Flow-Based Guidance Framework with Flex Distribution for w-Optimal Multi-Agent Path Finding

Shao-Hung Chan[1], **Thomy Phan**[2], **Sven Koenig**[3,4]
[1]University of Southern California
[2]University of Bayreuth
[3]University of California, Irvine
[4]University of Örebro
shaohung@use.edu, thomy.phan@uni-bayreuth.de, sven.koenig@uci.edu

Abstract

Multi-Agent Path Finding (MAPF) is the one-shot problem of finding collision-free paths in a shared environment while minimizing the sum of the agents' travel times. Since solving MAPF optimally is NP-hard, w-optimal algorithms such as Explicit Estimation Conflict-Based Search (EECBS) have been used to speed up the search while providing a guarantee on the solution quality. However, the scalability of EECBS is limited in large-scale MAPF instances. While EECBS can be accelerated for regularly structured environments, such as Kiva warehouses, by utilizing specialized guidance heuristics, these heuristics are ineffective in more general and large-scale environments. To fill this gap, we propose the *Flow-Based Guidance Framework* (FBGF), a general two-phase process that simulates a list of paths and then generates the *Flow-Based Guidance Heuristic* (FH) without making prior assumptions about the environment's structure. We identify features that distinguish w-optimal MAPF from other MAPF variants and propose strategies to enhance its effectiveness for guidance, complemented by the flex distribution technique from EECBS. The empirical evaluation demonstrates that our FH significantly reduces collisions, thereby achieving higher success rates than the state-of-the-art within 60 seconds.

1 Introduction

Multi-Agent Path Finding (MAPF) is the one-shot problem of finding collision-free paths in a shared environment, allowing each agent to move from its start to its target location. An optimal solution for a MAPF instance is a list of collision-free paths with a minimal sum of travel time for each agent. MAPF has numerous applications, including autonomous warehouses (Wurman, D'Andrea, and Mountz 2008), traffic management (Li et al. 2023), and drone control (Ho et al. 2019). Since finding optimal MAPF solutions is NP-hard (Yu and LaValle 2013), researchers have been motivated to trade off solution quality with runtime. One direction is to find a *w-optimal* (i.e., bounded-suboptimal) solution whose sum of travel time deviates from the optimum by at most a user-specified factor $w \geq 1$. Explicit Estimation Conflict-Based Search (EECBS) (Li, Ruml, and Koenig 2021) is the leading w-optimal MAPF algorithm. Derived from a two-level framework, EECBS first finds paths for agents individually on the low level and then resolves collisions on the high level while maintaining the best-known sum of lowerbounds on the sum of travel time of the optimal solution.

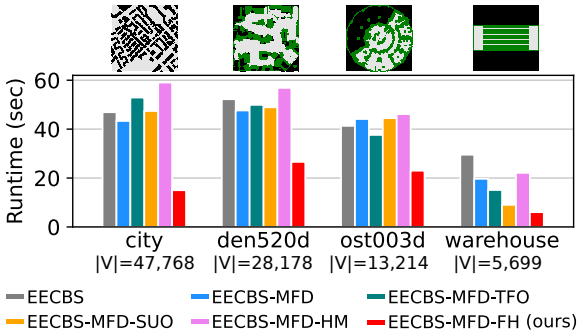


Figure 1: Average runtime between the state-of-the-art and our approach (in red) over all MAPF instances on each graph with number of vertices $|V|$. The MAPF instances here are the same as those in Figure 3.

Although EECBS is the state-of-the-art approach for finding a w-optimal solution for a MAPF instance, it still exhibits limited scalability for large-scale MAPF instances where many agents move in a large environment. Prior guidance approaches either specialize in particular environment structures or require a lifelong setting, where guidance heuristics can be computed online, without further guarantees on the solution quality. Nevertheless, it is non-trivial to directly apply these lifelong approaches to one-shot tasks, and efficiently generating heuristics that can effectively guide the agents while guaranteeing w-optimality in large-scale environments remains challenging. This is because, unlike lifelong MAPF, w-optimal MAPF is a *one-shot problem* with the following features:

(F1) Agents wait at their target locations permanently,

(F2) SOC must be guaranteed w-optimal, and

(F3) A finite time budget is specified w.r.t. resources.

Here, feature (F2) also determines the difference between w-optimal MAPF and (unbounded) suboptimal MAPF.

Thus, to improve the efficiency of EECBS in finding w-optimal solutions, we propose the *Flow-Based Guid-*

ance Framework as a general pre-processing technique, which outputs the *Flow-Based Guidance Heuristic* (FH) that guides the agents while guaranteeing w-optimality. Given a MAPF instance, our framework generates flows by simulating agents' paths with strategies targeting w-optimal MAPF. Next, it constructs the *Flow-Based Guidance Graph* (FGG) to coordinate flows and compute FH as guidance without further prior assumptions about the environment structure. Our contributions are as follows:

- We introduce a two-phase *Guidance Framework* as a general pre-processing technique for w-optimal MAPF.
- We propose the *Flow-Based Guidance Framework* (see Figure 2) that contains strategies exploiting observations from w-optimal MAPF, resulting in *Flow-Based Guidance Heuristics* (FH) that guides the agents to find guaranteed w-optimal solutions.
- Our Flow-Based Guidance Framework can efficiently output FH to speed up EECBS effectively while providing the w-optimality guarantee, outperforming the state-of-the-art guidance heuristics in runtime (see Figure 1).

