On Merging Agents in Multi-Agent Pathfinding Algorithms

Eli Boyarski,[1] Shao-Hung Chan,[2] Dor Atzmon,[1] Ariel Felner,[1] Sven Koenig[2]

[1] Ben-Gurion University of the Negev,
[2] University of Southern California
boyarske@post.bgu.ac.il, shaohung@usc.edu, dorat@post.bgu.ac.il, felner@bgu.ac.il, skoenig@usc.edu

Abstract

In Multi-Agent Pathfinding (MAPF), the task is to find non-colliding paths for a set of agents. This paper focuses on search-based MAPF algorithms from the Conflict-Based Framework, which is introduced here. A common technique in such algorithms is to merge a group of dependent agents into a meta-agent and plan non-colliding paths for the meta-agent using a low-level MAPF sub-solver. We analyze the patterns that emerge when agents are merged in an arbitrary order. We then introduce policies for choosing which agents or meta-agents to merge to achieve improved efficiency in three algorithms: Independence Detection (ID) and Improved Conflict-Based Search (ICBS), which are optimal, and Priority-Based Search (PBS), which is a fast suboptimal algorithm. Experimental results show a significant improvement in efficiency.

Introduction and Overview

The task in a Multi-Agent Pathfinding (MAPF) problem is to find a set of collision-free paths for a set of agents, each from its start location to its designated goal location, often while minimizing a cost function. The most common cost function is the *sum-of-costs* of the paths of the agents. MAPF is a well-known and well-studied topic with numerous real-world applications. For example, MAPF is a core challenge in automated warehouse logistics (Wurman, D'Andrea, and Mountz 2008), automated parcel sortation (Kou et al. 2020), automated valet parking (Okoso, Otaki, and Nishi 2019), computer games (Silver 2006) and a variety of other contexts (Ma et al. 2016). A survey on the difffrent variants of MAPF appeares in Stern et al. (2019). Many optimal and suboptimal approaches to solving MAPF are search-based; a summary is given by Felner et al. (2017).

While *Conflict-Based Search* (CBS) (Sharon et al. 2012a) is a specific search-based MAPF algorithm, in this paper we provide a general description of a framework for MAPF algorithms which we call the *Conflict-Based Framework* (CBF). CBF received its name because CBS is a commonly used algorithm that is a special case of it, but, as we will show, many other search-based MAPF algorithms fit the Conflict-Based Framework. CBF has two levels of search. High-level nodes contain a set of agents (or meta-agents) coupled with a set of *constraints* on the agents (with the constraints of CBS being a special case). A low-level sub-solver is then called to find paths for the agents that satisfy the constraints. Finally, the paths are validated. If conflicts are found, an *operator* is applied on the high-level node to create child nodes. Operators resolve the conflict by imposing more constraints. MAPF algorithms of the Conflict-Based Framework differ mainly in the operators that resolve conflicts and in the constraints they employ. CBS, as a special case of CBF, uses a *split* operator that generates two child nodes, where each child node imposes a constraint on one of the conflicting agents that forces it to avoid the location in conflict.

Another operator is the *merge* operator, which resolves conflicts by coupling two or more agents into a *meta-agent*. Agents in a meta-agent are solved as a coupled sub-problem from that point onward. Therefore, the paths of agents inside the same meta-agent never conflict. The merge operator was directly used in Independence Detection (ID) (Standley 2010), Meta-Agent Conflict-Based Search (MA-CBS) (Sharon et al. 2012b), and Improved Conflict-Based Search (Boyarski et al. 2015), all optimal MAPF algorithms. But, as we will show, a different type of merge operator is also used in Priority-Based Search (Ma et al. 2019), a suboptimal MAPF algorithm.

This paper focuses on search-based MAPF algorithms that employ a *merge* operator on agents and studies policies for merging. The merge operator was first suggested over a decade ago but, despite its strength, remained less researched than other MAPF techniques. The MAPF algorithms that use the merge operator only use simple policies for deciding when conflicting agents. However, no attention has been paid to the question of which agents to merge, given that multiple pairs of conflicting agents may fit the merge criterion. Usually, the first two agents that are found to conflict and fit the merge criterion are merged (Sharon et al. 2015; Standley 2010).

In this paper, we delve into this question. We suggest general policies for deciding which agents to merge. Our policies can be applied to any MAPF algorithm that uses a merge operator. The policy we recommend for optimal MAPF algorithms is to merge meta-agents with fewer agents before meta-agents with more agents, and among meta-agents with the same number of agents, to merge meta-agents that

have more conflicts with other agents before meta-agents that have fewer conflicts with other agents. For suboptimal MAPF solvers, in which the low-level sub-solver is very fast, our policy suggests that meta-agents with *more* agents should be merged before meta-agents with fewer agents. We analyze the advantages of our policies. We then experimentally show the generality of these policies and the speedups they provide over the previous policy on three MAPF algorithms that fit CBF, namely ICBS, ID, and PBS. Our new policies reduced the search effort by up to a factor of 6 and allowed the MAPF algorithms to solve many more instances within the time limit.

Definitions

The *Multi-Agent Pathfinding* (MAPF) problem (Stern et al. 2019) receives as input a graph $G = (V, E)$ and a set of n agents $A = \{a_1, \ldots, a_n\}$, where each agent $a_i \in A$ has a unique start vertex $s_i \in V$ and a unique goal vertex $g_i \in V$. A solution to MAPF is a set of paths $\Pi = \{\pi_1, \ldots, \pi_n\}$ for the agents, such that path $\pi_i \in \Pi$ for agent a_i starts in start vertex s_i and ends in its corresponding goal vertex g_i. A solution is *valid* iff it is conflict-free. There are two commonly considered types of conflicts: *vertex conflicts* and *swapping conflicts* (Stern et al. 2019). Let $\pi_i(t)$ denote the location of agent a_i on path π_i at timestep t. There is a vertex conflict $\langle \pi_i, \pi_j, v, t \rangle$ between two paths π_i and π_j iff both paths occupy vertex v at timestep t (i.e., $\pi_i(t) = \pi_j(t) = v$). There is a swapping conflict $\langle \pi_i, \pi_j, e, t \rangle$ between two paths π_i and π_j iff both paths traverse edge e in opposite directions between timesteps t and $t+1$ (i.e. $\pi_i(t) = \pi_j(t+1) \text{AND} \pi_i(t+1) = \pi_j(t)$). The cost $C(\pi)$ of path π is $|\pi| - 1$. The cost $C(\Pi)$ of a solution Π is the sum of the costs of its paths: $\sum_{\pi \in \Pi} C(\pi)$ (also known as *sum-of-costs*). A solution is *optimal* iff it is valid and has the lowest cost among all valid solutions.

