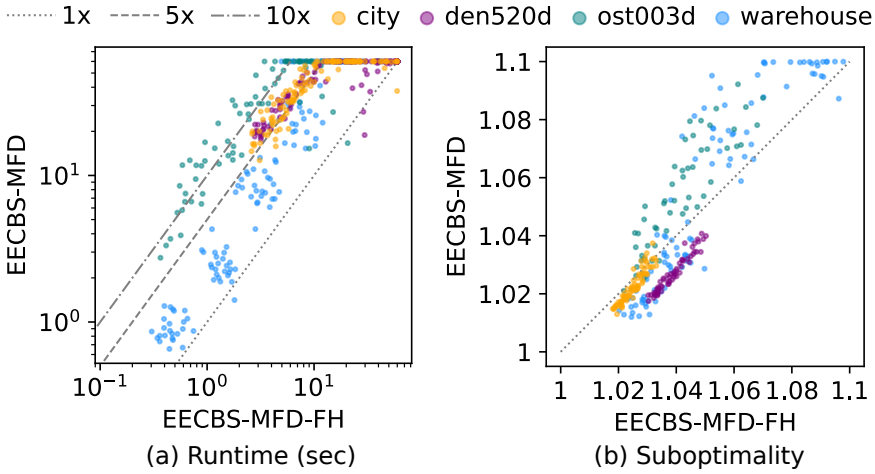


Figure 5.5: (a) Runtimes (in seconds) of EECBS-MFD and EECBS-MFD-FH for the bounded-suboptimality factor $w = 1.10$ over all MAPF instances on all graphs. The x- and y-coordinates of each dot represent the runtimes of EECBS-MFD-FH and EECBS-MFD, respectively. All the x- and y-coordinates are on logarithmic scales. (b) Empirical suboptimalities of EECBS-MFD and EECBS-MFD-FH for the bounded-suboptimality factor $w = 1.10$ over all MAPF instances that are successfully solved by both MAPF algorithms within the 60-second runtime limit on all graphs. The x- and y-coordinates of each dot represent the empirical suboptimalities of EECBS-MFD-FH and EECBS-MFD, respectively. Dots in yellow, purple, green, and blue represent the MAPF instances on the `city`, `den520d`, `ost003d`, and `warehouse` graphs, respectively.

each graph. Table 5.3 shows the average values and the standard deviations of runtimes of EECBS-MFD with different guidance heuristics over all MAPF instances on each graph.

Table 5.4 shows the success rates of EECBS, EECBS-MFD, and EECBS-MFD-FH for the 60-second and 120-second runtime limits, respectively, and the bounded-suboptimality factor $w = 1.10$ over all MAPF instances on each graph. EECBS-MFD-FH has higher success rates than EECBS and EECBS-MFD over

all MAPF instances on each graph for both 60-second and 120-second runtime limits and the bounded-suboptimality factor $w = 1.10$. In particular, while EECBS and EECBS-MFD have the success rates of 0.508 and 0.574, respectively, for the 60-second runtime limit over all MAPF instances, EECBS-MFD-FH has the success rate of 0.884. This indicates that EECBS-MFD-FH finds bounded-suboptimal solutions more quickly than EECBS and EECBS-MFD. Figure 5.5 (a) shows the runtimes of EECBS-MFD and EECBS-MFD-FH. Among the 500 MAPF instances over all graphs, there are 437 ones where EECBS-MFD has higher runtimes than EECBS-MFD-FH. Among these 437 MAPF instances, there are 172 ones where EECBS-MFD has runtimes that are 5 times higher than those of EECBS-MFD-FH. Among these 172 MAPF instances, there are 34 ones where EECBS-MFD has runtimes that are 10 times higher than those of EECBS-MFD-FH. Figure 5.5 (b) shows the empirical suboptimalities of the solutions from EECBS-MFD and EECBS-MFD-FH over all MAPF instances that are successfully solved by both MAPF algorithms within the 60-second runtime limit. The solutions of EECBS-MFD-FH have lower empirical suboptimalities for MAPF instances on the `ost003d` and `warehouse` graphs and achieve similar empirical suboptimalities as for those on the `city` and `den520d` graphs. Thus, EECBS-MFD-FH has significantly lower runtimes than EECBS-MFD while still finding bounded-suboptimal solutions with low empirical suboptimalities.

5.4.4.2 Comparison with the State-Of-The-Art Suboptimal MAPF Algorithm

Lazy Constraints Addition Search for MAPF (LaCAM) [50] is the state-of-the-art suboptimal MAPF algorithm. Since EECBS-MFD-FH targets bounded-suboptimal MAPF, we omit the anytime version of LaCAM [49] but use the version called LaCAM3 that incorporates the state-of-the-art enhancements [48]. LaCAM finds solutions within the 60-second runtime limit for all MAPF instances on each graph. Figure 5.6 shows the runtimes and empirical suboptimalities of LaCAM and EECBS-MFD-FH. Since LaCAM does not maintain a lower bound on an optimal solution during the search, we set the lower bound to the SOC of

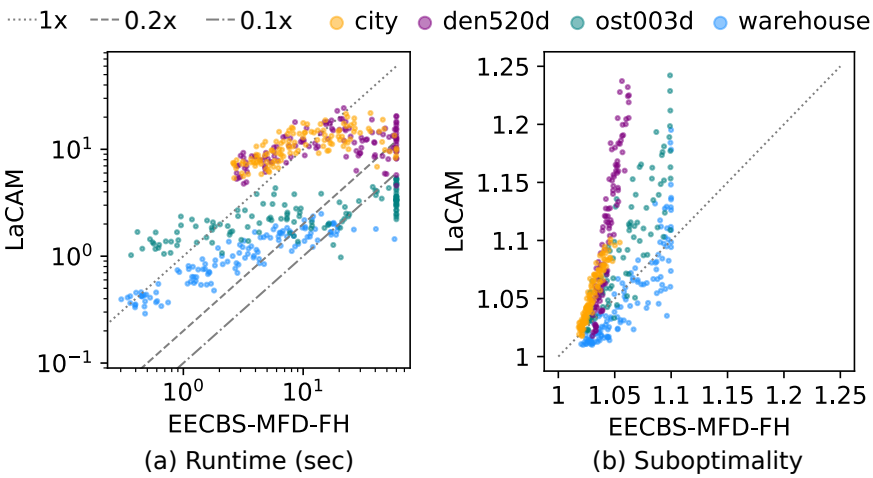


Figure 5.6: (a) Runtimes (in seconds) of LaCAM and EECBS-MFD-FH for the bounded-suboptimality factor $w = 1.10$ over all MAPF instances on all graphs. The x- and y-coordinates of each dot represent the runtimes of EECBS-MFD-FH and LaCAM, respectively. All the x- and y-coordinates are on logarithmic scales. (b) Empirical suboptimalities of LaCAM and EECBS-MFD-FH for the bounded-suboptimality factor $w = 1.10$ over all MAPF instances that are successfully solved by both MAPF algorithms within the 60-second runtime limit. The x- and y-coordinates of each dot represent the empirical suboptimalities of EECBS-MFD-FH and LaCAM, respectively. Dots in yellow, purple, green, and blue represent the MAPF instances on the `city`, `den520d`, `ost003d`, and `warehouse` graphs, respectively.

minimum-cost paths for each agent when computing its empirical suboptimalities. Although LaCAM typically finds solutions faster than EECBS-MFD-FH, it does not provide any guarantee on the solution quality and thus typically finds solutions with higher empirical suboptimalities than EECBS-MFD-FH, although it has lower runtimes and empirical suboptimalities than EECBS-MFD-FH on the `warehouse` graph. On the other hand, EECBS-MFD-FH can find solutions efficiently while guaranteeing their bounded suboptimalities. Among the 500 MAPF instances over all graphs, there are 326 ones where EECBS-MFD-FH has lower empirical suboptimalities than LaCAM, 166 ones where EECBS-MFD-FH has lower runtimes than LaCAM, and 146 ones where EECBS-MFD-FH has lower runtimes and empirical suboptimalities than LaCAM. Also, as shown in Figure 1.4, EECBS-MFD-FH has higher w-success rates than LaCAM on large-scale MAPF instances, such as those with 1,800 and 2,000 agents on the `den520d` graph. These results show that EECBS-MFD-FH is competitive with LaCAM in terms of empirical suboptimalities of its solutions.