2 Background

2.1 Multi-Agent Path Finding

A MAPF instance (Stern et al. 2019) (Russell and Norvig 2010) consists of an undirected graph $G = (V, E)$ and a set of k agents $\{a_1, \dots, a_k\}$. Each agent a_i has a unique start vertex s_i and a unique target vertex l_i. A *path* of an agent, starting at its start vertex and ending at its target vertex, is a sequence of vertices indicating where the agent is at each timestep. Each agent permanently waits at its target vertex after it completes its path. The *cost* of a path is the number of timesteps needed by the agent to move from its start vertex to its target vertex, ignoring subsequent timesteps after reaching its target vertex. When a pair of agents respectively follow their paths, a *vertex conflict* occurs iff they reach the same vertex at the same timestep. An *edge conflict* occurs iff these two agents traverse the same edge in opposite directions at the same timestep. A *solution* is a list of conflict-free paths, one for each agent. A solution is *optimal* iff its *sum of (path) costs* (SOC) is minimum, denoted as C^*, and *w-optimal* iff its SOC is at most $w \cdot C^*$, where $w \geq 1$ is a user-specified suboptimality factor.

2.2 Explicit Estimation Conflict-Based Search

Explicit Estimation Conflict-Based Search (EECBS) (Li, Ruml, and Koenig 2021) is a two-level w-optimal algorithm for MAPF. Its strategy is to iteratively resolve a conflict by introducing constraints on the high level and then finding the paths on the low level to satisfy the constraints. On the high level, EECBS constructs a *Constraint Tree* (CT), where a constraint indicates that an agent is not allowed to reach a vertex or traverse an edge at a particular timestep. A *CT node* N contains a set of constraints and a list of paths, one for each agent, that satisfy its set of constraints. EECBS then runs Explicit Estimation Search (EES) (Thayer and Ruml 2011), a w-optimal heuristic search algorithm, on CT. It maintains a set of lists that contain all the generated but not yet expanded CT nodes, denoted as LISTs.

On the low level, to find a path for an agent a_i in a CT node N, EECBS constructs a search tree with each vertex-timestep (v-t) node n containing a tuple (v, t) that indicates an agent staying at vertex v at timestep t. For a v-t node $n = (v, t)$, we define a priority function $f_i(n) = g_i(n) + h_i(v)$, where $g_i(n) = t$ is the number of timesteps for agent a_i to move from its start vertex s_i to vertex v and $h_i(v)$ is an admissible heuristic that underestimates the number of timesteps needed to move from vertex v to its target vertex l_i. The number of conflicts $x_i(n)$ is computed with the paths of the other agents. EECBS runs *focal search* (Pearl and Kim 1982) to find an individually w-optimal path. Focal search maintains two lists: OPEN$_L$ and FOCAL$_L$. OPEN$_L$ sorts all the generated but not yet expanded v-t nodes n in increasing order of priority function $f_i(n)$. FOCAL$_L$ contains those v-t nodes in OPEN$_L$ whose $f_i(n)$ are less than or equal to a *threshold* $\tau_i = w \cdot f_{\min,i}(N)$, where $f_{\min,i}(N)$ is the minimum f-value among all v-t nodes in OPEN$_L$. Focal search sorts these v-t nodes $n \in$ FOCAL$_L$ in increasing order of their number of conflicts $x_i(n)$, tie-breaking with a secondary heuristic $\tilde{h}(n)$ (Cohen et al. 2016). At each iteration, focal search expands the top v-t node in FOCAL$_L$ that has the minimum x_i value. Since $f_i(n)$ of any v-t node n in FOCAL$_L$ is at most threshold τ_i, the focal search always finds an individually w-optimal path that satisfies constraints, i.e., $c_i(N) \leq w \cdot lb_i(N), \forall i \in [k]$ holds. The SOC of the CT node N thus satisfies

$$C(N) = \sum_{j \in [k]} c_j(N) \leq w \cdot \sum_{j \in [k]} lb_j(N) = w \cdot LB(N). \quad (1)$$

EECBS keeps track of the minimum sum of lowerbounds (SOLB) among all the CT nodes in LISTs, which indicates the best-known SOLB LB on the SOC of the optimal solution.

2.3 Flex Distribution

EECBS finds a w-optimal solution by requiring each path in a CT node to be individually w-optimal, i.e., $c_i(N) \leq w \cdot lb_i(N), \forall i \in [k]$. However, as long as the SOC of a CT node is w-optimal, EECBS can still find a w-optimal solution without forcing each path to be individually w-optimal. Suppose that EECBS expands CT node $\hat{N}$ and generates one of its child CT nodes N. To find a path for agent a_i in CT node N via focal search while remaining the paths of other agents $a_{j \in [k] \setminus \{i\}}$ fixed, Chan et al. (2022) uses *flex distribution*, which is an approach that relaxes the threshold $\tau_i(N)$ by adding $\sum_{j \in [k] \setminus \{i\}} (w \cdot lb_j(N) - c_j(N))$, where the range $w \cdot lb_j(N) - c_j(N)$ is defined as *flex* from the path for agent a_j. In this case, although each path in a CT node is no longer individually w-optimal, the SOC of each CT node remains w-optimal toward its SOLB. Meanwhile, with the relaxed threshold, the paths in the generated CT nodes can further reduce the number of conflicts and thus speed up the search.

