Learning to Resolve Conflicts for Multi-Agent Path Finding with Conflict-Based Search

Taoan Huang, Sven Koenig, Bistra Dilkina
University of Southern California
{taoanhua, skoenig, dilkina}@usc.edu

Abstract

Conflict-Based Search (CBS) is a state-of-the-art algorithm for multi-agent path finding. On the high level, CBS repeatedly detects conflicts and resolves one of them by splitting the current problem into two subproblems. Previous work chooses the conflict to resolve by categorizing conflicts into three classes and always picking one from the highest-priority class. In this work, we propose an oracle for conflict selection that results in smaller search tree sizes than the one used in previous work. However, the computation of the oracle is slow. Thus, we propose a machine-learning (ML) framework for conflict selection that observes the decisions made by the oracle and learns a conflict-selection strategy represented by a linear ranking function that imitates the oracle's decisions accurately and quickly. Experiments on benchmark maps indicate that our approach, ML-guided CBS, significantly improves the success rates, search tree sizes and runtimes of the current state-of-the-art CBS solver.

Introduction

Multi-Agent Path Finding (MAPF) is the problem of finding a set of conflict-free (that is, collision-free) paths for a given number of agents on a given graph that minimize the sum of costs or the makespan. Although MAPF is NP-hard to solve optimally (Yu and LaValle 2013), significant research effort has been devoted to MAPF to support its application in distribution centers (Ma et al. 2017a; Hönig et al. 2019), traffic management (Dresner and Stone 2008), airplane taxiing (Morris et al. 2015; Balakrishnan and Jung 2007) and computer games (Ma et al. 2017b).

Conflict-Based Search (CBS) (Sharon et al. 2015) is one of the leading algorithms for solving MAPF optimally, and a number of enhancements to CBS have been developed (Boyarski et al. 2015; Li et al. 2019a; Felner et al. 2018; Barer et al. 2014). The key idea behind CBS is to use a bi-level search that resolves conflicts by adding constraints at the high level and replans paths for agents respecting these constraints at the low level. The high level of CBS preforms a best-first search on a binary search tree called *constraint tree* (CT). To expand a CT node (that consists of a set of paths and a set of constraints on these paths), CBS chooses a conflict between two current paths to resolve and adds constraints that prevent this conflict in the child CT nodes. Picking good conflicts is important, and a good strategy for conflict selection could have a big impact on the efficiency of CBS by reducing both the size of CT and its runtime. Boyarski et al. (2015) propose to prioritize conflicts by categorizing them into three classes and always picking one from the highest-priority class. This strategy has been proven to be efficient (Boyarski et al. 2015) and is commonly used for conflict selection in recent research (Li et al. 2019a; Felner et al. 2018; Li et al. 2019c). In this paper, we propose a new conflict-selection oracle that results in smaller CT sizes than the one used in previous work but is much more computationally expensive since it has to compute 1-step lookahead heuristics for each conflict.

To overcome the high computational cost of the oracle, we leverage insights from studies on variable selection for branching in Mixed Integer Linear Programming (MILP) solving and propose to use machine learning (ML) approaches for designing conflict-selection strategies that imitate the oracle's decisions to speed up CBS. Variable selection for branching in MILP solving is analogous to conflict selection in CBS. As part of the branch-and-bound algorithm for MILP solving (Wolsey and Nemhauser 1999), non-leaf nodes in the CT must be expanded into two child nodes by selecting one of the unassigned variables and splitting its domain by adding new constraints, while CBS selects and splits on conflicts. Recent studies (Khalil et al. 2016, 2017; He, Daume III, and Eisner 2014) have shown that data-driven ML approaches for MILP solving are competitive with and can even outperform state-of-the-art commercial solvers.