The Conflict-Based Framework (CBF)

Conflict-Based Search (CBS) (Sharon et al. 2012a) is a prominent algorithm for optimally solving MAPF. In this section, we show that CBS can be generalized and describe a framework called the *Conflict-Based Framework* (CBF). We show that many existing MAPF algorithms are special cases of this framework.

The Conflict-Based Framework has two search levels: the *high level* and the *low level*. The high level builds a high-level tree (also known as the *Constraint Tree* (CT) (Sharon et al. 2012a)) of high-level nodes. Each high-level node N contains a set of *constraints* ($N.constraints$) on the agents and a set of paths ($N.\Pi$) for the agents that must satisfy the constraints of that node. The cost of node N is the cost of its paths ($C(N.\Pi)$). The high-level search begins at a root node with an empty set of constraints. Given a high-level node N, a *low-level* search is called to find paths that satisfy the set of constraints of N. After the low level returns paths that satisfy the constraints, the paths are checked for conflicts. If the paths are conflict-free, the node is declared as a goal node and the search halts. Otherwise, let $\langle \pi_i, \pi_j, x, t \rangle$ (x is either a vertex or an edge) be a conflict in node N. To resolve this conflict, MAPF algorithms that fit CBF apply an operator on N and generate children for N in the CT.

Algorithms in CBF differ in two aspects: (1) in the operators they apply (e.g., split or merge) to resolve conflicts (where each operator is associated with its own type of constraints), and (2) in the way they search the high-level tree (Best-first search or Depth-first search). We next cover MAPF algorithms that are special cases of CBF.

Conflict-Based Search

The operator used by Conflict-Based Search (CBS) algorithm (Sharon et al. 2012a) is called *split*. When a conflict $\langle \pi_i, \pi_j, x, t \rangle$ is found in a high-level node N, the split operator creates two high-level child nodes N_i and N_j. The classical version of CBS imposes the constraint $\langle \pi_i, x, t \rangle$ on N_i and the constraint $\langle \pi_j, x, t \rangle$ on N_j, where the constraint $\langle \pi_i, x, t \rangle$ prohibits agent a_i (analogously for agent a_j) from occupying vertex x at timestep t (or edge x between timesteps t and $t+1$). This retains optimality, since the two child nodes together allow any solution that their parent node N allowed. Improved CBS (ICBS) (Boyarski et al. 2015) specifies which conflict should be resolved. ICBS finds *all* conflicts among the paths in a node. It prioritizes resolveing the first *cardinal conflict* that it encounters over *semi-cardinal* conflicts over *non-cardinal* conflicts.

As in classical search, the high-level tree must be explored in a *best-first search* manner, according to the costs of the nodes to attain optimality. Felner et al. (2018) provided an admissible heuristic function for high-level nodes and performed an A*-like search on the high-level tree while preserving optimality. This heuristic function was further improved by Li et al. (2019a) and later by Boyarski et al. (2021). Similarly to IDA* (Korf 1985), it is also possible to find optimal solutions by performing an iterative-deepening search on the high-level tree (Boyarski et al. 2020).

Li et al. (2019c) showed that, on grids, constraints can be enlarged. This can be done by, first, identifying a rectangle of cells in which two agents conflict on any pair of their shortest paths. Then, to prevent the conflict on all cells of the rectangle, a *barrier constraint* is set on each of the agents. A barrier constraint contains a set of cells that prohibits either of the agents from occupying the rectangle.

Instead of prohibiting one of the two agents from occupying a contested vertex (or traversing an edge), Li et al. (2019b) showed that it is possible to resolve a conflict by creating two new high-level nodes in which one node prohibits an agent from occupying a vertex and the other node prohibits all other agents from occupying it. Thus, no solution is allowed by both child nodes, which avoids duplicate work in handling invalid solutions.

Independence Detection

The Independence Detection algorithm (ID) (Standley 2010) predated CBS by a few years but, in fact, is also a special case of CBF. ID does not use the split operator to resolve conflicts; it only uses a merge operator. In the root node, the low level finds individual optimal paths for the agents. When a conflict is found in node N between agents a_i and a_j, a merge operator is applied on the conflicting

agents and the merge constraint $\langle a_i, a_j, \cup \rangle$ is added. Logically, the merge operator creates a new high-level child node N' with this new constraint. The merge constraint prevents agents a_i and a_j from having any conflicts. Given a set of merge constraints, all agents are divided into groups of agents $\{a_1, \ldots, a_m\}$ such that any two agents that have a merge constraint between them belong to the same group. Each such group is called *meta-agent*. The agents in each meta-agent are coupled together and the low-level search must find optimal paths for these agents without any conflicts (and thus the conflict that was found for N is resolved). It is common to apply a variant of A* for the low-level solver to find an optimal solution for the meta-agent. Each state in the search space in the low-level search contains the location of every agent in the MAPF problem instance at a certain timestep, and states may contain only non-conflicting configurations of agents.

Two meta-agents that have a conflict between any pair of their individual agents can also be merged. So, in ID, one can recognize the high-level tree as a chain of nodes, each of them merges more agents until a conflict-free solution is found for all (meta-) agents.