3 Related Work

3.1 Highway Heuristics

Given the graph $G = (V, E)$ of a MAPF instance, Cohen et al. (2016) constructs a weighted directed graph for com-

puting the highway heuristics. For each edge $(u, v) \in E$ (where $u, v \in V$), a pair of directed edges, $e_{uv} = u \to v$ and $e_{vu} = v \to u$, are added to the set of edges in the weighted directed graph. The weight w_{uv} of each directed edge e_{uv} is determined as

$$w_{uv} = \begin{cases} 1, & \text{if } e_{uv} \in \textit{highways}; \\ w, & \text{otherwise}, \end{cases} \tag{2}$$

where *highways* is a subset of directed edges. When computing the heuristics, edges not belonging to highways are penalized by increasing their weights to w.

When finding a path for agent a_i, for each v-t node $n = (v, t)$, the *highway heuristic* is defined as the minimum distance from vertex v to its target vertex l_i in the weighted directed graph. Then, the highway heuristic is used as the secondary heuristic for the low-level focal search, breaking ties between two v-t nodes with the same number of conflicts in FOCAL$_L$ with a lower highway heuristic value. That is, agents tend to move along the paths following more edges belonging to highways.

To select the subset of highways, Cohen et al. (2016) proposes separate approaches: the *Criss-Cross* (CC) and the *Heat-Map* (HM). The CC-based approach selects highways via handcrafted rules. However, it strongly depends on the geometry of the graph, which limits its generality. On the other hand, the HM-based approach is an iterative process. At each iteration, it randomly selects a pair of start and target vertices (that may not belong to the given MAPF instance) and then finds a minimum-cost path on a weighted directed graph, with the cost of each directed edge e_{uv} being a function of the numbers that e_{uv} and e_{vu} appeared in the previous paths. When the process terminates, it selects a subset of directed edges with low weights as the highways. In general, the HM-based approach encourages agents to follow the previously planned paths with lower edge weights involved.

3.2 Guidance Heuristics of Other MAPF Variants

To the best of our knowledge, the highway heuristics approach is the only work that aims to guide agents while guaranteeing the finding of w-optimal solutions. Still, this approach motivated researchers to explore ideas of guiding agents for coordinated movements in two other MAPF variants: lifelong MAPF and (unbounded) suboptimal MAPF.

For lifelong MAPF, where agents receive a sequence of target vertices, Chen et al. (2024) proposed traffic flow optimization (TFO) that coordinates paths by updating the heuristics based on the traffic flows of the simulated time-independent paths (i.e., agents are not allowed to perform wait actions). However, as shown in the empirical evaluation, directly applying traffic flow optimization from lifelong MAPF results in inefficiency for w-optimal MAPF. On the other hand, Zhang et al. (2024) uses a model-based approach to build a guidance graph representing the estimated action costs per vertex. However, their model is graph-dependent, which limits its generality. Also, as the number of vertices increases, the number of variables grows, which causes huge overhead regarding time and data, and is thus limited in scalability. In this paper, we target large graphs with more

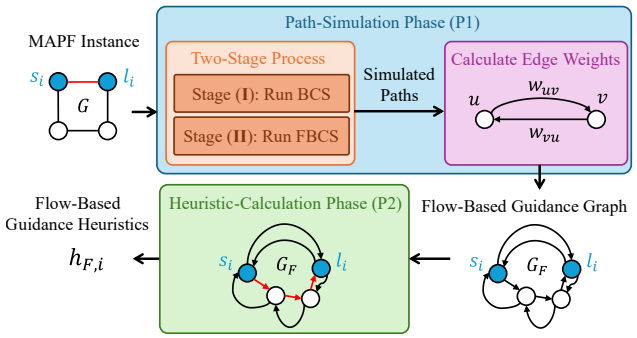


Figure 2: The block diagram of the Flow-Based Guidance Framework, which takes a MAPF instance as input and returns the Flow-Based Guidance Heuristics for each agent on each vertex. The shortest paths from vertices s_i to l_i in graphs G and G_F are marked in red.

than $5K$ vertices, which are intractable with their approach. For suboptimal MAPF, where the solution quality is unbounded, Han and Yu (2022) proposed space-utility optimization (SUO) that estimates the space utility in the graph of the MAPF instance. Their approach guides the agents with the paths that traverse the vertices/edges with low congestion. That is, their approach optimized the space utility by finding paths that avoid congestion. However, since their approach targets suboptimal MAPF without any guarantees on the solution quality, agents following their guidance when aiming for a w-optimal path may inevitably encounter conflicts, which causes overhead to resolve (see Table 1).

4 Guidance Framework for MAPF

Pre-processing MAPF instances to avoid conflicts or congestion while finding paths for agents has become popular. Based on the related work, we thus introduce a general framework called *Guidance Framework*, which contains two phases: the *path-simulation* phase (P1) and the *heuristic-calculation* phase (P2). In the path-simulation phase (P1), given a graph $G = (V, E)$ from a MAPF instance, the guidance framework constructs a weighted directed graph $G' = (V, E', W')$, known as the guidance graph (Zhang et al. 2024). For each undirected edge $e = \{u, v\} \in E$, there is a pair of directed edges $e_{uv} = (u \to v), e_{vu} = (v \to u) \in E'$ with respective weights $w_{uv}, w_{vu} \in W'$. The guidance framework then finds paths from graph G' and updates its weights accordingly. Since the guidance graph G' is different from the original graph G in the MAPF instance, we term finding a path in G' to be *simulation*. After the path-simulation phase (P1), the guidance framework moves to the heuristic-calculation phase (P2) to generate *guidance heuristics* for each agent a_i by calculating the shortest-path distances in the guidance graph G' from each vertex $v \in V$ to its target vertex l_i.