We borrow such ML approaches from MILP solving (Khalil et al. 2016) and propose a data-driven framework for designing conflict-selection strategies for CBS. In the first phase of our approach, we observe and record decisions made by the oracle on a set of instances and collect data on features that characterize the conflicts at each CT node. In the second phase, we learn a ranking function for conflicts in a supervised fashion that imitates the oracle but is faster to calculate. In the last phase, we use the learned ranking function to replace the oracle and select conflicts in CBS to solve unseen instances. Compared to previous work on conflict selection for CBS, *our ML-guided CBS is able* *to discover more efficient rules for conflict selection that significantly improve the success rate and reduce the both the runtime of the search and the CT size*. Our approach is flexible since we are able to customize the conflict-selection strategy easily for different environments and do not need to hard-code different rules for different scenarios. Different from recent work on ML-guided MILP solving, we utilize problem-specific features which contain essential information about the conflicts, while previous work only takes MILP-level features (e.g., counts and statistics of variables) into account (Khalil et al. 2016, 2017). Another advantage of our offline learning approach over training an instance-specific model on-the-fly is that our learned ranking function is able to generalize to instances and graphs unseen during training.

MAPF

Given an undirected unweighted underlying graph $G = (V, E)$, the *Multi-Agent Path Finding (MAPF) problem* is to find a set of conflict-free paths for a set of agents $\{a_1, \ldots, a_k\}$. Each agent a_i is assigned a start vertex $s_i \in V$ and a goal vertex $t_i \in V$. Time is discretized into time steps, and, at each time step, every agent can either move to an adjacent vertex or wait at the same vertex in the graph. The cost of an agent is the number of time steps until it reaches its goal vertex and no longer moves. We consider two types of conflicts: i) a vertex conflict $\langle a_i, a_j, v, t\rangle$ occurs when agents a_i and a_j are at the same vertex v at time step t; and ii) an edge conflict $\langle a_i, a_j, u, v, t\rangle$ occurs when agents a_i and a_j traverse the same edge $(u, v) \in E$ in opposite directions between time steps t and $t + 1$. Our objective is to find a set of conflict-free paths that move all agents from their start vertices to their goal vertices (called a solution) with the optimal cost, that is, the minimum sum of all agents' costs.

Background and Related Work

In this section, we first provide a brief introduction to CBS and its variants. Then, we summarize other related work on using ML in MAPF and MILP solving.

Conflict-Based Search (CBS)

CBS is a bi-level tree search algorithm. It records the following information for each CT node N:

1. N_{Con}: The set of constraints imposed so far in the search. There are two types of constraints: i) a vertex constraint $\langle a_i, v, t\rangle$, corresponding to a vertex conflict, prohibits agent a_i from being at vertex v at time step t; and ii) an edge constraint $\langle a_i, u, v, t\rangle$, corresponding to an edge conflict, prohibits agent a_i from moving from vertex u to vertex v between time steps t and $t + 1$.
2. N_{Sol}: A solution of N consists of a set of individually cost-minimal paths for all agents respecting the constraints in N_{Con}. An individually cost-minimal path for an agent is a cost-minimal path between its start and goal vertices under the assumption that it is the only agent in the graph.
3. N_{Cost}: The cost of N is the sum of costs of the paths in N_{Sol}.
4. N_{Conf}: The set of conflicts between any two paths in N_{Sol}.

On the high level, CBS starts with a CT with only one node whose set of constraints is empty and then expands the CT in a best-first manner by always expanding a CT node with the lowest N_{Cost}. After choosing a CT node to expand, CBS identifies the set of conflicts N_{Conf} in N_{Sol}. If there are none, CBS terminates and returns N_{Sol}. Otherwise, CBS randomly (by default) selects one of the conflicts to resolve and adds two child nodes of N by imposing, depending on the type of conflict, an edge or vertex constraint on one of the two conflicting agents and adding it to the N_{Con} of one of the child nodes and similarly for the other conflicting agent and N_{Con} of the other child node. On the low level, it replans the paths in N_{Sol} to accommodate the newly-added constraints, if necessary. CBS guarantees completeness by exploring both ways of resolving each conflict and optimality by performing best-first searches on both of its high and low levels.