	$\lvert\mathcal{X}_0\rvert$	$\lvert\mathcal{X}_0^l\rvert$	Searches	T'	T_0
EECBS	1.70	1.53	2.52	0	8.43
EECBS-MFD	1.71	1.53	1.79	0	7.35
EECBS-MFD-TFO	1.70	1.54	1.87	15.78	3.20
EECBS-MFD-SUO	1.64	1.52	1.66	0	7.79
EECBS-MFD-HM	1.70	1.52	1.80	18.51	5.82
EECBS-MFD-FH	0.16	0.07	0.39	6.45	3.07

Table 5.5: Numbers (in thousands) of conflicts and target conflicts of the root CT node, numbers of searches of the low-level Focal Search, and runtimes (in seconds) of generating guidance heuristics and generating root CT node, averaging over all MAPF instances. The column "$|\mathcal{X}_0|$" contains the average numbers (in thousands) of conflicts among paths in the root CT node, the column "$|\mathcal{X}_0^l|$" contains the average numbers (in thousands) of target conflicts among paths in the root CT node, the column "Searches" contains the average numbers (in thousands) of searches of the low-level Focal Search, the column "T'" contains the average runtimes of generating guidance heuristics, and the column "T_0" contains the average runtimes of generating root CT node. Since EECBS, EECBS-MFD, and EECBS-SUO do not generate a guidance heuristic before the search, their $T' = 0$.

		$k_{\max} = 0.25k$	$k_{\max} = 0.5k$	$k_{\max} = 0.75k$	$k_{\max} = k$
`city`	$k = 2{,}000$	0.29	1.80	4.11	44.75
`den520d`	$k = 2{,}000$	0.19	1.49	3.68	52.17
`ost003d`	$k = 800$	0.04	0.37	17.11	22.88
`warehouse`	$k = 700$	0.19	0.67	0.90	10.46

Table 5.6: Average runtimes (in seconds) of calculating FH for different values of path-found thresholds $k_{\max}$ over 25 MAPF instances with k agents on each graph.

5.4.5 Empirical Insights

As shown in Table 5.5, EECBS-MFD-FH finds paths for the root CT node with fewer conflicts $|\mathcal{X}_0|$ than the other guidance heuristics. Furthermore, in terms of Strategy (S1), since FBGF uses target obstacles to simulate paths for FH, EECBS-MFD-FH finds paths for the root CT node with fewer target conflicts $|\mathcal{X}_0^l|$ than the other guidance heuristics. The small numbers of conflicts and target conflicts result in fewer low-level searches. FBGF also has a lower runtime T' for calculating the guidance heuristics than the other approaches. In terms of Strategy (S2), since EECBS-MFD-FH reuses the simulated paths from FBGF as the paths for the root CT node, it has a lower runtime T_0 for generating the root CT node than the other approaches. In terms of strategy (S3), as shown in Table 5.6, the path-found threshold $k_{\max}$ can

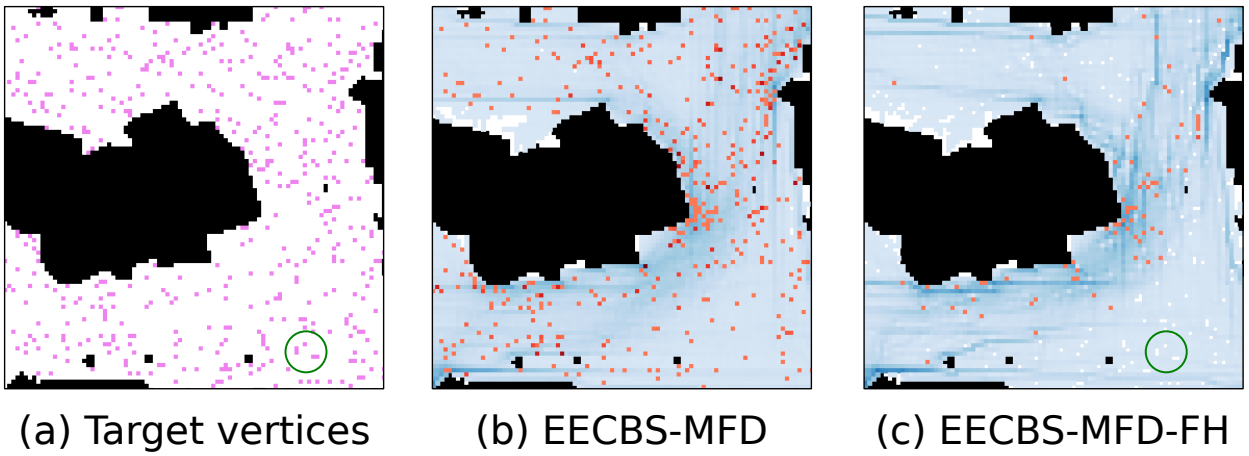


Figure 5.7: Center part of a MAPF instance with 2,000 agents on the `den520d` graph. (a) Target vertices. (b) Heatmap of the paths (in blue) and conflicts (in red) of the root CT node of EECBS-MFD. (c) Heatmap of the paths (in blue) and conflicts (in red) of the root CT node of EECBS-MFD-FH. For (b) and (c), the darker the color, the higher the value. The green circles show examples of avoiding traversing target vertices when the agents are guided by FH.

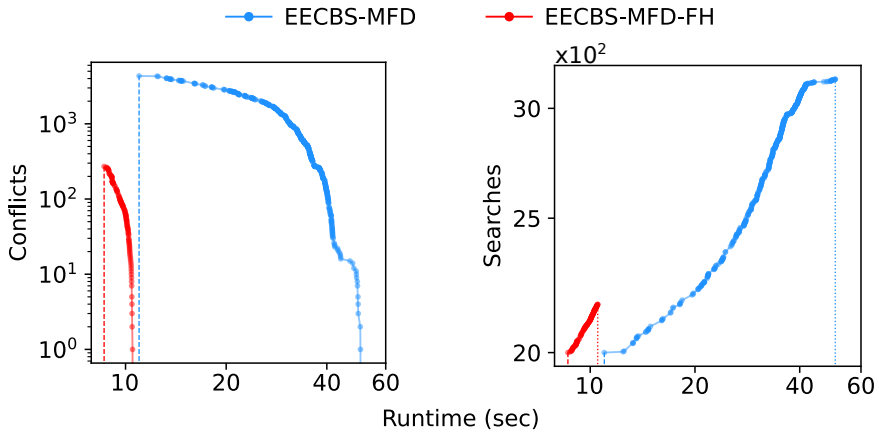


Figure 5.8: Number of conflicts among paths in each expanded CT node at the time (in seconds) when it is expanded by EECBS-MFD and EECBS-MFD-FH, and cumulative number of low-level searches of EECBS-MFD and EECBS-MFD-FH versus the runtime. Dashed lines indicate the time when EECBS-MFD or, respectively, EECBS-MFD-FH, begins to expand its root CT node. Dotted lines indicate the time when EECBS-MFD or, respectively, EECBS-MFD-FH, finds a solution. For EECBS-MFD-FH, the runtime and the number of low-level searches used to generate FH are included. All the x- and y-coordinates are on logarithmic scales.

significantly increase the runtime of FBGF. For example, the runtime of calculating FH can use most of the 60-second runtime limit for MAPF instances with 2,000 agents on the `city` and `den520d` graphs.

5.4.6 Case Study

We run EECBS-MFD and EECBS-MFD-FH with the bounded-suboptimality factor $w = 1.10$ on a MAPF instance with 2,000 agents on the `den520d` graph. Figure 5.7 shows (a) the target vertices and the heat map of the paths in the root CT node of (b) EECBS-MFD and (c) EECBS-MFD-FH. Without the guidance from FH, EECBS-MFD finds paths with many target conflicts, as shown in Figure 5.7 (b). With the guidance from FH, EECBS-MFD-FH finds paths with fewer target conflicts than EECBS-MFD, as shown in Figure 5.7 (c). Also, since FBGF uses Strategy (S1) to generate FH, the paths in the root CT node of EECBS-MFD-FH traverse fewer target vertices than in EECBS-MFD, resulting in white cells in their heatmap. An example of this is shown in the green circle in Figure 5.7 (c). The guidance of FH reduces the number of conflicts from 4,838 to 287, where the number of target conflicts is reduced from 4,653 to 179. Figure 5.8 shows the number of conflicts among paths in each expanded CT node and the number of low-level searches of EECBS-MFD and EECBS-MFD-FH as a function of runtime when solving this MAPF instance. With the guidance from FH, EECBS-MFD-FH starts with a root CT node with fewer conflicts and finds a solution faster than EECBS-MFD. Also, EECBS-MFD-FH runs fewer low-level searches than EECBS-MFD and finds a bounded-suboptimal solution even before EECBS-MFD starts expanding its root CT node.