Meta-Agent Conflict-Based Search

Meta-Agent Conflict-Based Search (MA-CBS) (Sharon et al. 2012b) is an optimal algorithm that combines the split and merge operators. When a conflict is found between two agents, MA-CBS employs a policy that decides whether to use the merge or the split operators. The typical policy used is to apply the merge operation if the number of conflicts between the two (meta-) agents exceeds a given threshold B. Otherwise, a split operator is applied, as in CBS. Different values for B were used by Sharon et al. (2012b). In fact, ID can be seen as a special case of MA-CBS with $B = 1$; that is, merge is applied as soon as a conflict is found and split is never used. At the other extreme, CBS uses $B = \infty$; it always uses the split operator and never uses merge. It is important to note that merge is permanent. Once agents are merged they are never separated. *Improved CBS* (ICBS) (Boyarski et al. 2015) is an enhanced version of MA-CBS. Its authors showed that it is beneficial to *restart* the high-level search when agents are merged, with the merged agents as meta-agents at the root node, since restarting the high-level search guarantees that a pair of agents would not be merged multiple times at different places in the CT.

Other A*-Based Algorithms

A*-based algorithms, such as A*+OD (Standley 2010) and EPEA* (Goldenberg et al. 2014), can also be seen as degenerate members of CBF. We set $B = 0$ and all agents are merged already in the root node into a single meta-agent, which is solved in a coupled manner by an A*-based search.

Non-CBF MAPF Algorithms

The M* algorithm (Wagner and Choset 2015) uses a different framework for merging agents, called *subdimensional expansion*, which only merges agents locally when a conflict is found. The subdimensional expansion locally merges agents in specific map locations. Similarly to the merge operator described above, once agents are locally merged in M*, they are never unmerged. M* and its variants (rM*, ODrM*) are not covered in this paper, but can also benefit from our guidelines.

Other search-based algorithms, such as Increasing Cost Tree Search (ICTS) (Sharon et al. 2013) are not members of the Conflict-Based Framework. At its high level, ICTS searches the increasing cost tree (ICT). Every node in the ICT consists of a k-ary vector $[C_1, \ldots, C_k]$ which represents all possible solutions in which the path cost of agent a_i is C_i. The root node of the ICT is $[opt_1, \ldots, opt_k]$, where opt_i is the cost of the shortest individual path for agent a_i. A child node in the ICT is generated by increasing the path cost of one of the agents by 1. The ICT is searched in breadth-first order until a goal node is found. An ICT node is a goal node if there is a non-conflicting solution with matching costs. The low level of ICT performs this goal test.

Priority-Based Search and Confict-Based Search With Priorities

Priority-Based Search (PBS) (Ma et al. 2019) is a new suboptimal member of CBF. Recently, an application for PBS for multi-arm assembly systems has been suggested (Chen et al. 2022). PBS employs a *prioritized merge* operator. When two agents a_i and a_j have a conflict in a high-level node N they are prioritized-merged: Two new high-level nodes N' and N'' are created, one with the prioritized-merge constraint $\langle a_i, a_j, \prec \rangle$ and the other one with $\langle a_j, a_i, \prec \rangle$. The meaning of this constraint is that these two conflicting agents have a *priority order* ($\prec$) between them; $a_i \prec a_j$ denotes that agent a_i has higher priority than agent a_j. The priority is effected in the low-level, when the path of agent a_j is planned under the constraint that it may not collide with the paths of higher-priority agents, including that of a_i.

This planning task at the low level of PBS is a type of a *Prioritized Planning* (PrP) task (Erdmann and Lozano-Perez 1986), where plans are planned sequentially in decreasing order of priorities of the agents, and each must avoid conflicts with the plans of agents of a higher priority. The sub-solver that PBS uses is Cooperative A* (CA*) (Silver 2005), which receives the order in which to plan paths for agents from the high level. CA*is a CBF algorithm which uses PrP. In CA*, all agents are prioritized merged at the root node, such that each two agents receive a priority order between them. Thus, a total priority order on all agents is created. Obviously, such conflict resolution is not optimal, however CA* is commonly used in practice because it is easy to understand and implement.

PBS performs a depth-first search on the high-level tree. Out of the two new high-level nodes, PBS first explores the child node with the lower cost.

A meta-agent in PBS can be seen as a weakly connected component in a *priority graph*. A priority graph is a directed acyclic graph (DAG), in which the vertices are agents and a directed edge denotes the agent at its source has priority over the agent at its end. Every time a priority order between two different meta-agents is defined (a new edge

connects two previously-separate weakly connected components), their meta-agents are merged.

Differently from ICBS and from ID, not all of the paths of the meta-agent are necessarily replanned when it is created. In PBS, CA* only replans the paths of agents from the lower-priority meta-agent that have a conflict with a higher-priority agent. Another distinction is that a single merge operation in PBS might not leave a meta-agent fully conflict-free. Sometimes, some internal conflicts remain between agents in the same meta-agent. Such conflicts are resolved in later branch operations of PBS.

Ma et al. (2019) also proposed CBS with Priorities (CBSw/P). CBSw/P behaves more similarly to CBS than PBS, with the addition of maintaining a partial priority order in every node. Each time a conflict $\langle a_i, a_j, v, t\rangle$ is resolved in high-level node N, the child node N' that receives the additional constraint $\langle a_i, v, t\rangle$ also receives the additional priority order $a_j \prec a_i$. High-level nodes below N' are only generated if their accumulated partial priority order contains no contradiction. Unlike in PBS, conflicts between the same two individual agents may still occur after the first conflict between them is resolved.

We next study the merge operator and suggest our policies for merging.