Variants of CBS

CBS chooses conflicts randomly, but this conflict-selection strategy can be improved. Improved CBS (ICBS) (Boyarski et al. 2015) categorizes conflicts into three types to prioritize them. A conflict is cardinal iff, when CBS uses the conflict to split CT node N, the costs of both resulting child nodes are strictly larger than N_{Cost}. A conflict is semi-cardinal iff the cost of one of the child nodes is strictly larger than N_{Cost} and the cost of the other child node is the same as N_{Cost}. A conflict is non-cardinal otherwise. By first resolving cardinal conflicts, then semi-cardinal conflicts and finally non-cardinal conflicts, CBS is able to improve its efficiency since it increases the lower bound on the optimal cost more quickly by generating child nodes with larger costs. ICBS uses Multi-Valued Decision Diagrams (MDD) to classify conflicts. An MDD for agent a_i is a directed acyclic graph consisting of all cost-minimal paths from s_i to t_i on G that respect the current constraints N_{Con}. The nodes at depth t of the MDD are exactly the vertices that agent a_i could be at at time step t when following one of its cost-minimal paths. A vertex (edge) conflict $\langle a_i, a_j, v, t\rangle$ ($\langle a_i, a_j, u, v, t\rangle$) is cardinal iff vertex v (edge (u, v)) is the only vertex at depth t (the only edge from depth t to depth $t + 1$) in the MDDs of agents a_i and a_j. Li et al. (2019b) propose to add disjoint constraints to both child nodes when expanding a CT node in CBS and prioritize conflicts based on the number of singletons in or the widths of the MDDs of both conflicting agents.

Another line of research focuses on speeding up CBS by calculating a tighter lower bound on the optimal cost to guide the high-level search. When expanding a CT node N, CBSH (Felner et al. 2018) uses the CG heuristic, which builds a conflict graph (CG), whose vertices represent agents and whose edges represent cardinal conflicts in N_{Sol}. Then, the lower bound on the optimal cost within the subtree rooted at N is guaranteed to increase at least by the size of the minimum vertex cover of this CG. We refer to this

	The Random Map				The Game Map			
	Runtime	CT Size	Oracle Time	Search Time	Runtime	CT Size	Oracle Time	Search Time
CBSH2+O_0	9.95s	2,362 nodes	**0.00s**	9.95s	2.3min	952 nodes	**0.0min**	2.3min
CBSH2+O_1	24.89s	746 nodes	21.34s	3.55s	19.8min	**565 nodes**	19.0min	**0.8min**
CBSH2+O_2	12.13s	**632 nodes**	9.52s	**2.61s**	27.4min	2,252 nodes	23.4min	4.0min
ML-S	**6.19s**	998 nodes	0.88s	5.31s	**1.6min**	754 nodes	0.2min	1.4min

Table 1: Performance of CBSH2 with different oracles and our solver. Oracle time is the runtime of the oracle. Search time is the runtime minus the oracle time. All entries are averaged over the instances that are solved by all solvers.

increment as the h-value of the CT node. Based on CBSH, CBSH2 (Li et al. 2019a) uses the DG and WDG heuristics that generalize CG and compute h-values for CT nodes using (weighted) pairwise dependency graphs ((W)DG) that take into account semi-cardinal and non-cardinal conflicts besides cardinal ones. CBSH2 with the WDG heuristic is the current state-of-the-art CBS solver for MAPF (Li et al. 2019a).

To the best of our knowledge, other than prioritizing conflicts using MDDs, conflict prioritization has not yet been explored. Barer et al. (2014) propose a number of heuristics to prioritize CT nodes for the high-level search, including those using the number of conflicts, the number of conflicting agents and the number of conflicting pairs of agents. However, their work uses conflict-related metrics to select CT nodes, while we learn to select conflicts.