5.4.7 Strategy Evaluation

We evaluate Strategies (S1), (S2), and (S3) by comparing them against other FBGF variants. To evaluate the effectiveness of Strategy (S1) in achieving Feature (F1), we implement an FBGF variant that simulates paths without target obstacles, while Strategies (S2) and (S3) remain unchanged. We then compare EECBS-MFD-FH guided by the original FH with target obstacles to EECBS-MFD-FH guided by the new FH without target obstacles. As shown in Figure 5.9, the original FH with target obstacles achieves higher success rates

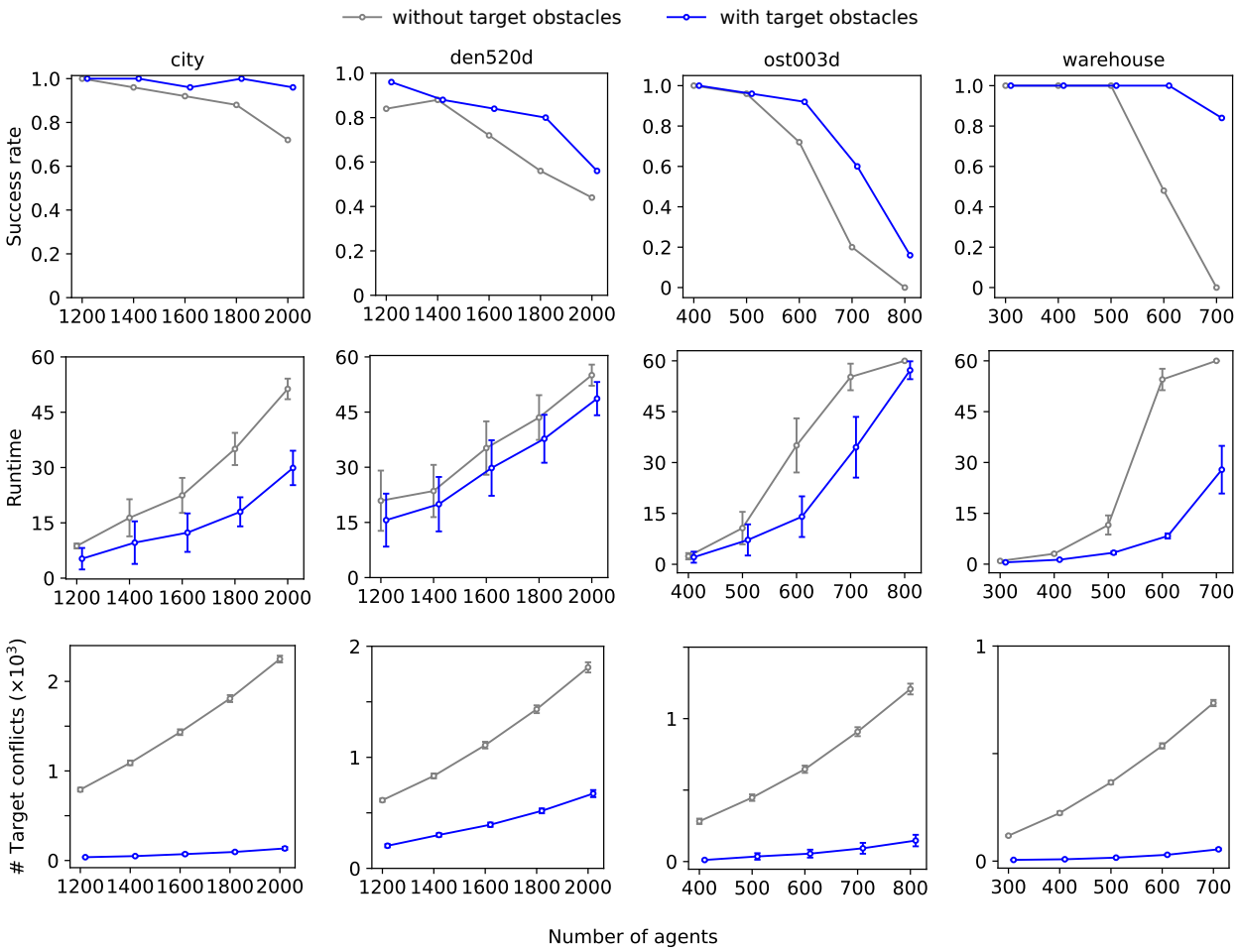


Figure 5.9: Success rates, average runtimes, and average numbers of target conflicts of EECBS-MFD-FH with and without target obstacles (to evaluate Strategy (S1)) over all MAPF instances with each number of agents on each graph. The vertical bars indicate the 95% confidence intervals.

and lower average runtimes than the new FH without target obstacles. Additionally, when EECBS-MFD-FH generates the root CT node, the original FH with target obstacles yields fewer target conflicts than the new FH without target obstacles, allowing EECBS-MFD-FH to solve MAPF instances more quickly.

To evaluate the effectiveness of Strategy (S2) in achieving Feature (F2), we implement an FBGF variant that uses the low-level Focal Search to simulate paths one by one, while Strategies (S1) and (S3) remain unchanged. We then compare EECBS-MFD-FH guided by the original FH with the two-stage low-level (F)BCS to EECBS-MFD-FH guided by the new FH with the low-level Focal Search. Due to the target obstacles in Strategy (S1), the simulated paths from the low-level Focal Search cannot be reused as paths