The related works can be viewed as instantiations of our guidance framework. For example, the heat map-based approach (Cohen et al. 2016) is an instantiation that (P1) sequentially selects a start vertex and a target vertex and simulates the shortest paths on a directed weighted graph with

edge weights taking into account the simulated paths, and (P2) calculates the highway heuristics from the weighted graph as guidance. The guidance framework can also be applied to approaches used to solve other MAPF variants. For example, the traffic flow optimization for lifelong MAPF (Chen et al. 2024) is an instantiation that (P1) selects all the agents in the lifelong MAPF instance and simulate (or refine) their time-independent paths on a weighted graph, where the weights are from the handcrafted functions that consider the simulated paths, and (P2) calculates the guidance heuristics via backward breadth first search.

5 Flow-Based Guidance Framework (FBGF)

To guide the agents and improve the efficiency of one-shot w-optimal MAPF, we follow our two-phase guidance framework and propose the Flow-Based Guidance Framework (FBGF) as a general pre-processing technique before running EECBS. Figure 2 shows the block diagram of the framework. In the path-simulation phase (P1), we simulate paths to construct the *Flow-Based Guidance Graph* (FGG) G_F, which is an instantiation of the guidance graph G' in the guidance framework. Next, in the heuristic-calculation phase (P2), we rely on FGG and calculate the *Flow-Based Guidance Heuristic* (FH) for each agent on each vertex.

5.1 Path-Simulation Phase (P1)

In the path-simulation phase (P1), we simulate one path for each agent sequentially. Before finding the simulated path for an agent a_i, we run the backward Dijkstra algorithm from its target vertex l_i on the graph G to get the shortest-path distance for each vertex v, which serves as an admissible heuristic $h_i(v)$. To handle the features (F1), (F2), and (F3) (see Introduction section) when simulating a path for FGG, we develop the following strategies, respectively:

(S1) View other agents' target vertices as static obstacles,

(S2) Use a two-stage process with flexible bounded-cost search (FBCS) to find paths sequentially, and

(S3) Limit the number of simulated paths.

For feature (F1), after an agent waits at its target vertex permanently, it may encounter numerous conflicts with other agents that traverse through, known as *target conflicts*, which may cause computational overhead to resolve, even when advanced approaches are used (Li et al. 2020). Thus, we develop our strategy (S1) that guides agents to avoid target conflicts. When finding a simulated path for an agent a_i, we view other agents' target vertices $l_{j\in[k]\setminus\{i\}}$ as static obstacles, defined as *target obstacles*. That is, the simulated paths we use for constructing FGG never traverse any target vertices except their own.

As we introduce additional target obstacles from strategy (S1), focal search may expand v-t nodes whose f_i values exceed the threshold. Thus, to handle feature (F2), instead of running focal search sequentially, we propose our strategy (S2), which is a two-stage process that simulates paths. At stage (**I**), we run bounded-cost search (BCS), modified from Haslum (2013), sequentially, attempting to simulate one path for each agent that considers target obstacles from

Algorithm 1: Flexible Bounded-Cost Search (FBCS)

```
1:  procedure FBCS(s_i, l_i, P)
2:      ▷ P[j] is the simulated path of agent a_j. ◁
3:      τ_i ← w · f_i(s_i)
4:      for j from 1 to k do        ▷ Skip the for loop for BCS.
5:          if j = i or P[j] is empty then
6:              continue
7:          c_j ← cost of path P[j]
8:          τ_i ← τ_i + (w · f_j((s_j, 0)) − c_j)
9:      root v-t node r ← (s_i, 0)
10:     FOCAL_L and CLOSED_L ← empty sets
11:     Insert v-t node r to FOCAL_L
12:     while FOCAL_L not empty do
13:         n_x ← top v-t node in FOCAL_L
14:         Remove v-t node n_x from FOCAL_L
15:         Insert v-t node n_x to CLOSED_L
16:         if IsGoal(n_x) then
17:             return ExtractPath(n)
18:         neighbors ← ExpandNode(n)
19:         for n' = (v', t') ∈ neighbors do
20:             if v' ∈ {l_{j∈[k]\{i}}} or f_i(n') > τ_i then
21:                 continue ▷ Use strategies (S1) and (S2).
22:             Find n̄ with the same (v', t') in CLOSED_L
                ∪ FOCAL_L
23:             if n̄ not found then
24:                 Insert v-t node n' to FOCAL_L
25:                 continue
26:             if IsDominant(n', n̄) then
27:                 isInCLOSED ← true if n̄ ∈ CLOSED_L
28:                 g_i(n̄) ← g_i(n'), x_i(n̄) ← x_i(n')
29:                 if isInCLOSED is true then
30:                     Remove n̄ from CLOSED_L
31:                     Insert v-t node n̄ to FOCAL_L
32:                 else
33:                     Update the priority of n̄ in FOCAL_L
34:     return No path exists
```