Other Related Work

ML techniques are not often applied to MAPF. Sartoretti et al. (2019) propose a reinforcement-learning framework for learning decentralized policies for agents offline to avoid the cost of planning online. Our work is different from their work since we focus on search algorithms and use ML to find efficient and flexible conflict-selection strategies to speed up them. Furthermore, our ML model is simple and easy to implement, without the need to train and fine-tune a deep neural network.

Using ML to speed up search has been explored in the context of MILP solving. Khalil et al. (2016) use ML to learn strategies for branching that mimic strong branching. Our framework is similar to the one in (Khalil et al. 2016) but different in several aspects. Instead of collecting training data and learning a model online, we collect training data and learn a model offline. We leverage insights from existing heuristics for computing h-values to design problem-specific labels and features for learning. Finally, once our model is learned, it performs well on unseen instances while Khalil et al. (2016) learn instance-specific models. This line of work also includes learning when to run primal heuristics to find incumbents in a tree search (Khalil et al. 2017) and learning how to order nodes adaptively for branch-and-bound algorithms (He, Daume III, and Eisner 2014).

Oracles for Conflict Selection

Given a MAPF instance, an oracle for conflict selection at a particular CT node N is a ranking function that takes the set of conflicts N_{Conf} as input, calculates a real-valued score per conflict and outputs the ranks determined by the scores. We say that CBS follows an oracle for conflict selection iff CBS builds the CT by always resolving the conflict with the highest rank. We define oracle O_0 to be the one proposed by (Boyarski et al. 2015), that uses MDDs to rank conflicts.

Definition 4.1. Given a CT node N, oracle O_0 ranks the conflicts in N_{Conf} in the order of cardinal conflicts, semi-cardinal conflicts and non-cardinal conflicts, breaking ties in favor of conflicts at the smallest time step and remaining ties randomly.

Next, we define oracles O_1 and O_2, that both calculate 1-step lookahead scores by using, for each conflict, the two child nodes of N that would result if the conflict were resolved at N.

Definition 4.2. Given a CT node N, oracle O_1 computes the score $v_c = \min\{g_c^l + h_c^l, g_c^r + h_c^r\}$ for each conflict $c \in N_{\mathsf{Conf}}$, where g_c^l and g_c^r would be the costs of the two child nodes of N and h_c^l and h_c^r would be the h-values given by the WDG heuristic of the two child nodes of N if conflict c were resolved at N. Then, it outputs the ranks determined by the descending order of the scores (i.e., the highest rank for the highest score).

Oracle O_1 selects the conflict that results in the tightest lower bound on the optimal cost in the child nodes. We use the WDG heuristic to compute the h-values since it is the state of the art. The intuition behind using this oracle is that the sum of the cost and the h-value of a node is a lower bound on the cost of any solution found in the subtree rooted in the node, and, thus, we want CBS to increase the lower bound as much as possible to find a solution quickly.

Definition 4.3. Given a CT node N, oracle O_2 computes the score $v_c = \min\{m_c^l, m_c^r\}$ for each conflict $c \in N_{\mathsf{Conf}}$, where m_c^l and m_c^r would be the number of conflicts in the two child nodes of N if conflict c were resolved at N. Then, it outputs the ranks determined by the increasing order of the scores (i.e., the highest rank for the lowest score).

Oracle O_2 selects the conflict that results in the least number of conflicts in the child nodes.