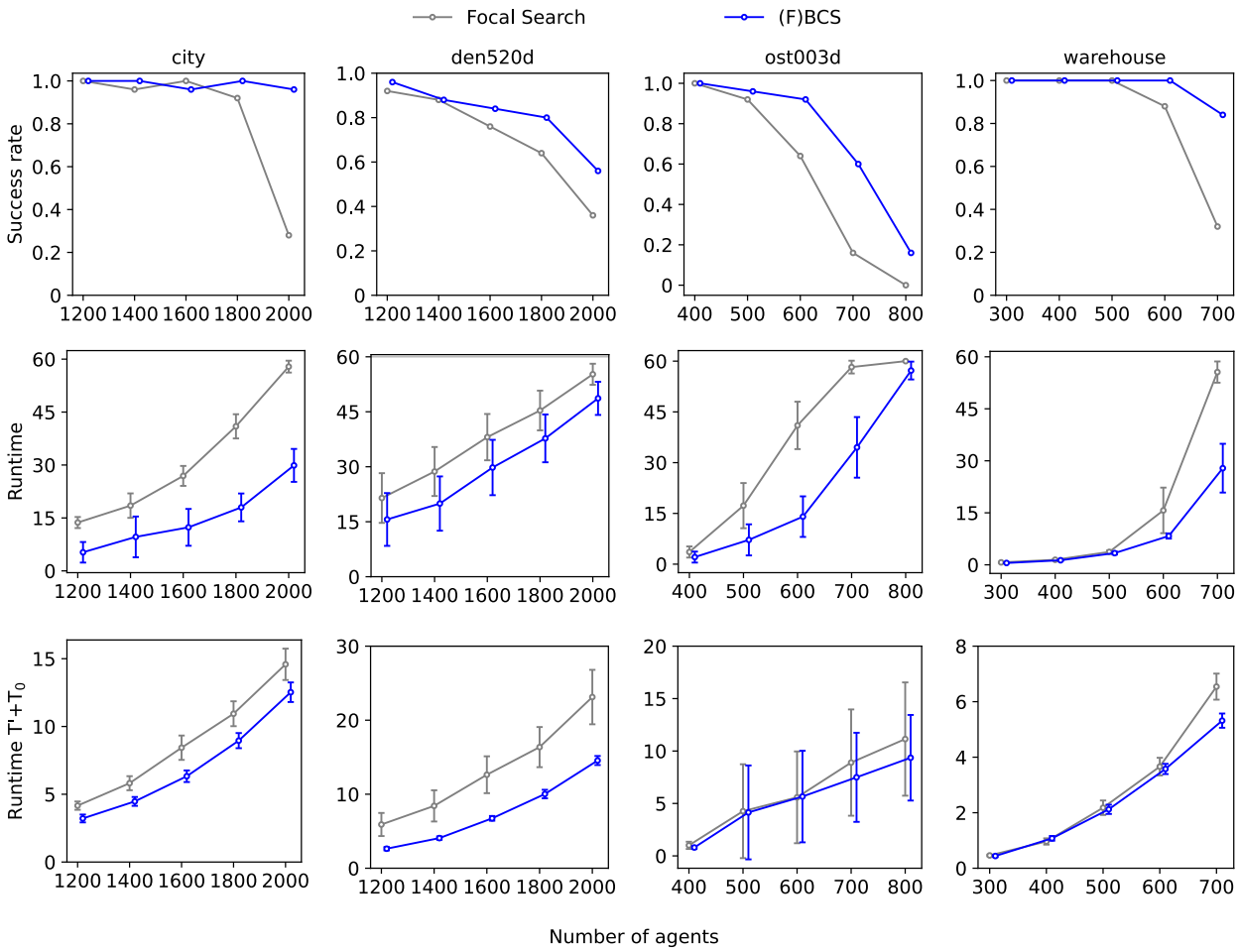


Figure 5.10: Success rates, average runtimes, and average sums of runtimes $T' + T_0$ of calculating FH (T') and generating the root CT node (T_0) of EECBS-MFD-FH with FBGF using the low-level Focal Search and the two-stage low-level (F)BCS (to evaluate Strategy (S2)) over all MAPF instances with each number of agents on each graph. The vertical bars indicate the 95% confidence intervals.

when EECBS-MFD generates the root CT node (see Section 5.3.3). However, by using the two-stage low-level (F)BCS, EECBS-MFD-FH can reuse the simulated paths as the paths of its root CT node, thereby reducing the runtime T_0 for generating the root CT node. As shown in Figure 5.10, EECBS-MFD-FH guided by the original FH that uses the two-stage low-level (F)BCS achieves higher success rates and lower average runtimes than EECBS-MFD-FH guided by the new FH that uses the low-level Focal Search. We also evaluate the average sum of the runtimes for calculating FH (denoted as T') and generating the root CT node (denoted as T_0), which corresponds to the time point when EECBS-MFD starts resolving

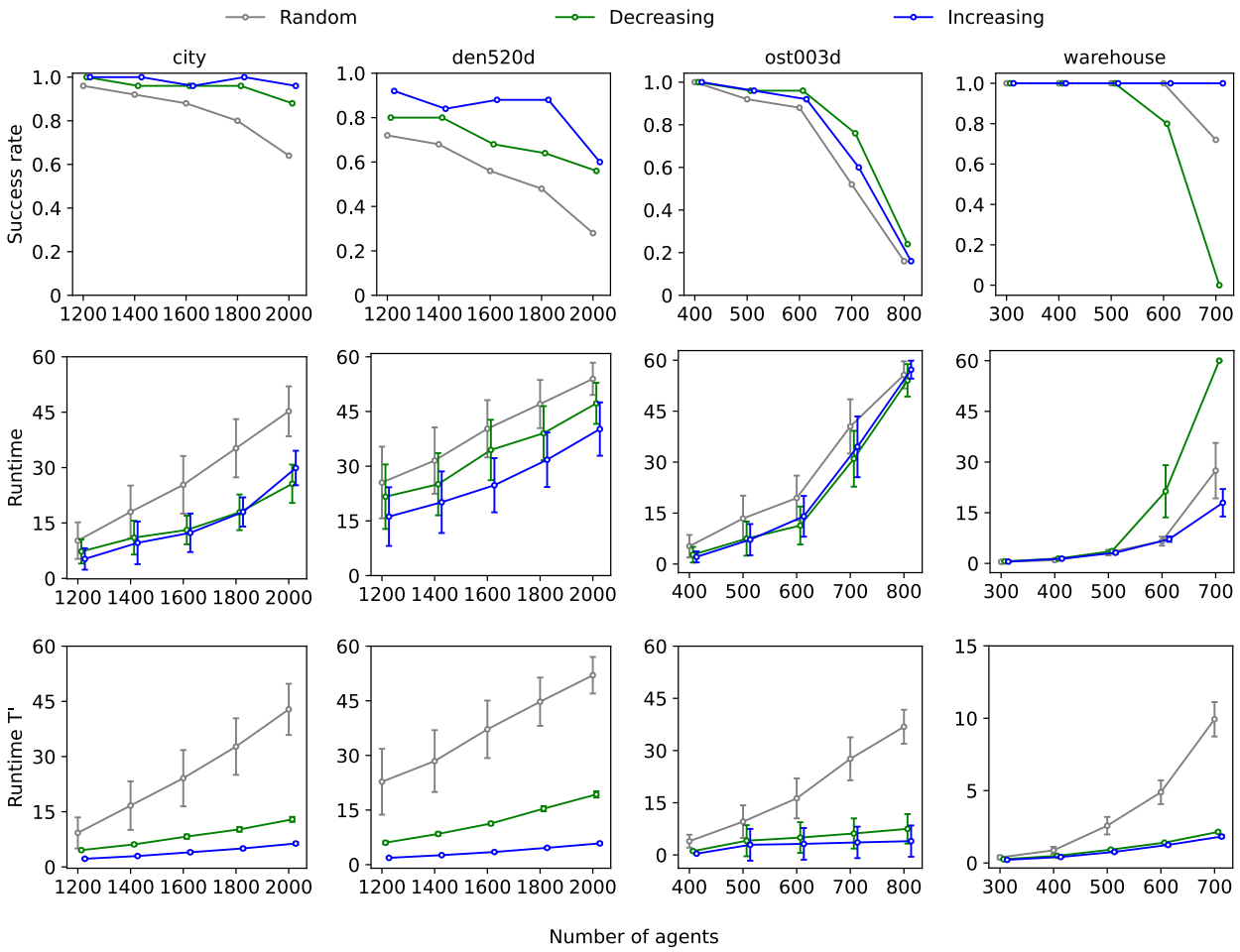


Figure 5.11: Success rates, average runtimes, and average runtimes T' of calculating FH with random, decreasing, and increasing orders of agents (to evaluate Strategy (S3)) over 25 MAPF instances with each number of agents on each graph. The vertical bars indicate the 95% confidence intervals.

conflicts and expanding CT nodes. The two-stage low-level (F)BCS can reuse the simulated path, allowing EECBS-MFD-FH to resolve conflicts earlier and thus solve MAPF instances more quickly.

To evaluate the effectiveness of Strategy (S3), we implement two FBGF variants that sort the agents in a decreasing order and a random order of the costs of their minimum-cost paths, i.e., the $h_i(s_i)$ values of agents a_i. As shown in Figure 5.11, especially on the `den520d` and `warehouse` graphs, EECBS-MFD-FH guided by the original FH that sorts agents in an increasing order typically achieves higher success rates and lower average runtimes than EECBS-MFD-FH guided by the new FHs that sort agents in decreasing and random orders (except for MAPF instances with 700 and 800 agents on the `ost003d` graph, where

the decreasing order slightly outperforms the other two). Also, the low-level (F)BCS typically finds paths faster for agents with lower costs of their minimum-cost paths. Thus, simulating $k_{\max}$ paths for agents sorted in increasing order of the cost of their minimum-cost paths yields the lowest average runtime T' for calculating FH compared to the other two variants, allowing more runtime for EECBS-MFD-FH. Interestingly, the random order results in the worst success rates and average runtimes of all orders (except for MAPF instances on the `warehouse` graph). It also has the highest average runtime T' for calculating FH and the widest 95% confidence intervals, indicating the uncertainty in its performance.