strategy (S1). BCS maintains only one list, FOCAL_L, that contains all the generated but not yet expanded v-t nodes n, sorted in increasing order of their number of collisions $x_i(n)$, tie-breaking with a smaller h_i value for v-t nodes in FOCAL_L. To start BCS, we generate a root v-t node $n_0 = (s_i, 0)$ and add it to FOCAL_L, where $f_i(n_0) = h_i(s_i)$ is the lowerbound on the cost of the optimal path for agent a_i since $h_i(s_i)$ is admissible. At each iteration, to deploy our strategies (S1) and (S2), we expand the top v-t node n from FOCAL_L (which has the minimum number of collisions) and only generate its child v-t nodes $n' = (v', t')$ if $v' \notin \{l_j \mid a_{j\neq i} \in A\}$ and $f_i(n') \leq w \cdot f_i(n_0)$. We terminate BCS either when a target v-t node (l_i, t_l) is expanded, where t_l is the timestep reaching target vertex l_i (i.e., a bounded-cost path is found) or when FOCAL_L is empty (i.e., no such bounded-cost path exists). If a simulated path is found, it is guaranteed to have cost at most $w \cdot f_i(n_0)$, where $f_i(n_0)$ is the lowerbound on the cost of the optimal path for agent a_i. Thus, the simulated path found by BCS is individually w-optimal under the situation where no constraint is intro-

Algorithm 2: Flow-Based Guidance Framework (FBGF)

```
1: procedure FBGF(MAPF instance, k_max)
2:     k_cnt ← 0
3:     P ← ∅                                   ▷ A list of simulated paths
4:     A' ← SORT(A)                            ▷ A list of sorted agents
5:     for i from 1 to k do ▷ Iteratively find path with BCS
6:         a_i' ← A'[i]                        ▷ Agent in the i-th order of A'
7:         p_i' ← BCS(s_i', l_i', P)
8:         if ∃p_i' then
9:             P[i'] ← p_i'
10:            k_cnt ← k_cnt + 1
11:        if k_cnt = k_max then               ▷ Use strategy (S3)
12:            break
13:    ▷ Iteratively find the rest of the paths with FBCS ◁
14:    for i from 1 to k do
15:        a_i' ← A'[i]                        ▷ Agent in the i-th order in A
16:        if P[i'] ≠ φ then ▷ Simulated path for a_i' exists
17:            continue
18:        p_i' ← FBCS(s_i', l_i', P)          ▷ Use Alg. 1
19:        if ∃p_i' then
20:            P[i'] ← p_i'
21:            k_cnt ← k_cnt + 1
22:        if k_cnt = k_max then               ▷ Use strategy (S3)
23:            break
24:    E_F ← e_uv, e_vu, ∀e = {u, v} ∈ E
25:    W_F ← ∅                                 ▷ Set of weight values
26:    for e_uv ∈ E_F do
27:        Φ_uv ← number of agents traversing e_uv
28:    Φ_max ← max{Φ_uv | ∀e_uv ∈ E_F}
29:    for e_uv ∈ E_F do
30:        Compute w_uv via Eq. (5)
31:        Insert w_uv to W_F
32:    return G_F = (V, E_F, W_F)
```

duced, which can be used as the initial paths for EECBS (i.e., paths in its root CT node).

Due to the target obstacles and the bounded-cost constraint, BCS may not find a path within the fixed threshold. However, since EECBS is guaranteed to find a w-optimal solution as long as the SOC of each CT node is w-optimal, we can further increase the threshold by introducing the flex from other simulated paths. Thus, in our stage (**II**) of the path-simulation phase, we propose the *flexible bounded-cost search* (FBCS) to sequentially find simulated paths for agents that failed in stage (**I**). As shown in Alg. 1, when finding a simulated path for an agent a_i that does not have a path after stage (**I**), FBCS increases the threshold τ_i by adding the *flex* from other simulated paths, i.e.,

$$\tau_i = w \cdot f_i((s_i, 0)) + \sum_{\{j|P[j]\neq\phi\}} (w \cdot f_j((s_j, 0)) - c_j), \quad (3)$$

where c_j is the cost of the simulated path $P[j]$. Thus, the range $w \cdot f_j((s_j, 0)) - c_j$ is the flex from the simulated path $P[j]$. After finding a simulated path for agent a_i with FBCS, if it exists, we add it to the simulated path list P. Let $J' = \{j' \mid j' \neq i \wedge P[j'] \neq \phi\}$. The SOC of P thus becomes

$$\begin{aligned}
&\sum_{\{j|P[j]\neq\phi\}} c_j \leq \tau_i + \sum_{j'\in J'} c_{j'} \\
&= w \cdot f_i((s_i, 0)) + \sum_{j'\in J'} (w \cdot f_{j'}(s_{j'}) - c_{j'}) + \sum_{j'\in J'} c_{j'} \\
&= w \cdot \sum_{\{j|P[j]\neq\phi\}} f_j((s_j, 0)),
\end{aligned} \quad (4)$$

which is still w-optimal toward its SOLB. Thus, the simulated path list P can be used as the (initial) paths for the root CT node of EECBS. On the other hand, simulating paths for all the agents can result in huge runtime overhead (see Table 2). Thus, to handle feature (F3), instead of randomly selecting a fixed subset of agents and simulating their paths, our strategy (S3) is to first sort the agents in increasing order according to their shortest-path distance in G, i.e., $h_i(s_i), \forall i \in [k]$, and then sequentially find their simulated paths until a user-specified *path-found threshold* k_{max}. By controlling k_{max}, we can split the time budget into the preprocessing and MAPF problem-solving.