The Merge Operator

Merging agents into a meta-agent has its tradeoffs. It reduces complexity in the high level because the high level now deals with fewer agents. Once (meta-)agents are merged, their paths will never conflict again; this reduces the number of future high-level conflicts to resolve, thus reducing the size of the high-level tree. However, the low level has to deal with more agents so its complexity increases, and replanning their joint paths in the future would necessarily be more costly than planning them individually. Given a meta-agent with k agents, there are now $O(|V|^k)$ states in the low-level search problem. Given two conflicting meta-agents with k_1 and k_2 agents, resolving the conflict with a split operator results in solving two search problems of $O(|V|^{k_1})$ and $O(|V|^{k_2})$ states. Resolving the same conflict with the merge operator results in a single search problem of $O(|V|^{k_1+k_2})$ states. However, the merge operator resolves all conflicts between the two meta-agents, while the split operator may leave other unresolved conflicts between the two meta-agents.

As mentioned, the merge operation has hardly been researched in the past. Merge should be applied thoughtfully by addressing two questions: (1) whether to apply the merge operator, and, (2) given a set of candidate agent-pairs to merge, which pair to merge. The first question has been researched in MA-CBS and further work on it is beyond the scope of this paper. We next introduce a number of policies for answering the second question.

First Policy

The authors of ID, MA-CBS, and PBS did not provide guidelines regarding which agents-pair to merge (or which conflict to address). These algorithms originally assumed that the *first* conflict encountered in the high-level node (during a simulation of the execution of the plans of the agents)

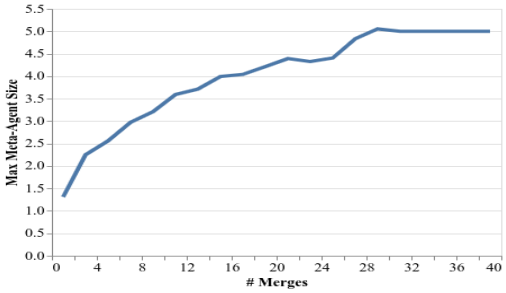
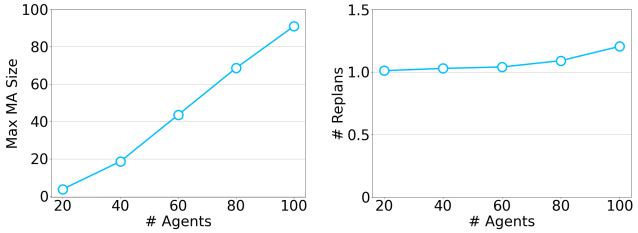


Figure 1: Average size of the largest meta-agent at the end of the search as a function of the number of merge operations that were performed by baseline ID+EPEA*



(a) Average size of the largest meta-agent at the end of the search for baseline PBS as a function of the number of agents in random-32-32-20 MAPF instances

(b) Average number of paths that were replanned in each high-level non-root node of PBS as a function of the number of agents in a random-32-32-20 MAPF instance

Figure 2: Two figures for PBS

is the one to be resolved. MA-CBS resolved the conflict with a merge operation if the number of previously-seen conflicts between the two agents is larger than the threshold B, and with a split operation otherwise. Denote this the *First* policy.

The arbitrary choice to resolve the first encountered conflict causes an undesirable behavior when merging agents. Large meta-agents occupy more locations, thus tend to have more conflicts than small meta-agents which occupy fewer locations. Consequently, large meta-agents are more likely to be chosen to be merged, producing an even larger meta-agent. Eventually, there will be a few large meta-agents, while the rest of the meta-agents are of size 1 or 2. Figure 1 shows the average size of the largest meta-agent at the end of the search as a function of the number of merge operations that were performed during the search when running ID+EPEA* on 56,254 standard MAPF benchmark instances (see the Experimental Results section for details). The first 15 merge operations are quite likely to increase the size of the largest meta-agent. After 28 merges, the largest meta-agent isn't likely to grow larger under a 1-minute time limit, either because there are no more meta-agents that can collide with the largest meta-agents and have not already been merged with it, or simply because the solver times out.

This tendency to create large meta-agents delegates a large part of the MAPF problem to the low-level solver. Thus, it burdens optimal CBF algorithms because their low

level solves a hard task - solving a MAPF sub-problem optimally. In contrast, in suboptimal CBF algorithms, the low-level solver is very fast, so as much of the work as possible should be delegated to it.

To demonstrate how efficient the low level of PBS is, we present two figures. Figure 2a shows the average size of the largest meta-agent at the end of the search as a function of the number of agents in problem instances using the random-32-32-20 map. Despite the fact that the largest meta-agent includes almost all the agents in this problem instance, Figure 2b indicates that the average number of individual paths that were replanned in each non-root high-level PBS node does not grow much above 1. The CA* sub-solver rarely needs to replan more than one individual path, even for very large meta-agents.

We next introduce new policies for selecting which pair of agents to merge (among candidate pairs). The candidate pairs for this selection remain the same, namely each pair of conflicting meta-agents in ID or in PBS, and each pair of meta-agents that have had at least B previous conflicts in ICBS. Later, we experimentally demonstrate that our policies indeed improve the efficiency of the algorithms.

Most-Conflicting Smallest (MCS) Policy

For optimal CBF algorithms, we suggest the following policy, which we denote as the *Most-Conflicting Smallest* (MCS) policy.

First, find all the pairs of conflicting meta-agents that should be merged according to the merge criterion of the algorithm. For example, in ICBS a pair of conflicting agents should be merged if, given the merging threshold B, at least B separate conflicts between them have been encountered during the high-level search. Then, among these pairs find the pairs whose combined size is smallest (it may be a single pair, or several pairs of the same combined size). This 'myopic' choice does not guarantee minimizing the maximum size of a meta-agent at the end of the search, but it is simple and effective.

Second, among the pairs of conflicting meta-agents that were found, choose the pair that has the largest number of conflicts with other meta-agents (most-conflicting). The rationale for this choice is that every time two meta-agents are merged and a new path is computed for the resulting merged meta-agent, 'old' conflicts with other meta-agents may disappear for the new path (while other conflicts may emerge). Such indirect conflict resolutions are desirable since they dispense with the need for directly resolving the conflicts (e.g., via another merge operation). So, as a secondary consideration, MCS tries to maximize the potential for indirect conflict resolutions.