5.5 Summary

We presented the Flow-Based Guidance Framework (FBGF), a pre-processing technique that generates the Flow-Based Guidance Heuristic (FH) to guide the low-level Focal Search, thereby improving the efficiency of EECBS-MFD. Based on our strategies targeting bounded-suboptimal MAPF, FBGF simulates paths with target obstacles, uses a two-stage low-level (Flexible) Bounded-Cost Search to find simulated paths, and employs a threshold to reduce the number of simulated paths (accompanied by a sorting method for ordering the agents). Then, based on the simulated paths, FBGF calculates FH according to their flows. Our empirical evaluation shows that FBGF efficiently calculates FH and that FH effectively guides the low-level Focal Search to speed up EECBS-MFD. Our empirical evaluation showed that while EECBS and EECBS-MFD have the success rates of 0.508 and 0.574, respectively, for the 60-second runtime limit over all MAPF instances, EECBS-MFD with FH has the success rate of 0.884.

Algorithm 5.1 Flexible Bounded-Cost Search (FBCS)

procedure FBCS(s_i, l_i, P)
  ▷ *$P[j]$ is the simulated path of agent a_j (i.e., P is the simulated path list), so $P[i]$ is null.* ◁
  $T_i \leftarrow$ Compute the time horizon from paths P ▷ *See Section 2.4.2.*
  $\tau_i \leftarrow w \cdot h_i(s_i)$
  $J \leftarrow \{j \in [k] \mid P[j]$ is **not** null$\}$
  for $j \in J$ **do**
    $c_j \leftarrow$ cost of path $P[j]$
    $\tau_i \leftarrow \tau_i + (w \cdot h_j(s_j) - c_j)$
  Generate the root v-t node n_0 with state representation $(s_i, 0)$
  $g_i(n_0) \leftarrow 0$; $h_i(n_0) \leftarrow h_i(s_i)$; $f_i(n_0) \leftarrow g_i(n_0) + h_i(n_0)$; $x_i(n_0) \leftarrow 0$
  OPEN$_L$, CLOSED$_L \leftarrow$ empty list
  Insert v-t node n_0 into OPEN$_L$
  while OPEN$_L$ **not** empty **do**
    $\hat{n} \leftarrow$ v-t node in OPEN$_L$ with the minimum x_i-value
    Remove v-t node $\hat{n}$ from OPEN$_L$
    Insert v-t node $\hat{n}$ into CLOSED$_L$
    if IsTargetReached($\hat{n}$, l_i, $\emptyset$) **then**
      $p_i \leftarrow$ Extract the path by back-tracking v-t node $\hat{n}$
      return p_i
    neighbors $\leftarrow$ FindNeighbors($\hat{n}$)
    for $n = (v, t) \in$ *neighbors* **do**
      if $v \in \{l_j \mid j \in [k] \setminus \{i\}\}$ **or** $f_i(n) > \tau_i$ **then** ▷ *Use Strategies (S1) and (S2).*
        continue
      $n \leftarrow$ GenerateChildNode($\hat{n}$, v, t, P)
      $\bar{n} \leftarrow$ FindNode(n, $T_i(N)$, CLOSED$_L$, OPEN$_L$)
      if $\bar{n}$ is null **then**
        Insert v-t node n into OPEN$_L$
        continue
      if IsDominant(n, $\bar{n}$) **then**
        UpdatePriority(n, $\bar{n}$, CLOSED$_L$, OPEN$_L$)
  return No path

procedure UpdatePriority(n, $\bar{n}$, CLOSED$_L$, OPEN$_L$)
  $(v, t) \leftarrow$ state representation of v-t node n
  $(v, \bar{t}) \leftarrow$ state representation of v-t node $\bar{n}$
  $g_i(\bar{n}) \leftarrow g_i(n)$; $f_i(\bar{n}) \leftarrow f_i(n)$; $x_i(\bar{n}) \leftarrow x_i(n)$; $\bar{t} \leftarrow t$; *parent*$(\bar{n}) \leftarrow$ *parent*(n)
  if $\bar{n} \in$ CLOSED$_L$ **then**
    Remove v-t node $\bar{n}$ from CLOSED$_L$
    Insert v-t node $\bar{n}$ into OPEN$_L$
  else
    Update the priority of $\bar{n}$ in OPEN$_L$

Algorithm 5.2 Phase (P1) of the Flow-Based Guidance Framework (FBGF)

procedure RunPhaseP1(MAPF instance with k agents on graph $G = (V, E)$, $k_{\max}$)
  $k_{cnt} \leftarrow 0$
  $P \leftarrow$ empty list ▷ *The simulated path list*
  $A' \leftarrow$ Sort all k agents $\{a_i \mid i \in [k]\}$ in increasing order of their $h_i(s_i)$ ▷ *Use Strategy (S3)*
  for $i \in [k]$ **do** ▷ *Stage (I) starts*
    $a_{i'} \leftarrow A'[i]$ ▷ *Agent in the i-th order of A'*
    $p_{i'} \leftarrow$ BCS($s_{i'}, l_{i'}, P$)
    if $p_{i'}$ is found **then**
      $P[i'] \leftarrow p_{i'}$
      $k_{cnt} \leftarrow k_{cnt} + 1$
    if $k_{cnt} = k_{\max}$ **then** ▷ *Use Strategy (S3)*
      break
  for $i \in [k]$ **do** ▷ *Stage (II) starts*
    $a_{i'} \leftarrow A'[i]$ ▷ *Agent in the i-th order in A*
    if $P[i']$is **not** null **then** ▷ *A simulated path for $a_{i'}$ exists*
      continue
    $p_{i'} \leftarrow$ FBCS($s_{i'}, l_{i'}, P$) ▷ *Use Algorithm 5.1*
    if $p_{i'}$ is found **then**
      $P[i'] \leftarrow p_{i'}$
      $k_{cnt} \leftarrow k_{cnt} + 1$
    if $k_{cnt} = k_{\max}$ **then** ▷ *Use Strategy (S3)*
      break
  ▷ *Construct FGG $G_F = (V, E_F, W_F)$* ◁
  $E_F \leftarrow \emptyset$; $W_F \leftarrow \emptyset$
  for $\{u, v\} \in E$ **do**
    $e_{uv} \leftarrow$ directed edge from vertex u to vertex v
    $e_{vu} \leftarrow$ directed edge from vertex v to vertex u
    $E_F \leftarrow E_F \cup \{e_{uv}, e_{vu}\}$
  for $e_{uv} \in E_F$ **do**
    $\Phi_{uv} \leftarrow$ number of agents traversing the directed edge e_{uv} when they follow their simulated paths
  $\Phi_{\max} \leftarrow \max\{\Phi_{uv} \mid e_{uv} \in E_F\}$
  for $e_{uv} \in E_F$ **do**
    Compute w_{uv} via Equation 5.4
    Insert w_{uv} into W_F
  return $G_F = (V, E_F, W_F)$

Chapter 6

Conclusions and Future Directions

Multi-Agent Path Finding (MAPF) is the problem of finding a list of collision-free paths, one for each agent, that move the agents from their start locations to their target locations in a shared environment. The objective is to minimize a metric such as the sum of travel times or, equivalently, the sum of (path) costs (SOC). MAPF serves as the foundation for coordinating fleets of agents and has numerous real-world applications, including for automated warehouses, traffic management, and even controlling animated characters in video games. In this dissertation, we studied MAPF on four-neighbor grid graphs where time is discretized into timesteps.

Since minimizing the SOC is computationally intractable, researchers have developed a spectrum of MAPF algorithms that strike a balance between scalability and guarantees on the solution quality. One direction is to use (unbounded) suboptimal MAPF algorithms, which quickly find a solution but typically yield poor solution quality. Another direction is to use anytime MAPF algorithms, which first run an (unbounded) suboptimal MAPF algorithm and then gradually improve its solution quality within a finite runtime limit. However, neither suboptimal MAPF algorithms nor anytime MAPF algorithms provide guarantees on the solution quality. A third direction is to use bounded-suboptimal MAPF algorithms, which find solutions with SOCs that are at most a factor w larger than the SOC of an optimal solution, where

w is a user-specified bounded-suboptimality factor. Compared to optimal MAPF algorithms, bounded-suboptimal MAPF algorithms offer a promising direction for achieving scalability. Compared to suboptimal and anytime MAPF algorithms, bounded-suboptimal MAPF algorithms provide guarantees on the solution quality. The state-of-the-art bounded-suboptimal MAPF algorithm is Explicit Estimation Conflict-Based Search (EECBS), which guarantees to find a bounded-suboptimal solution to a MAPF instance if a solution exists. Thus, we improved the efficiency of EECBS in this dissertation.