We use CBSH2 with the WDG heuristic as our search algorithm and run it with oracles O_0, O_1 and O_2 on (1) the random map, which is a 20×20 four-neighbor grid map with 25% randomly generated blocked cells, and (2) the game map "lak503d" (Sturtevant 2012), which is a 192×192 four-neighbor grid map with 51% blocked cells from the video game *Dragon Age: Origins*. The maps are shown in Table 4. The experiments are conducted on 2.4 GHz Intel Core i7 CPUs with 16 GB RAM. We set the runtime limit to 20 minutes for the random map and 1 hour for the game map. We set the number of agents to $k = 18$ for the random map and $k = 100$ for the game map and run the solvers on 50 instances for each map. Following Stern et al. (2019), the start and goal vertices of each instance are randomly paired among all vertices in each map's largest connected component throughout the paper. In Table 1, we present the performance of the three oracles as well as our solver. All entries are averaged over the instances that are solved by all solvers. We evaluate the oracles according to the resulting CT sizes since they determine the runtime when the calculation of the oracles is not taken into account (and everything else being equal) and first look at the CT sizes of CBSH2 with each of the three oracles. Oracle O_2 is best for the random map, followed closely by oracle O_1. Oracle O_1 is best for the game map. Overall, oracle O_1 is best. Therefore, in the rest of the paper, we mainly focus on learning a ranking function that imitates oracle O_1. Table 1 shows that, by learning to imitate oracle O_1, our solver ML-S achieves the best runtime, even though it induces a larger CT than CBSH2+O_1. We introduce our machine ML methodology and show experimental results in the next two sections.

Machine Learning Methodology

We now introduce our framework for learning which conflict to resolve in CBS. The key idea is that, by observing and recording the features and ranks of conflicts determined by the scores given by the oracle, we learn a ranking function that ranks the conflicts as similarly as possible to the oracle without actually probing the oracle. Our framework consists of three phases:

1. Data collection. We obtain two sets of instances, a training dataset $\mathcal{I}_{\mathsf{Train}}$ and a test dataset $\mathcal{I}_{\mathsf{Test}}$. For each instance $I \in \mathcal{I}_{\mathsf{Train}} \cup \mathcal{I}_{\mathsf{Test}}$, we obtain an instance dataset D_I by running the oracle.
2. Model learning. The training dataset is fed into a machine learning algorithm to learn a ranking function that maximizes the prediction accuracy.
3. ML-guided search. We replace the oracle with the learned ranking function to rank conflicts in the CBSH2 solver. We run the new solver on randomly generated instances on either the same graphs seen during training or unseen graphs.

Data Collection

The first task in our pipeline is to construct a training dataset from which we can learn a model that imitates the oracle's output. We first fix the graph underlying the instances that we want to solve and the number of agents. The number of agents is only fixed during the data collection and model learning phases. We obtain two sets of instances, $\mathcal{I}_{\mathsf{Train}}$ for training and $\mathcal{I}_{\mathsf{Test}}$ for testing. An instance dataset D_I is obtained for each instance $I \in \mathcal{I}_{\mathsf{Train}}$ ($\mathcal{I}_{\mathsf{Test}}$), and the final training (test) dataset is obtained by concatenating these datasets. To obtain dataset D_I, we run CBSH2 on I and oracle O_1 is run for each CT node N to produce the ranking for N_{Conf}. The data consists of: (i) a set of CT nodes $\mathcal{N}$; (ii) a set of conflicts N_{Conf} for all $N \in \mathcal{N}$; (iii) binary labels $y_N \in \{0,1\}^{|N_{\mathsf{Conf}}|}$ for all $N \in \mathcal{N}$ transformed from the oracle's ranking of the conflicts; and (iv) a feature map $\Phi_N : N_{\mathsf{Conf}} \to [0,1]^p$ for all $N \in \mathcal{N}$ that describes conflict $c \in N_{\mathsf{Conf}}$ at CT node N with p features. The test dataset is used to evaluate the prediction accuracy of the learned model.

Features We collect a p-dimensional feature vector $\Phi_N(c)$ that describes a conflict $c \in N_{\mathsf{Conf}}$ in CT node N. The $p = 67$ features of a conflict $\langle a_i, a_j, v, t\rangle$ ($\langle a_i, a_j, u, v, t\rangle$) in our implementation are summarized in Table 2. They consist of (1) the properties of the conflict, (2) statistics of CT node N, the conflicting agents a_i and a_j and the contested vertex or edge w.r.t. N_{Sol}, (3) the frequency of a conflict being resolved for a vertex or an agent, and (4) features of the MDD and the WDG. For each feature, we normalize its value to the range $[0, 1]$ across all conflicts in N_{Conf}. All features of a given conflict $c \in N_{\mathsf{Conf}}$ can be computed in $O(|N_{\mathsf{Conf}}| + k)$ time.