After simulating paths with our strategies (S1), (S2), and (S3), we determine the edge weights of the FGG via linear interpolation of their flows. The flow of a directed edge is defined as the number of agents traversing through it when following their simulated paths. Based on the flow of a directed edge, we normalize its weight in the range $[1, c_p]$, where c_p is defined as the *maximum penalty cost*. Given a directed edge $e_{uv} \in E_F$ with the number of agents Φ_{uv} traversing it among all the simulated paths, its weight $w_{uv} \in W_F$ is determined as

$$w_{uv} = 1 + (c_p - 1) \cdot \frac{\Phi_{\max} - \Phi_{uv}}{k}, \quad (5)$$

where $\Phi_{\max}$ is the maximum flow among all the simulated paths. According to Eq. (5), the weights of the directed edges have a negative correlation with their flows: the weights of directed edges with the maximum flow are set to 1, and those with zero flow are set to c_p if $\Phi_{\max} = k$ holds. That is, we promote the directed edges with higher flow (i.e., more agents traversing) by giving them lower weight values. Thus, we coordinate the agents during the search by encouraging them to follow the directed edges that have been optimized by the flows from the simulated paths. The rationale is as follows: the FGG is already based on (F)BCS, which aims to minimize the number of conflicts (with the previously simulated paths). Thus, following such flows can result in fewer conflicts (see Table 1). Alg. 2 shows the process of our path-simulation phase (P1), which includes simulating paths via BCS [lines 2-14] in stage (**I**) and via FBCS [lines 15-26] in stage (**II**), and calculating edge weights E_F [lines 27-34]. The output is the FGG G_F that will be passed into the heuristic-calculation phase (P2).

5.2 Heuristic-Calculation Phase (P2)

In the heuristic-calculation phase (P2), we calculate the *Flow-Based Guidance Heuristic* (FH) for each agent from

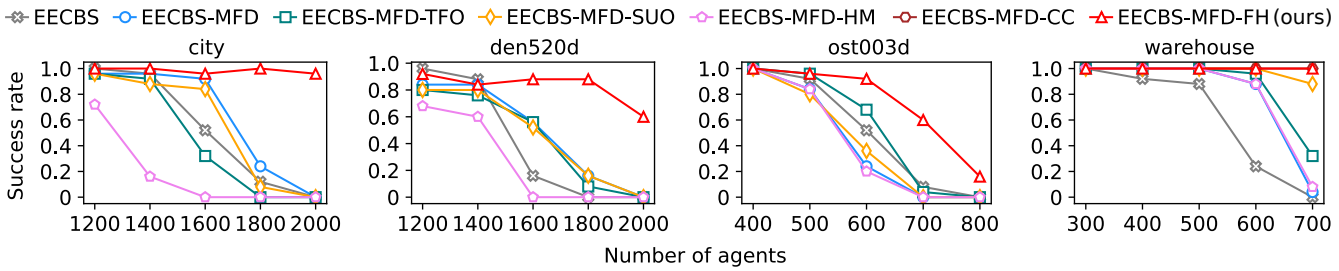


Figure 3: Success rates of EECBS, EECBS-MFD, and EECBS-MFD with different guidance heuristics (TFO, SUO, HM, CC, and our FH) over MAPF instances with the same number of agents on the same graph.

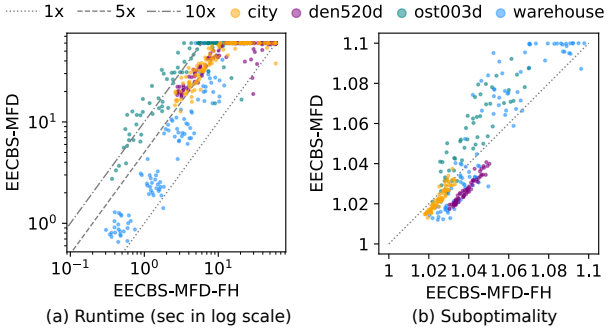


Figure 4: Comparisons between EECBS-MFD and EECBS-MFD-FH in (a) runtime among all MAPF instances and (b) suboptimality among all solutions.

the FGG constructed in phase (P1). The value of FH $h_{F,i}(v)$ is defined as the minimum distance from vertex v to the target vertex l_i in the FGG G_F, which can be obtained by running the backward Dijkstra algorithm on FGG G_F from target vertex l_i. Given a vertex v, its FH $h_{F,i}(v)$ for an agent a_i serves as a "guidance" that indicates which vertex v to explore during EECBS search. We follow Cohen et al. (2016) and use FH as the secondary heuristic.