EECBS performs a heuristic search on a Constraint Tree (CT), where each CT node contains a list of paths, one for each agent, and a list of constraint sets used to resolve collisions, one for each agent. EECBS guarantees to find a bounded-suboptimal solution to a MAPF instance, if a solution exists, by requiring each path in each CT node to be individually bounded-suboptimal, i.e., the cost of the path is at most the bounded-suboptimality factor w larger than the lower bound on the cost of the minimum-cost path that satisfies the constraints. To find an individually bounded-suboptimal path, EECBS uses Focal Search on its low level. Focal Search returns a lower bound on the cost of a minimum-cost path that satisfies the constraints and a path whose cost is at most a threshold, which is w times the lower bound, resulting in the path being individually bounded-suboptimal. By requiring each path to be individually bounded-suboptimal, EECBS ensures that the SOC of each agent's path is at most the bounded-suboptimality factor w larger than the sum of their lower bounds, thereby ensuring that the SOC of each agent's path is at most the bounded-suboptimality factor w larger than the SOC of each agent's minimum-cost path that satisfies the constraints.

However, since the definition of bounded-suboptimal MAPF is based on the SOC being at most the bounded-suboptimality factor w larger than the SOC of an optimal solution, it is unnecessary to require each path to be individually bounded-suboptimal in each CT node. Thus, we proposed Flex Distribution, which relaxes the individually bounded-suboptimality of each path while still guaranteeing that the SOC of the paths in each CT node remains at most the bounded-suboptimality factor w larger than its sum of

lower bounds, thereby guaranteeing EECBS to find a bounded-suboptimal solution to a MAPF instance if a solution exists. We hypothesized that one can improve the efficiency of EECBS via Flex Distribution. In Chapter 3, we defined the flex of a path in a CT node as the difference between its threshold and its cost. To find a path for an agent that satisfies the constraints in a CT node, we proposed Greedy Flex Distribution (GFD), which greedily adds all flex from the other agents' paths to the threshold. Theoretically, we proved that EECBS with GFD is guaranteed to find a bounded-suboptimal solution to a MAPF instance if a solution exists. Empirically, we showed that GFD improves the efficiency of EECBS for large-scale MAPF instances, especially for small bounded-suboptimality factors.

In Chapter 4, we further proposed additional mechanisms for determining how much flex to distribute. Given a CT node, one mechanism uses the number of collisions an agent will encounter if it follows its path to determine how much flex to distribute. Another mechanism uses the extra path cost needed for an agent to satisfy its constraints to determine how much flex to distribute. A third mechanism, called Mix-Strategy Flex Distribution (MFD), combines the previous two mechanisms in a hierarchical framework. Theoretically, we proved that EECBS with MFD is guaranteed to find a bounded-suboptimal solution to a MAPF instance if a solution exists. Empirically, we showed that sophisticated flex distribution mechanisms, such as MFD, improve the efficiency of EECBS for large-scale MAPF instances beyond the improvement achieved by GFD.

In Chapter 5, building on top of EECBS with MFD, we proposed the Flow-Based Guidance Framework (FBGF) that further improves the efficiency of EECBS with MFD by using Flex Distribution to generate a heuristic that guides the low-level searches for paths for the agents. It is a pre-processing approach that generates the Flow-Based Guidance Heuristic (FH) for this purpose. Given a MAPF instance, FBGF simulates a set of paths and minimizes the conflicts among them. Then, FBGF counts the number of agents traversing each edge in each direction when they follow their simulated paths as the flow and constructs a weighted directed graph by assigning a lower cost to a directed edge with a higher flow. The FH on each

vertex for an agent is the cost of the minimum-cost path from that vertex to the agent's target vertex. To calculate FH that is effective in improving the efficiency of EECBS with MFD, we proposed strategies that a simulated path of an agent should avoid the target vertices of the other agents, the SOC of the simulated paths should be at most the bounded-suboptimality factor w larger than the SOC of the minimum-cost paths, and the number of simulated paths is at most a user-specified threshold $k_{\max}$. Theoretically, we proved that EECBS with MFD and FH is still guaranteed to find a bounded-suboptimal solution to a MAPF instance if a solution exists. Empirically, we showed that FBGF can efficiently calculate FH, which improves the efficiency of EECBS with MFD for large-scale MAPF instances.

Since we have shown that Flex Distribution can improve the efficiency of EECBS, this dissertation serves as a starting point for further exploration. Possible future research directions include, but are not limited to:

- **Developing learning-based approaches for Flex Distribution.** Our mechanisms for determining the amount of distributed flex rely on heuristics, which can be imprecise. One possible research direction is to rely on learning-based approaches to determine the amount of distributed flex. One bottleneck for EECBS or, more generally, bounded-suboptimal MAPF algorithms is to prove that the solutions they find are bounded-suboptimal. It would help to have a black-box approach available as a pre-processing technique that outputs a good estimate of the cost of a path before the search for the path. One possible approach is imitation learning. We can collect MAPF instances as data and their solutions as labels. Then, we can train a machine learning model that maps features of a MAPF instance to the cost of each path in the solution. Ideally, the machine learning model can handle different numbers of agents and different graphs. Such a machine learning model can then be deployed as a pre-processing technique that allows EECBS to improve its Flex Distribution.

- **Extending Flex Distribution to other bounded-suboptimal MAPF algorithms.** Since Flex Distribution is a technique that relaxes the individual bounded-suboptimality of paths, we might be

able to extend it to other bounded-suboptimal MAPF algorithms. An example is Large-Neighborhood Search for (suboptimal) MAPF, called MAPF-LNS2 [28]. While MAPF-LNS [27] is an anytime MAPF algorithm that reduces the SOC over time, MAPF-LNS2 is a suboptimal MAPF algorithm that reduces the number of collisions over time. At each iteration, MAPF-LNS2 selects a subset of agents and updates their paths to reduce the number of collisions that the selected agents are involved in. We can transform MAPF-LNS2 to a bounded-suboptimal MAPF algorithm by utilizing Algorithm 5.2 in Chapter 5. That is, we can use Flexible Bounded-Cost Search without target obstacles to iteratively find a path for each selected agent, which generates a list of paths whose SOC is guaranteed bounded-suboptimal. We could then leverage parallel processing. For example, we could use the multi-threading framework [7] to run an optimal MAPF algorithm, such as Conflict-Based Search, in one thread to improve the lower bound on the SOC of an optimal solution, while running our modified MAPF-LNS2 in the other threads to resolve collisions.

- **Applying Flex Distribution to multi-objective/constraint scenarios.** MAPF is an abstraction of multi-agent systems where multiple agents move without collisions in a discretized space and time. The objective of MAPF is to minimize the SOC, but real-world applications often require multiple objectives and constraints, such as kinodynamic constraints, battery levels, and even environmental elevation. Thus, one future direction is to deploy Flex Distribution to multi-objective MAPF [54]. For example, in an environment with various elevation levels, in addition to minimizing the number of collisions, EECBS may need to consider minimizing the sum of absolute values of the elevation differences (SOED). Suppose that there is a path for an agent with a short travel time but a high SOED, and there is another path with a long travel time but a low SOED. We could use EECBS with Flex Distribution to find the path with a long travel time but a low SOED, compared to the one found without Flex Distribution, while guaranteeing the SOC of the solution is bounded-suboptimal.

Bibliography

[1] Zain Alabedeen Ali and Konstantin Yakovlev. "Improved Anonymous Multi-Agent Path Finding Algorithm". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2024, pp. 17291–17298.

[2] Dor Atzmon, Roni Stern, Ariel Felner, Glenn Wagner, Roman Bartak, and Neng-Fa Zhou. "Robust Multi-Agent Path Finding". In: *Proceedings of the International Symposium on Combinatorial Search (SoCS)*. 2021, pp. 2–9.

[3] Jacopo Banfi, Nicola Basilico, and Francesco Amigoni. "Intractability of Time-Optimal Multirobot Path Planning on 2D Grid Graphs with Holes". In: *IEEE Robotics and Automation Letters* 2.4 (2017), pp. 1941–1947.

[4] Eli Boyarski, Shao-Hung Chan, Dor Atzmon, Ariel Felner, and Sven Koenig. "On Merging Agents in Multi-Agent Pathfinding Algorithms". In: *Proceedings of the International Symposium on Combinatorial Search (SoCS)*. 2022, pp. 11–19.

[5] Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, David Tolpin, Oded Betzalel, and Solomon Eyal Shimony. "ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2015, pp. 740–746.