Least-Conflicting Largest (LCL) Policy

For comparison, we also define the *Least-Conflicting Largest* (LCL) policy, the opposite of the MCS policy, for optimal CBF algorithms. Its rules for choosing the pair of agents to merge are: (1) Find all pairs of conflicting agents that have the maximum combined size. (2) Among them, find all pairs that have the least conflicts with other agents. With optimal algorithms, LCL is expected to perform worse than the arbitrary choice of the First policy.

Balanced (BAL) Policy

Finally, we define the *Balanced* (BAL) policy for optimal CBF algorithms. It attempts to balance between favoring agent pairs that have more conflicts with other agents with favouring agent pairs that result in smaller meta-agents. BAL uses the formula

$$\arg\max_{\text{pair}} \frac{\#\text{external–conflicts(pair)}}{2^{\text{combined–size(pair)}}}$$

to select the pair of agents to merge among all candidate pairs. The formula assigns an exponential weight to the combined size of the pair since the number of agents exponentially affects the computation time of the sub-solver.

Optimization note: When meta-agents a_x and a_y, with respective costs C_x and C_y, are merged to create meta-agent a_z, some facts are known about its cost C_z even before its path is computed. In a_z, the combined cost of the paths of the agents originally from a_x must still be at least C_x because they are more constrained in a_z (they need to also avoid conflicts with the paths of the agents originally from a_y). The same holds for a_y, of course (therefore $C_z \geq C_x + C_y$). This information can be used to speed up the low-level search for meta-agent a_z, and subsequent paths of meta-agents it is merged into. For example, for A*-based MAPF algorithms, it can be used to improve the heuristic estimate for some search nodes. The heuristic value of a search node, where the combined f-value for agents from a_x is $C_x - k$, can be increased by k. This optimization had already been in place in the original implementation of MA-CBS (Sharon et al. 2015), but had not been reported previously. To be consistent with experiments in prior work, we also used it in the ID and ICBS solvers in our experiments.

Merge Policies For PBS

We now present merge policies for suboptimal CBF algorithms. First, it can happen that after the last merge operation, the resulting meta-agent still contains internal conflicts between its constituent agents. In this case, we call it 'not fully merged'. As an example, Figure 3 shows a priority graph with six agents: D, E and F, which form one meta-agent, and X, Y, and Z which form another meta-agent. E has higher priority than D and F, and X and Z have higher priorities than Y. Both meta-agents can contain an internal conflict, shown with a curly line.

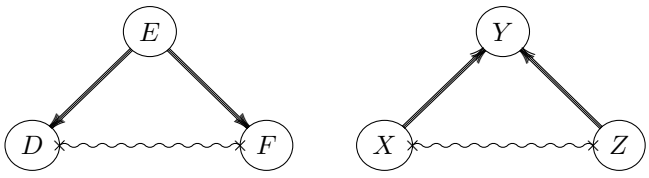


Figure 3: A priority graph for a PBS node with two meta-agents, each containing three agents and an internal conflict

In both merge policies below, if the last meta-agent that was created is still not fully merged, one of its pairs of conflicting constituent agents should be chosen and its conflict should be resolved by assigning an internal order between them. This choice does not increase the size of the meta-agent and ensures that the merge policies below can assume that meta-agents are internally conflict-free.

With suboptimal solvers, delegating as much work as possible to the low level is desirable because the low level is very fast. On the other hand, checking the plans of all pairs of agents for conflicts takes a relatively large part of the runtime. Therefore, we define the following policies for suboptimal algorithms. The *Largest* policy selects the pair of conflicting meta-agents with the largest combined size for merging, unless there are still internal conflicts within the last meta-agent that was created by merging. The *Smallest* policy selects a pair of conflicting meta-agents with the smallest combined size for merging.

We briefly describe how the new policies are implemented next.

Algorithm Modifications

Algorithm 1 shows the high-level of ICBS, with use of heuristics for high-level nodes (Felner et al. 2018). Note the only modifications required to enable choosing the agents to merge are in line 6 and in lines 9 to 11, where instead of resolving any conflict of the highest priority, all highest priority conflicts are found, and if any of them merit a merge, a policy is explicitly used to choose one.

Modifying ID and PBS to explicitly choose the pair of agents whose conflicts would be resolved via a merge is simple. The necessary changes are similar to those made to ICBS in Algorithm 1.

Algorithm 1: High-level of ICBS with heuristics and choice of agents to merge

```
1  Main(MAPF problem instance)
2      Init node R, with initial paths for the (meta-)agents
3      insert R into OPEN
4      while OPEN not empty do
5          N ← node with lowerst f-value from OPEN
6          Simulate the paths in N and find all conflicts
7          if N has no conflict then
8              return N.solution
9          C̄ ← AllHighestPriorityConflicts(N)
10         if ShouldMergeAny(C̄) then
11             a_i, a_j ← ChooseMetaAgentsToMerge(C̄)
12             a_ij = merge(a_i,a_j)
13             if MergeAndRestart enabled then
14                 Restart high-level search with agents merged
15             Update N.constraints
16             Update N.solution by invoking low level(a_ij)
17             Update N.NC, N.g, N.h, andN.f
18             Insert N back into OPEN
19             continue // Go back to the while statement
20         C ← Conflict ⟨a_i, a_j, v/e, t⟩ from C̄
21         Children ← ∅
22         foreach agent a_i in C do
23             A ← GenerateChild(N, ⟨a_i, v/e, t⟩)
24             if (A.g = N.g) and (A.NC < N.NC) then
25                 (N.solution, N.NC, N.h, N.f) ←
                   (A.solution, A.NC, A.h, A.f)
26                 Insert N back into OPEN
27                 Children ← ∅
28                 break
29             else
30                 Insert A into Children
31         Insert Children into OPEN
32 GenerateChild(Node N, Constraint C = ⟨a_i, v/e, t⟩)
33     A.constraints ← N.constraints + ⟨a_i, v/e, t⟩
34     A.solution ← N.solution
35     Update A.solution by invoking low level(a_i)
36     A.NC ← number of conflicts in A.solution
37     A.g ← C(A.solution)
38     A.h ← h(A)
39     A.f ← A.g + A.h
40     return A
```

Experimental Results

Our experiments with ID and ICBS, implemented in C#, were run on a Windows laptop with a 2.60GHz Intel® Core™ i7-6700HQ processor and 16GB of RAM. PBS was implemented in C++, and its experiments were run on CentOS Linux on an AMD EPYC 7302 16-Core Processor with a memory limit of 16 GB of RAM.