Theorem 1. *EECBS is complete and w-optimal if any arbitrary heuristics are used as the secondary heuristics for the low-level focal search.*

Proof. Since the secondary heuristic is only used to break ties when two v-t nodes in FOCAL$_L$ have the same number of conflicts, the v-t nodes in FOCAL$_L$ have their f values at most the threshold, resulting in a path with cost at most the threshold, and the SOC of each CT node is w away from its SOLB. Thus, EECBS still finds a w-optimal solution when expanding a CT node with zero conflicts. This is also valid when flex distribution is introduced (Section 2.3). □

6 Empirical Evaluation

6.1 Experiment Setup

We evaluate our approach on four large four-connected grid graphs, from the MAPF benchmark suite (Stern et al. 2019): a `city` graph (*Boston_0_256*), two game graphs, which are `den520d` and `ost003d`, and a `warehouse` graph (*warehouse-10-20-10-2-1*) with corridor widths of one. We use the available 25 random scenarios under each number of agents listed on the x-axis of Figure 3. For the `warehouse` graph, we follow Cohen et al. (2016) and set the start and target vertices on two sides. Since we set up 5 different numbers of agents for each of the four graphs, we have 500 MAPF instances in total. For MAPF instances with more than 1000 agents, we generate agents with random start and target vertices in addition to the existing 1000 agents from each random scenario. We set the suboptimality factor $w = 1.1$ and the time budget of 60 seconds, following Li, Ruml, and Koenig (2021). The enhancements for EECBS include bypassing conflicts, prioritizing conflicts, and symmetric reasoning (Boyarski et al. 2015; Chan et al. 2022; Li et al. 2020). We use Mixed-Strategy Flex Distribution (MFD) (Chan et al. 2025) as our flex distribution mechanism. All the experiments are run on CentOS Linux, Intel Xeon 2640v4 CPUs, and 64 GB RAM.

6.2 Performance Comparison

We set the path-found threshold $k_{\max} = 0.75$ and the maximum penalty cost $c_p = 20$ across different ranges. As shown in Figure 3, we use the success rate as the metric to evaluate the efficiency of each approach. Used as the secondary heuristics in the low-level focal search of EECBS-MFD, we compare our FH with the state-of-the-art guidance heuristics, including Traffic-Flow Optimization (TFO) (Chen et al. 2024), Space-Utility Optimization (SUO) (Han and Yu 2022), Heat-Map (HM) (Cohen et al. 2016), and Criss Cross (CC) (Cohen et al. 2016). While CC can only be applied in the well-constructed `warehouse` graph, our FH reaches the same success rates of 1.00 as CC. Also, since our FBGF contains strategies targeting w-optimal MAPF, the resulting EECBS-MFD using FH outperforms all the state-of-the-art guidance heuristics in large-scale MAPF instances on more general graphs other than the `warehouse`.

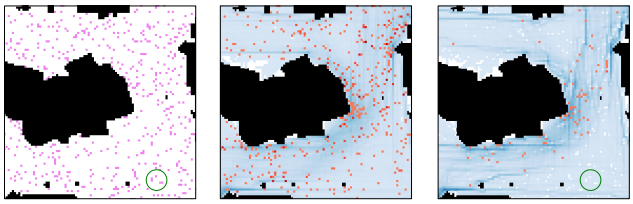
Figure 4(a) shows the instance-wise runtime comparison between EECBS-MFD with and without FH. Among all the 500 MAPF instances, 437 of them show that EECBS-MFD-FH is faster than EECBS-MFD, 172 of them show that EECBS-MFD-FH is 5 times faster than EECBS-MFD, and 34 of them show that EECBS-MFD-FH is 10 times faster than EECBS-MFD. To evaluate the solution quality, we define the suboptimality as the ratio between SOC and SOLB. Figure 4(b) shows the instance-wise suboptimality

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

Table 1: The numbers (in thousands) of conflicts $\mathcal{X}_0$ and target conflicts $\mathcal{X}_0^l$ of the root CT node, the number (in thousands) of low-level focal search runs #Runs, and the runtime (in seconds) of calculating guidance heuristics T' and generating root CT node T_0, averaging over all MAPF instances. Numbers in bold are the minimum among all approaches. TFO, SUO, HM, and FH (ours) are based on EECBS-MFD.

	$0.25k$	$0.50k$	$0.75k$	$1.00k$
`city`, k : 2000	0.29	1.80	4.11	44.75
`den520d`, k : 2000	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 2: The average runtime (in seconds) of calculating FH for different $k_{\max}$ among 25 MAPF instances with k agents.



(a) Target vertices (b) EECBS-MFD (c) EECBS-MFD-FH

Figure 5: The (a) target vertices of an MAPF instance in the center of the `den520d` graph and the heatmaps of paths (in blue) and conflicts (in red) from the root CT nodes of (b) EECBS-MFD and (c) EECBS-MFD-FH (ours). For (b) and (c), the darker the color, the higher the value a grid contains. The green circles show examples of avoiding target vertices when the agents are guided by FH.

comparison between EECBS-MFD and EECBS-MFD-FH, ignoring those when one of the approaches cannot find a solution within the 60-second time budget. Compared to EECBS-MFD, EECBS-MFD-FH has lower suboptimality for MAPF instances in `ost003d` and `warehouse` graphs and reaches similar suboptimality for those in `city` and `den520d` graphs. That is, EECBS-MFD, guided by our FH, can significantly reduce runtime while finding w-optimal solutions that are close to optimal.