[6] Shao-Hung Chan, Zhe Chen, Teng Guo, Han Zhang, Yue Zhang, Daniel Harabor, Sven Koenig, Cathy Wu, and Jingjin Yu. "The League of Robot Runners: Competition Goals, Designs, and Implementation". In: *Proceedings of the ICAPS 2024 System's Demonstration Track*. 2024.

[7] Shao-Hung Chan, Zhe Chen, Dian-Lun Lin, Yue Zhang, Daniel Harabor, Tsung-Wei Huang, Sven Koenig, and Thomy Phan. "Operation Parallelism in Large Neighbourhood Search for Anytime Multi-Agent Path Finding". In: *Proceedings of the International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*. 2025, in print.

[8] Shao-Hung Chan, Jiaoyang Li, Graeme Gange, Daniel Harabor, Peter J. Stuckey, and Sven Koenig. "ECBS with Flex Distribution for Bounded-Suboptimal Multi-Agent Path Finding". In: *Proceedings of the International Symposium on Combinatorial Search (SoCS)*. 2022, pp. 159–161.

[9] Shao-Hung Chan, Jiaoyang Li, Graeme Gange, Daniel Harabor, Peter J. Stuckey, and Sven Koenig. "Flex Distribution for Bounded-Suboptimal Multi-Agent Path Finding". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2022, pp. 9313–9322.

[10] Shao-Hung Chan, Roni Stern, Ariel Felner, and Sven Koenig. "Greedy Priority-Based Search for Suboptimal Multi-Agent Path Finding". In: *Proceedings of the International Symposium on Combinatorial Search (SoCS)*. 2023, pp. 11–19.

[11] Jingkai Chen, Jiaoyang Li, Yijiang Huang, Caelan Garrett, Dawei Sun, Chuchu Fan, Andreas Hofmann, Caitlin Mueller, Sven Koenig, and Brian C. Williams. "Cooperative Task and Motion Planning for Multi-Arm Assembly Systems". In: *arXiv preprint arXiv:2203.02475*. 2022.

[12] Zhe Chen, Daniel Harabor, Jiaoyang Li, and Peter J. Stuckey. "Traffic Flow Optimisation for Lifelong Multi-Agent Path Finding". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2024, pp. 20674–20682.

[13] Zhe Chen, Jiaoyang Li, Daniel Harabor, Peter J. Stuckey, and Sven Koenig. "Multi-Train Path Finding Revisited". In: *Proceedings of the International Symposium on Combinatorial Search (SoCS)*. 2022, pp. 38–46.

[14] Liron Cohen, Tansel Uras, T. K. Satish Kumar, and Sven Koenig. "Optimal and Bounded-Suboptimal Multi-Agent Motion Planning". In: *Proceedings of the International Symposium on Combinatorial Search (SoCS)*. 2021, pp. 44–51.

[15] Liron Cohen, Tansel Uras, T. K. Satish Kumar, Hong Xu, Nora Ayanian, and Sven Koenig. "Improved Solvers for Bounded-Suboptimal Multi-Agent Path Finding". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2016, pp. 3067–3074.

[16] Caitlin Dawson. "Could This Nearly Invincible Drone Be the Future of Disaster Relief?" In: *USC Viterbi School News* (2020). URL: https://viterbischool.usc.edu/news/2020/02/could-this-nearly-invincible-drone-be-the-future-of-disaster-relief/.

[17] Scott Dresser. "Amazon Launches a New AI Foundation Model to Power its Robotic Fleet and Deploys its 1 Millionth Robot". In: *About Amazon, News* (2025). URL: https://www.aboutamazon.com/news/operations/amazon-million-robots-ai-foundation-model.

[18] Gilad Fine, Dor Atzmon, and Noa Agmon. "Anonymous Multi-Agent Path Finding with Individual Deadlines". In: *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 2023, pp. 869–877.

[19] Teng Guo and Jingjin Yu. "Toward Efficient Physical and Algorithmic Design of Automated Garages". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 1364–1370.

[20] Shuai D. Han and Jingjin Yu. "Optimizing Space Utilization for More Effective Multi-Robot Path Planning". In: *Proceedings of the International Conference on Robotics and Automation (ICRA)*. 2021, pp. 10709–10715.

[21] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[22] Florence Ho, Ana Salta, Ruben Geraldes, Artur Goncalves, Marc Cavazza, and Helmut Prendinger. “Multi-Agent Path Finding for UAV Traffic Management”. In: *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 2019, pp. 131–139.

[23] Wolfgang Hönig, Scott Kiesel, Andrew Tinka, Joseph W. Durham, and Nora Ayanian. “Persistent and Robust Execution of MAPF Schedules in Warehouses”. In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1125–1131.

[24] Wolfgang Hönig, James A. Preiss, T. K. Satish Kumar, Gaurav S. Sukhatme, and Nora Ayanian. “Trajectory Planning for Quadrotor Swarms”. In: *IEEE Transactions on Robotics* 34.4 (2018), pp. 856–869.

[25] He Jiang, Yulun Zhang, Rishi Veerapaneni, and Jiaoyang Li. “Scaling Lifelong Multi-Agent Path Finding to More Realistic Settings: Research Challenges and Opportunities”. In: *Proceedings of the Symposium on Combinatorial Search (SoCS)*. 2024, pp. 234–242.

[26] Florian Laurent, Manuel Schneider, Christian Scheller, Jeremy Watson, Jiaoyang Li, Zhe Chen, Yi Zheng, Shao-Hung Chan, Konstantin Makhnev, Oleg Svidchenko, Vladimir Egorov, Dmitry Ivanov, Aleksei Shpilman, Evgenija Spirovska, Oliver Tanevski, Aleksandar Nikov, Ramon Grunder, David Galevski, Jakov Mitrovski, Guillaume Sartoretti, Zhiyao Luo, Mehul Damani, Nilabha Bhattacharya, Shivam Agarwal, Adrian Egli, Erik Nygren, and Sharada Mohanty. “Flatland Competition 2020: MAPF and MARL for Efficient Train Coordination on a Grid World”. In: *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*. 2021, pp. 275–301.

[27] Jiaoyang Li, Zhe Chen, Daniel Harabor, Peter J. Stuckey, and Sven Koenig. “Anytime Multi-Agent Path Finding via Large Neighborhood Search”. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2021, pp. 4127–4135.

[28] Jiaoyang Li, Zhe Chen, Daniel Harabor, Peter J. Stuckey, and Sven Koenig. “MAPF-LNS2: Fast Repairing for Multi-Agent Path Finding via Large Neighborhood Search”. In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2022, pp. 10256–10265.

[29] Jiaoyang Li, Zhe Chen, Yi Zheng, Shao-Hung Chan, Daniel Harabor, Peter J. Stuckey, Hang Ma, and Sven Koenig. “Scalable Rail Planning and Replanning: Winning the 2020 Flatland Challenge”. In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. 2021, pp. 477–485.

[30] Jiaoyang Li, Ariel Felner, Eli Boyarski, Hang Ma, and Sven Koenig. “Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search”. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2019, pp. 442–449.

[31] Jiaoyang Li, Graeme Gange, Daniel Harabor, Peter J. Stuckey, Hang Ma, and Sven Koenig. “New Techniques for Pairwise Symmetry Breaking in Multi-Agent Path Finding”. In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. 2020, pp. 193–201.

[32] Jiaoyang Li, Daniel Harabor, Peter J. Stuckey, Hang Ma, and Sven Koenig. “Symmetry-Breaking Constraints for Grid-Based Multi-Agent Path Finding”. In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2019, pp. 6087–6095.

[33] Jiaoyang Li, The Anh Hoang, Eugene Lin, Hai L. Vu, and Sven Koenig. “Intersection Coordination with Priority-Based Search for Autonomous Vehicles”. In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2023, pp. 11578–11585.

[34] Jiaoyang Li, Wheeler Ruml, and Sven Koenig. “EECBS: Bounded-Suboptimal Search for Multi-Agent Path Finding”. In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2021, pp. 12353–12362.

[35] Jiaoyang Li, Kexuan Sun, Hang Ma, Ariel Felner, T. K. Satish Kumar, and Sven Koenig. “Moving Agents in Formation in Congested Environments”. In: *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 2020, pp. 726–734.