Labels We label each conflict in N_{Conf} such that conflicts with higher ranks determined by the oracle have larger labels. Instead of using the full ranking provided by oracle O_1, we use a binary labeling scheme similar to the one proposed by (Khalil et al. 2016). We assign label 1 to a conflict if no more than 20% of the conflicts in N_{Conf} have the same or a higher score; otherwise, we assign label 0 to it, with one exception. When more than 20% of the conflicts have the same highest O_1 score, we assign label 1 to those conflicts and label 0 to the rest. By doing so, we ensure that at least one conflict is labeled 1 and conflicts with the same score have the same label. This labeling scheme relaxes the definition of "top" conflicts that allow the learning algorithm to focus on only high-ranking conflicts and avoid the irrelevant task of learning the correct ranking of conflicts with low scores.

Model Learning

We learn a linear ranking function with parameters $\boldsymbol{w} \in \mathbb{R}^p$

$$f : \mathbb{R}^p \to \mathbb{R} : f(\Phi_N(c)) = \boldsymbol{w}^{\mathsf{T}} \Phi_N(c)$$

that minimizes the loss function

$$L(\boldsymbol{w}) = \sum_{N \in \mathcal{N}} l(y_N, \hat{y}_N) + \frac{C}{2}||\boldsymbol{w}||_2^2,$$

where y_N is the ground-truth label vector, $\hat{y}_N$ is the vector of predicted scores resulting from applying f to the feature vectors of every conflict in N_{Conf}, $l(\cdot,\cdot)$ is a loss function measuring the difference between the ground truth labels and the predicted scores, and $C > 0$ is a regularization parameter. The loss function $l(\cdot,\cdot)$ is based on a pairwise loss that has been used in the literature (Joachims 2002). Specifically, we consider the set of pairs $\mathcal{P}_N = \{(c_i, c_j) : c_i, c_j \in N_{\mathsf{Conf}} \wedge y_N(c_i) > y_N(c_j))\}$, where $y_N(c)$ is the ground-truth label of conflict c in label vector y_N. The loss function $l(\cdot,\cdot)$ is the fraction of swapped pairs, defined as

$$l(y_N, \hat{y}_N) = \frac{1}{|\mathcal{P}_N|}|\{(c_i, c_j) \in \mathcal{P}_N : \hat{y}_N(c_i) \le \hat{y}_N(c_j)\}|.$$

We use an open-source software package (Joachims 2006) that implements a Support Vector Machine (SVM) approach (Joachims 2002) that minimizes an upper bound on the loss, which is NP-hard to minimize.