In our experiments, we used the following tie-breaking strategy: if all previous considerations still leave multiple candidate pairs of agents,use the First policy to choose among them. This tie-breaking strategy avoids arbitrary differences between the policies.

Results for ID

To evaluate the performance of ID with different merge policies, we experimented on the standard MAPF benchmarks (Stern et al. 2019), which contain 32 grids with different attributes (city maps, grids with random obstacles, mazes, warehouse maps, etc.). With each grid, we used the 25 'even' scenarios that specify the start and goal locations for up to 7,000 agents. We formed the first problem instance with the first two agents in the scenario, the second problem instance from the first three, and continued solving problem instances with increasing numbers of agents until we reached a problem instance the solver failed to solve under the runtime limit of 60 seconds. For higher numbers of agents, the solver was implicitly considered to have failed. We refer to MAPF instances on which any solver ran for more than half of the allotted time as *hard instances*.

Table 1 reports results for the following solvers: ID+EPEA* solvers with the following merge policies: *First*, *Least-Conflicting Largest*, *Most-Conflicting Smallest*, and *Balanced (BAL)*. We report the number of MAPF instances and the number of hard MAPF instances that were solved successfully by each solver. In total, out of 56,254 problem instances, 49,842 instances were solved by all solvers, and 55,462 instances were solved by at least one solver. As ex-

Group	#instances	First	LCL	MCS	BAL
All	56,254	52.3	52.2	53.1	**53.5**
Hard	6,412	4.2	4.1	5.0	**5.4**

Table 1: Number (x 1000) of MAPF instances solved and hard MAPF instances solved by ID+EPEA* with different merge policies: First, LCL, MCS, and BAL.

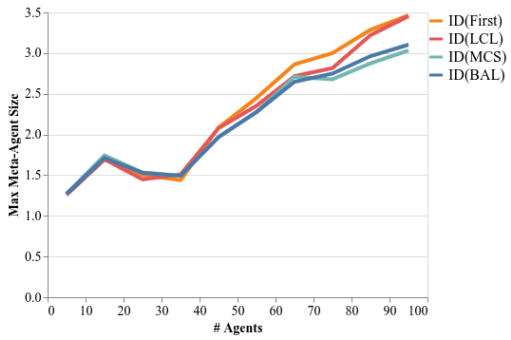


Figure 4: Average size of the largest meta-agent at the end of the search as a function of the number of agents in room maps from the MAPF benchmark

pected, ID with the LCL policy performs worse than ID with the original arbitrary First policy. Both the ID solvers with the MCS and with the BAL policies performed better than the ID solver with the First policy, especially on hard instances. ID with the BAL policy performed best, solving almost 30% more hard instances than ID with the First policy.

Figure 4 shows the average size of the largest meta-agent at the end of the search for every solver described above as a function of the number of agents in the problem instances using room maps from the MAPF benchmark. As could be expected, the ID solvers with the MCS and BAL policies created a smaller maximum size meta-agent on average than the solvers with the FIRST and LCL policies. Interestingly, the size of the largest meta-agent created by the ID solver with the LCL policy is often smaller on average than that of the ID solver with the First policy. It is also interesting to note that the BAL policy sometimes has the smallest average, even though it does not optimize for smallest size directly. This shows that taking into account the number of conflicts that can be indirectly resolved may eventually lead to fewer conflicts and, as a result, smaller meta-agents.

On hard problem instances that were successfully solved by all solvers described above, the average number of generated low-level nodes was generally similar for all solvers, with BAL leading its solver to generate 20% fewer nodes on average than the solver with the First policy on problem instances with 260 agents.

Results for ICBS

Table 2 shows the average runtime in seconds for six ICBS solvers: two with the merge threshold B set to 5, two with $B = 10$, and two with $B = 15$. Each pair has one solver with the First merge policy and one with the Most Conflicting Smallest policy. Failure to solve was counted as using the full 60 seconds time limit. ICBS restarts the search after every merge, so the number of external conflicts that the constituent agents of the chosen conflict have is of secondary concern, since all paths will be replanned at the new root node. For this reason, we did not include a solver with the BAL policy. Each row contains the average runtime for 'even' problem instances from the standard MAPF benchmark with a given number of agents. For almost all merge thresholds, maps and numbers of agents, the solver with the MCS policy ran faster on average, up to 45% in some cases.

	ICBS(5)		ICBS(10)		ICBS(15)	
Scenario and # Agents	First	MCS	First	MCS	First	MCS
empty-16×16, 30 agents	26.7	**20.4**	12.7	**7.7**	8.6	**4.8**
empty-16×16, 40 agents	53.9	**46.6**	39.9	**36**	38.6	**33**
empty-16×16, 50 agents	60	**58.2**	55.7	**54.9**	57.5	**52.6**
random-32-32-10, 50 agents	38.7	**35.3**	**26.8**	30.6	**24.5**	26.1
random-32-32-10, 70 agents	60.0	**58.2**	58	**52.4**	57.6	**55.4**

Table 2: Average runtime (s) of ICBS solvers with the First and MCS policies over various types of problem instances.