[36] Jiaoyang Li, Pavel Surynek, Ariel Felner, Hang Ma, T. K. Satish Kumar, and Sven Koenig. “Multi-Agent Path Finding for Large Agents”. In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2019, pp. 7627–7634.

[37] Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W. Durham, T. K. Satish Kumar, and Sven Koenig. “Lifelong Multi-Agent Path Finding in Large-Scale Warehouses”. In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2021, pp. 11272–11281.

[38] Jiaoyang Li, Han Zhang, Mimi Gong, Zi Liang, Weizi Liu, Zhongyi Tong, Liangchen Yi, Robert Morris, Corina Pasareanu, and Sven Koenig. “Scheduling and Airport Taxiway Path Planning under Uncertainty”. In: *Proceedings of the AIAA Aviation Forum (AIAA)*. 2019, pp. 1–7.

[39] Minghua Liu, Hang Ma, Jiaoyang Li, and Sven Koenig. “Task and Path Planning for Multi-Agent Pickup and Delivery”. In: *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 2019, pp. 1152–1160.

[40] Hang Ma. “A Competitive Analysis of Online Multi-Agent Path Finding”. In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. 2021, pp. 234–242.

[41] Hang Ma, Daniel Harabor, Peter J. Stuckey, Jiaoyang Li, and Sven Koenig. “Searching with Consistent Prioritization for Multi-Agent Path Finding”. In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2019, pp. 7643–7650.

[42] Hang Ma, Jiaoyang Li, T. K. Satish Kumar, and Sven Koenig. “Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks”. In: *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 2017, pp. 837–845.

[43] Hang Ma, Craig Tovey, Guni Sharon, T. K. Satish Kumar, and Sven Koenig. “Multi-Agent Path Finding with Payload Transfers and the Package-Exchange Robot-Routing Problem”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2016, pp. 3166–3173.

[44] Hang Ma, Glenn Wagner, Ariel Felner, Jiaoyang Li, T. K. Satish Kumar, and Sven Koenig. "Multi-Agent Path Finding with Deadlines". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2018, pp. 417–423.

[45] Hang Ma, Jingxing Yang, Liron Cohen, T. K. Satish Kumar, and Sven Koenig. "Feasibility Study: Moving Non-Homogeneous Teams in Congested Video Game Environments". In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*. 2021, pp. 270–272.

[46] Akmaral Moldagalieva, Joaquim Ortiz-Haro, Marc Toussaint, and Wolfgang Hönig. "db-CBS: Discontinuity-Bounded Conflict-Based Search for Multi-Robot Kinodynamic Motion Planning". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 14569–14575.

[47] Bernhard Nebel. "On the Computational Complexity of Multi-Agent Pathfinding on Directed Graphs". In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. 2020, pp. 212–216.

[48] Keisuke Okumura. "Engineering LaCAM*: Towards Real-Time, Large-Scale, and Near-Optimal Multi-Agent Pathfinding". In: *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2024, pp. 1051–1059.

[49] Keisuke Okumura. "Improving LaCAM for Scalable Eventually Optimal Multi-Agent Pathfinding". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2023, pp. 243–251.

[50] Keisuke Okumura. "LaCAM: Search-Based Algorithm for Quick Multi-Agent Pathfinding". In: *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*. 2023, pp. 11655–11662.

[51] Keisuke Okumura and Xavier Défago. "Quick Multi-Robot Motion Planning by Combining Sampling and Search". In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2023, pp. 252–261.

[52] Judea Pearl and Jin H. Kim. "Studies in Semi-Admissible Heuristics". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*. 1982, pp. 392–399.

[53] Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. "Loosely Synchronized Search for Multi-agent Path Finding with Asynchronous Actions". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 9714–9719.

[54] Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. "Multi-objective Conflict-based Search for Multi-agent Path Finding". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 8786–8791.

[55] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall, 2010.

[56] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. "Conflict-Based Search for Optimal Multi-Agent Pathfinding". In: *Artificial Intelligence* 219 (2015), pp. 40–66.

[57] David Silver. “Cooperative Pathfinding”. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*. 2005, pp. 117–122.

[58] David Šišlák, Přemysl Volf, and Michal Pěchouček. “Agent-Based Cooperative Decentralized Airplane-Collision Avoidance”. In: *IEEE Transactions on Intelligent Transportation Systems* 12.1 (2011), pp. 36–46.

[59] Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Roman Barták, and Eli Boyarski. “Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks”. In: *Proceedings of the International Symposium on Combinatorial Search (SoCS)*. 2019, pp. 151–159.

[60] Pavel Surynek. “Multi-Goal Multi-Agent Path Finding via Decoupled and Integrated Goal Vertex Ordering”. In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2021, pp. 12409–12417.

[61] Jirí Svancara, Marek Vlk, Roni Stern, Dor Atzmon, and Roman Barták. “Online Multi-Agent Pathfinding”. In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2019, pp. 7732–7739.

[62] Mingkai Tang, Yuanhang Li, Hongji Liu, Yingbing Chen, Ming Liu, and Lujia Wang. “MGCBS: an optimal and efficient algorithm for solving multi-goal multi-agent path finding problem”. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2024, pp. 249–256.

[63] Jordan T. Thayer and Wheeler Ruml. “Bounded Suboptimal Search: A Direct Approach Using Inadmissible Estimates”. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2011, pp. 674–679.

[64] Qian Wan, Chonglin Gu, Sankui Sun, Mengxia Chen, Hejiao Huang, and Xiaohua Jia. “Lifelong Multi-Agent Path Finding in a Dynamic Environment”. In: *Proceedings of the International Conference on Control, Automation, Robotics and Vision (ICARCV)*. 2018, pp. 875–882.

[65] Ko-Hsin Cindy Wang and Adi Botea. “Fast and Memory-Efficient Multi-Agent Pathfinding”. In: *Proceedings of the International Conference on International Conference on Automated Planning and Scheduling (ICAPS)*. 2008, pp. 380–387.

[66] Peter R. Wurman, Raffaello D’Andrea, and Mick Mountz. “Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses”. In: *AI Magazine*. 2008, pp. 9–20.

[67] Qinghong Xu, Jiaoyang Li, Sven Koenig, and Hang Ma. “Multi-Goal Multi-Agent Pickup and Delivery”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 9964–9971.

[68] Jingtian Yan and Jiaoyang Li. “Multi-Agent Motion Planning With Bézier Curve Optimization Under Kinodynamic Constraints”. In: *IEEE Robotics and Automation Letters* 9.3 (2024), pp. 3021–3028.

[69] Jingjin Yu. “Intractability of Optimal Multirobot Path Planning on Planar Graphs”. In: *IEEE Robotics and Automation Letters* 2.4 (2016), pp. 33–40.

[70] Jingjin Yu and Steven M. LaValle. “Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs”. In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2013, pp. 1443–1449.

[71] Jingjin Yu and Daniela Rus. “Pebble Motion on Graphs with Rotations: Efficient Feasibility Tests and Planning Algorithms”. In: *Proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR)*. 2014, pp. 729–746.

[72] Han Zhang, Jingkai Chen, Jiaoyang Li, Brian Williams, and Sven Koenig. “Multi-Agent Path Finding for Precedence-Constrained Goal Sequences”. In: *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 2022, pp. 1464–1472.

[73] Hejia Zhang, Shao-Hung Chan, Jie Zhong, Jiaoyang Li, Peter Kolapo, Sven Koenig, Zach Agioutantis, Steven Schafrik, and Stefanos Nikolaidis. “Multi-robot geometric task-and-motion planning for collaborative manipulation tasks”. In: *Autonomous Robots* 47 (2023), pp. 1537–1558.

[74] Yulun Zhang, He Jiang, Varun Bhatt, Stefanos Nikolaidis, and Jiaoyang Li. “Guidance Graph Optimization for Lifelong Multi-Agent Path Finding”. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2024, pp. 311–320.

[75] Yi Zheng, Srivatsan Ravi, Erik Kline, Sven Koenig, and T. K. Satish Kumar. “Conflict-Based Search for the Virtual Network Embedding Problem”. In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. 2022, pp. 423–433.

[76] Shuai Zhou, Shizhe Zhao, and Zhongqiang Ren. “Loosely Synchronized Rule-Based Planning for Multi-Agent Path Finding with Asynchronous Actions”. In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2025, pp. 14763–14770.