Feature Descriptions	Count
Types of the conflict: binary indicators for edge conflicts, vertex conflicts, cardinal conflicts, semi-cardinal conflicts and non-cardinal conflicts.	5
Number of conflicts involving agent a_i (a_j) that have been selected and resolved so far during the search: their min., max. and sum.	3
Number of conflicts that have been selected and resolved so far during the search at vertex u (v): their min., max. and sum.	3
Number of conflicts that agent a_i (a_j) is involved in: their min., max. and sum.	3
Time step t of the conflict.	1
Ratio of t and the makespan of N_{Sol}.	1
Cost of the path of agent a_i (a_j): their min., max., sum, absolute difference and ratio of their max. and min.	5
Difference of the costs of the path of agent a_i (a_j) and its individually cost-minimal path: their min. and max.	2
Ratio of the costs of the path of agent a_i (a_j) and its individually cost-minimal path: their min. and max.	2
Difference of the cost of the path of agent a_i (a_j) and t: their min. and max.	2
Ratio of the cost of the path of agent a_i (a_j) and t: their min. and max.	2
Ratio of the cost of the path of agent a_i (a_j) and N_{Cost}: their min. and max.	2
Binary indicator whether none (at least one) of agents a_i and a_j has reached its goal vertex by time step t.	2
Number of conflicts $c' \in N_{\mathsf{Conf}}$ such that $\min\{d_{q,q'} : q \in V_c^T, q' \in V_{c'}^T\} = w$ $(0 \le w \le 5)$.	6
Number of agents a such that there exists $q' \in V_a$ and $q \in V_c^T$ such that $d_{q,q'} = w$ $(0 \le w \le 5)$.	6
Number of conflicts $c' \in N_{\mathsf{Conf}}$ such that $\min\{d_{q,q'} : q \in V_c, q' \in V_{c'}\} = w$ $(0 \le w \le 5)$.	6
Width of level w $(\lvert w - t \rvert \le 2)$ of the MDD for agent $a_i(a_j)$: their min. and max. (Li et al. 2019b).	10
Weight of the edge between agents a_i and a_j in the weighted dependency graph (Li et al. 2019a).	1
Number of vertices q' in graph G such that $\min\{d_{q',q} : q \in V_c\} = w$ $(1 \le w \le 5)$.	5

Table 2: Features of a conflict $c = \langle a_i, a_j, u, t\rangle$ ($\langle a_i, a_j, u, v, t\rangle$) of a CT node N. Given the underlying graph $G = (V, E)$, let $V_T = \{(v,t) : v \in V, t \in \mathbb{Z}_{\ge 0}\}$, $E_T = \{((u,t),(v,t+1)) : t \in \mathbb{Z}_{\ge 0} \wedge (u = v \vee (u,v) \in E)\}$ and define the time-expanded graph as an unweighted graph $G_T = (V_T, E_T)$. Let $d_{u,v}$ be the cost of the cost-minimal path between vertices u and v in G and $d_{(u',t'),(u,t)}$ be the distance from (u',t') to (u,t) in G_T if $t' \le t$ or from (u,t) to (u',t'), otherwise. For a conflict $c' = \langle a'_i, a'_j, u', t'\rangle$ ($\langle a'_i, a'_j, u', v', t'\rangle$) in N_{Conf}, define $V_{c'} = \{u'\}$ ($V_{c'} = \{u', v'\}$) and $V_{c'}^T = \{(u',t')\}$ ($V_{c'}^T = \{(u',t'),(v',t')\}$). For an agent a, define $V_a = \{(u,t)$: agent a is at vertex u at time step t following its path$\}$. The counts are the numbers of features contributed by the corresponding entries, which add up to $p = 67$.

		Warehouse	Room	Maze	Random	City	Game
Number of agents in instances in $\mathcal{I}_{\mathsf{Train}}$ and $\mathcal{I}_{\mathsf{Test}}$		30	22	30	18	180	100
Training on the same map	Swapped pairs (%)	5.78	12.58	4.5	10.89	2.89	4.40
	Top pick accuracy (%)	84.93	67.56	87.69	69.03	83.05	60.16
Training on the other maps	Swapped pairs (%)	6.08	15.24	21.98	19.64	7.66	7.45
	Top pick accuracy (%)	86.85	66.80	49.90	50.44	78.57	53.13

Table 3: Numbers of agents in instances in $\mathcal{I}_{\mathsf{Train}}$ and $\mathcal{I}_{\mathsf{Test}}$, test losses and accuracies. The swapped pairs are the percentages of swapped pairs averaged over all test CT nodes, and the top pick accuracy is the accuracy of the ranking function selecting one of the conflicts labled as 1 in the test dataset.