6.3 Results Analysis

As shown in Table 1, in our FBGF, since (F)BCS prioritizes v-t nodes in FOCAL$_L$ by the minimum number of conflicts, the resulting FH finds paths in the root CT node with fewer conflicts $\mathcal{X}_0$ than other approaches. Furthermore, since we deploy strategy (S1) via target obstacles, EECBS-MFD-FH finds paths in the root CT node with fewer target conflicts $\mathcal{X}_0^l$

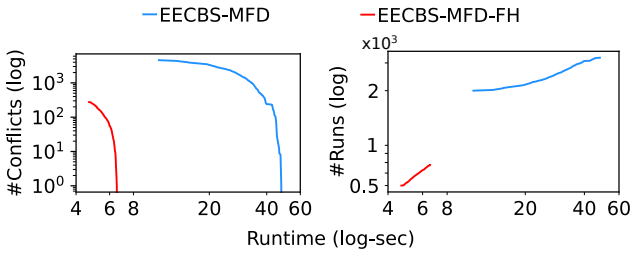


Figure 6: Starting from expanding the root CT node when solving the same MAPF instance as Figure 5, (a) the number of conflicts of the expanded CT node versus runtime, and (b) the cumulative number of low-level focal search runs.

than other approaches and thus reduces the number of low-level focal search runs #Runs to find a w-optimal solution. Meanwhile, our FH results in a lower runtime T' than other approaches in calculating guidance heuristics. Also, since we deploy strategy (S2) by using (F)BCS, we can reuse the simulated paths from FBGF as the paths for the root CT node for EECBS, resulting in a lower runtime T_0 in generating the root CT node than other approaches. As for strategy (S3), we show in Table 2 that increasing the path-found threshold $k_{\max}$ can significantly increase the runtime of FBGF, especially in large graphs like `city` and `den520d`, where the runtime of generating FH can take most of the 60-second time budget when considering 2000 agents. Figure 5 visualizes the center of a MAPF instance with 2000 agents on `den520d` graph. With the guidance from our FH, the paths from EECBS-MFD-FH avoid conflicts by following the flows, resulting in dark-blue lines with higher values on their heatmap (Figure 5(c)). The paths also avoid target conflicts due to our strategy (S1), resulting in blank grids in their heatmap. When guided by our FH in the root CT node, the number of conflicts significantly reduces from 4838 to 287, where the number of target conflicts reduces from 4653 to 179. When solving this MAPF instance, due to fewer conflicts and reusing simulated paths in the root CT node, EECBS-MFD-FH, with fewer low-level focal search runs, finds a w-optimal solution even before EECBS-MFD starts its expansion, as shown in Figure 6.

7 Conclusion

We presented the Flow-Based Guidance Framework, a general pre-processing technique that generates FH to guide agents for speeding up EECBS. Combined with flex distribution and our strategies targeting w-optimal MAPF, our Flow-Based Guidance Framework uses the (Flexible) Bounded-Cost Search to simulate paths for agents and then calculates FH based on the flow of the simulated paths. We show that guiding agents via FH can significantly reduce collisions and accelerate the search. The empirical evaluation shows that guiding agents via our FH outperforms state-of-the-art approaches in terms of success rate within 60 seconds, yielding w-optimal solutions close to optimal. Future work includes developing a more advanced guidance framework to efficiently adaptively fine-tune the heuristics.

8 Acknowledgments

The research at the University of California, Irvine and the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 2544613, 2434916, 2321786, 2112533, as well as gifts from Amazon Robotics and the Donald Bren Foundation.

References

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Tolpin, D.; Betzalel, O.; and Shimony, S. E. 2015. ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 740–746.

Chan, S.-H.; Li, J.; Gange, G.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2022. Flex Distribution for Bounded-Suboptimal Multi-Agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 9313–9322.

Chan, S.-H.; Phan, T.; Li, J.; and Koenig, S. 2025. New Mechanisms in Flex Distribution for Bounded Suboptimal Multi-Agent Path Finding. In *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, 47–55.

Chen, Z.; Harabor, D.; Li, J.; and Stuckey, P. 2024. Traffic Flow Optimisation for Lifelong Multi-Agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 20674–20682.

Cohen, L.; Uras, T.; Kumar, T. K. S.; Xu, H.; Ayanian, N.; and Koenig, S. 2016. Improved Solvers for Bounded-Suboptimal Multi-Agent Path Finding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 3067–3074.

Han, S. D.; and Yu, J. 2022. Optimizing Space Utilization for More Effective Multi-Robot Path Planning. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 10709–10715.

Haslum, P. 2013. Heuristics for Bounded-Cost Search. In *Proceedings of the Twenty-Third International Conference on International Conference on Automated Planning and Scheduling (ICAPS)*, 312–316.

Ho, F.; Salta, A.; Geraldes, R.; Goncalves, A.; Cavazza, M.; and Prendinger, H. 2019. Multi-Agent Path Finding for UAV Traffic Management. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 131–139.

Li, J.; Gange, G.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2020. New Techniques for Pairwise Symmetry Breaking in Multi-Agent Path Finding. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 193–201.

Li, J.; Hoang, T. A.; Lin, E.; Vu, H. L.; and Koenig, S. 2023. Intersection Coordination with Priority-Based Search for Autonomous Vehicles. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 11578–11585.

Li, J.; Ruml, W.; and Koenig, S. 2021. EECBS: Bounded-Suboptimal Search for Multi-Agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 12353–12362.

Pearl, J.; and Kim, J. H. 1982. Studies in Semi-Admissible Heuristics. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 392–399.

Russell, S.; and Norvig, P. 2010. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, 151–159.

Thayer, J. T.; and Ruml, W. 2011. Bounded Suboptimal Search: A Direct Approach Using Inadmissible Estimates. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 674–679.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. In *AI Magazine*, 9–20.

Yu, J.; and LaValle, S. M. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 1443–1449.

Zhang, Y.; Jiang, H.; Bhatt, V.; Nikolaidis, S.; and Li, J. 2024. Guidance Graph Optimization for Lifelong Multi-Agent Path Finding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 311–320.