Results for PBS

Figure 5 (top) shows the success rate and average runtime in seconds for three PBS solvers with the First,[1] Smallest and Largest merge policies on the original problem instances from (Ma et al. 2019)[2]. Those problem instances include four maps: an empty 20 × 20 grid, a 20 × 20 grid with 10% randomly-placed obstacles, and two large video game grids (brc202d and lak503d). A timeout failure was treated as solving the problem instance exactly at the end of the 60-second time limit. In addition, Figure 5 (bottom) shows results for the same solvers on problem instances from the standard MAPF benchmark (Stern et al. 2019) on one maze map (maze-32-32-2), one warehouse map (warehouse-20-40-10-2-1), one room map (room-32-32-4) and one grid with randomly-placed obstacles (random-32-32-20).

Despite PBS being incomplete in theory, PBS with the Largest policy solved all of the original problem instances successfully in practice, improving the success rate of PBS with the Earliest policy by more than 40% on problems instances with 100 agents on the grid with randomly-placed obstacles map. Its success rate on problem instances from the MAPF benchmark was very high, significantly improving on the success rate of PBS with the baseline Earliest policy on problem instances from all maps except the warehouse map. The PBS solver with the *Largest* policy is the fastest on all small maps, achieving a speedup of up to about

[1] In our implementation, PBS with the First policy checked the paths of the agents for conflicts by iteratively checking them at every timestep, starting from timestep 1. For every high-level node, it randomized the order in which we checked the pairs of agents for conflicts at each timestep to avoid an unintended merge pattern where agents with low indices tended to be merged before agents with higher indices.

[2] Graciously provided by Hang Ma.

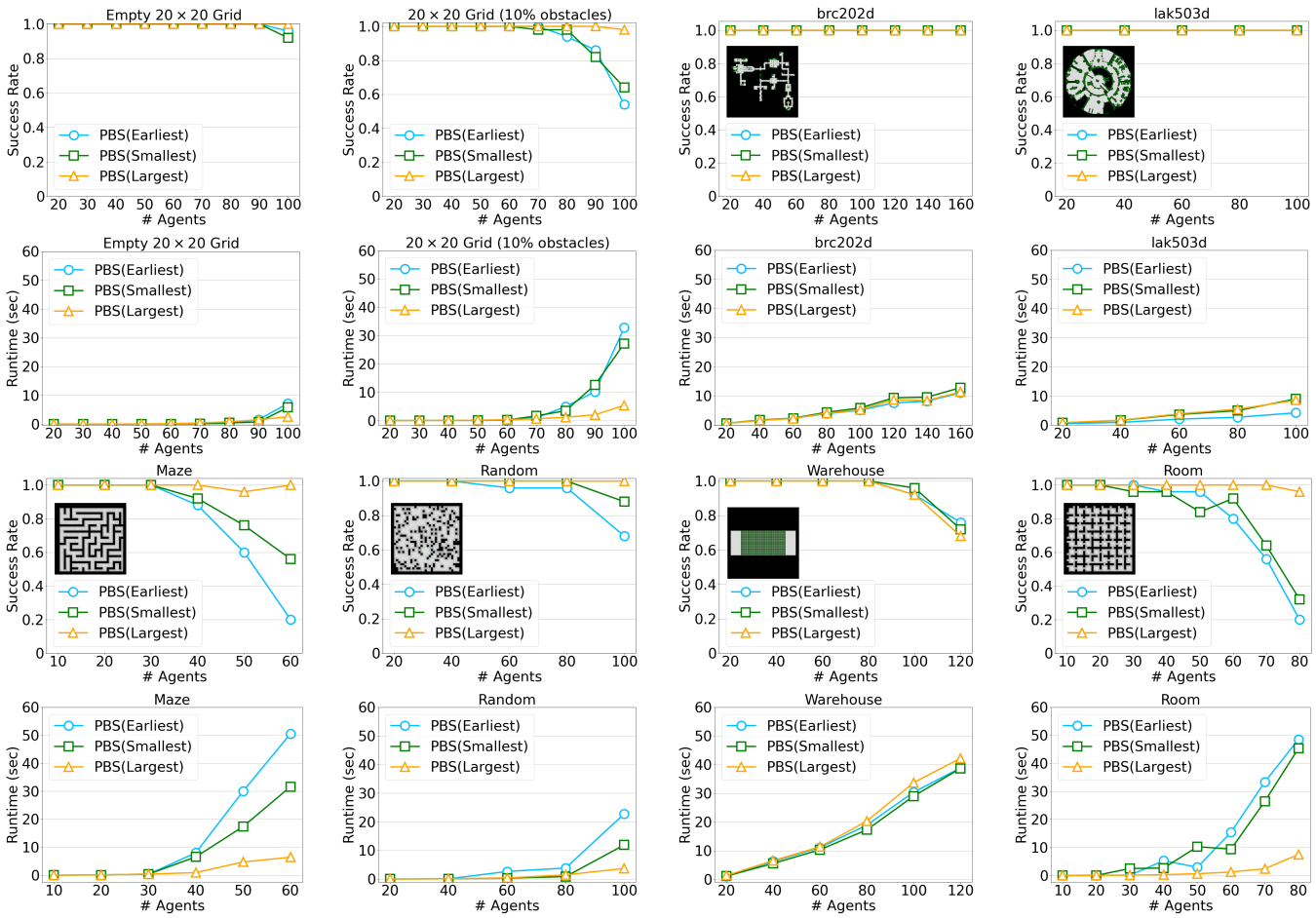


Figure 5: Success rate and average runtime in seconds as a function of the number of agents for PBS with the First, Smallest, and Largest policies on the original PBS MAPF problem instances and on additional problem instances from four standard scenarios: maze-32- 32-2, random-32-32-20, warehouse-10-20-10-2-1, and room-32-32-4

six times. On large maps, all solvers performed similarly. The sum-of-costs of the solution found by each of the PBS solvers for the same MAPF problem instance were always equal or almost equal.

Summary and Future Work

In this paper, we suggested several policies to be used in MAPF algorithms that use a merge operation. These policies choose the pair of agents to merge, among all eligible pairs of conflicting agents that fit the merging criterion. We demonstrated the effectiveness of these policies on three MAPF algorithms: ID, ICBS, and PBS. The significant improvement in the efficiency of PBS on small maps may allow PBS to replace prioritized planning in the initial stage of the novel MAPF-LNS2 algorithm (Li et al. 2022) in the future. Future research may also suggest combined policies that suggest both when to merge agents and which agents to merge.

Acknowledgments

This research was sponsored by the United States-Israel Binational Science Foundation (BSF) under grant numbers 2017692 and 2021643, and by Israel Science Foundation (ISF) under grant number 844/17. The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189, 1935712, and 21112533 as well as a gift from Amazon. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or the U.S. government.

References

Boyarski, E.; Felner, A.; Harabor, D.; Stuckey, P. J.; Cohen, L.; Li, J.; and Koenig, S. 2020. Iterative-Deepening Conflict-Based Search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-2020)*, 4084–4090.

Boyarski, E.; Felner, A.; Le Bodic, P.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2021. f-Aware Conflict Prioritization & Improved Heuristics For Conflict-Based Search. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI-2021)*, 12241–12248.

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Tolpin, D.; Betzalel, O.; and Shimony, E. S. 2015. ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-2015)*, 740–746. ISBN 9781577357384.

Chen, J.; Li, J.; Huang, Y.; Garrett, C.; Sun, D.; Fan, C.; Hofmann, A.; Mueller, C.; Koenig, S.; and Williams, B. C. 2022. Cooperative Task and Motion Planning for Multi-Arm Assembly Systems. *arXiv preprint arXiv:2203.02475*.

Erdmann, M. A.; and Lozano-Perez, T. 1986. On Multiple Moving Objects. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-1986)*, 3: 1419–1424.

Felner, A.; Li, J.; Boyarski, E.; Ma, H.; Cohen, L.; Kumar, T. K. S.; and Koenig, S. 2018. Adding Heuristics to Conflict-Based Search for Multi-Agent Path Finding. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-2018)*, 83–87.

Felner, A.; Stern, R.; Shimony, S. E.; Boyarski, E.; Goldenberg, M.; Sharon, G.; Sturtevant, N.; Wagner, G.; and Surynek, P. 2017. Search-Based Optimal Solvers for the Multi-Agent Pathfinding Problem: Summary and Challenges. In *Proceedings of the Annual Symposium on Combinatorial Search (SoCS-2017)*, 29–37.

Goldenberg, M.; Felner, A.; Stern, R.; Sharon, G.; Sturtevant, N.; Holte, R.; and Schaeffer, J. 2014. Enhanced Partial Expansion A*. *Journal of Artificial Intelligence Research (JAIR)*, 50: 141–187.

Korf, R. 1985. Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence*, 27(1): 97–109.

Kou, N. M.; Peng, C.; Ma, H.; Kumar, T. K. S.; and Koenig, S. 2020. Idle Time Optimization for Target Assignment and Path Finding in Sortation Centers. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI-2020)*, 9925–9932.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2022. MAPF-LNS2: Repairing Multi-Agent Path Finding via Large Neighborhood Search. In *To appear in Proceedings of the AAAI Conference on Artificial Intelligence (AAAI-2022)*.

Li, J.; Felner, A.; Boyarski, E.; Ma, H.; and Koenig, S. 2019a. Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-2019)*, 442–449.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2019b. Disjoint Splitting for Multi-Agent Path Finding with Conflict-Based Search. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-2019)*, 279–283.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2019c. Symmetry-Breaking Constraints for Grid-Based Multi-Agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI-2019)*, 6087–6095.

Ma, H.; Harabor, D.; Stuckey, P. J.; Li, J.; and Koenig, S. 2019. Searching with Consistent Prioritization for Multi-Agent Path Finding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI-2019)*, 7643–7650.

Ma, H.; Koenig, S.; Ayanian, N.; Cohen, L.; Hönig, W.; Kumar, T. K. S.; Uras, T.; Xu, H.; Tovey, C.; and Sharon, G. 2016. Overview: Generalizations of Multi-Agent Path Finding to Real-World Scenarios. In *IJCAI-16 Workshop on Multi-Agent Path Finding*.

Okoso, A.; Otaki, K.; and Nishi, T. 2019. Multi-Agent Path Finding with Priority for Cooperative Automated Valet Parking. In *Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC-2019)*, 2135–2140.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. 2012a. Conflict-Based Search for Optimal Multi-Agent Path Finding. In *Proceedings of the Conference on Artificial Intelligence (AAAI-2012)*, 563–568.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. 2012b. Meta-Agent Conflict-Based Search for Optimal Multi-Agent Path Finding. In *Proceedings of the Annual Symposium on Combinatorial Search (SoCS-2012)*, 97–104.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-Based Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence*, 219: 40–66.

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2013. The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding. *Artificial Intelligence*, 195(Supplement C): 470–495.

Silver, D. 2005. Cooperative Pathfinding. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-2005)*, 117–122.

Silver, D. 2006. Cooperative Pathfinding. In Rabin, S., ed., *AI Game Programming Wisdom 3*, 99–111. Charles River Media.

Standley, T. 2010. Finding Optimal Solutions to Cooperative Pathfinding Problems. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI-2010)*, 173–178.

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 Annual Symposium on Combinatorial Search (SoCS-2019)*, 151–159.

Wagner, G.; and Choset, H. 2015. Subdimensional Expansion for Multirobot Path Planning. *Artificial Intelligence*, 219: 1–24.

Wurman, P. R.; D'Andrea, R.; and Mountz, M. 2008. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Magazine*, 29(1): 9–19.