ML-Guided Search

After offline data collection and ranking function $f(\cdot)$ learning, we replace oracle O_1 for conflict selection in CBS with the learned function. At each CT node N, we first compute the feature vector $\Phi_N(c)$ for each conflict $c \in N_{\mathsf{Conf}}$ and pick the conflict with the maximum score $c^* = \arg\max_{c \in N_{\mathsf{Conf}}} f(\Phi_N(c))$. The time complexity of conflict selection at node N is $O(|N_{\mathsf{Conf}}|(|N_{\mathsf{Conf}}|+k))$. Even though the complexity of conflict selection with oracle O_0 is only $O(|N_{\mathsf{Conf}}|)$, we will show in our experiments that we are able to outperform CBSH2+O_0 in terms of both the CT size and the runtime.

Experimental Results

In this section, we demonstrate the efficiency and effectiveness of our solver, ML-guided CBS, through extensive experiments. We use the C++ code for CBSH2 with the WDG heuristic made available by Li et al. (2019a) as our CBS version. We compare against CBSH2+O_0 as baseline since O_0 is the most commonly used conflict-selection oracle. The reason why we choose CBSH2 with the WDG heuristic over CBS, ICBS and CBSH2 with the CG or DG heuristics is that it performs best, as demonstrated in (Li et al. 2019a). All reported results are averaged over 100 randomly generated instances.

Our experiments provide answers to the following ques-

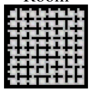
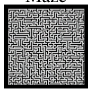
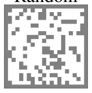
Map	k	Success Rate (%)			Runtime (min)			CT Size (nodes)			PAR10 Score (min)		
		CBSH2	ML-S	ML-O	CBSH2	ML-S	ML-O	CBSH2	ML-S	ML-O	CBSH2	ML-S	ML-O
Warehouse	30	93	**96 (93)**	**96 (93)**	0.20	**0.06**	0.07	1,154	**294**	378	7.18	**4.14**	4.25
	36	72	86 (71)	**88** (71)	0.54	0.24	**0.19**	3110	980	**977**	28.46	14.56	**12.81**
	42	55	68 **(55)**	**70 (55)**	1.27	0.65	**0.38**	6,834	2,874	**1,781**	45.70	32.61	**30.56**
	48	17	**32 (17)**	**32 (17)**	1.99	1.12	**0.56**	9,646	5,357	**2,221**	83.34	68.64	**68.48**
	54	6	**16 (6)**	15 **(6)**	2.82	1.70	**1.23**	12,816	8,886	**6,427**	94.17	**84.42**	85.36
	Improvement over CBSH2				0	49.8%	**64.4%**	0	56.6%	**68.2%**	0	**22.9%**	22.7%
Room 	22	83	**91 (83)**	**91 (83)**	0.61	**0.49**	0.51	7,851	**5,648**	5,888	17.51	**9.76**	9.83
	26	47	**57 (47)**	55 (46)	1.32	**1.01**	1.14	15,791	**11,087**	12,108	53.68	**43.97**	45.91
	30	28	**36 (28)**	34 **(28)**	2.08	**1.21**	1.45	21,279	**10,284**	12,117	73.22	**65.32**	67.28
	32	17	**24 (17)**	**24 (17)**	1.88	**1.39**	1.70	22,152	**13,943**	16,327	83.77	**77.02**	77.14
	34	9	**14 (9)**	**14 (9)**	3.99	**2.70**	3.24	39,447	**22,611**	28,392	91.36	**86.56**	86.63
	Improvement over CBSH2				0	**26.6%**	21.3%	0	**35.2%**	32.0%	0	**14.0%**	11.6%
Maze 	30	90	**91 (90)**	90 **(90)**	0.54	0.47	**0.42**	500	373	**289**	10.49	**9.51**	10.38
	32	84	**87 (84)**	**87 (84)**	0.49	**0.39**	0.42	519	427	**397**	16.42	**13.59**	13.60
	36	80	81 **(80)**	**82** (79)	0.73	0.65	**0.57**	1,200	1,067	**910**	20.66	19.68	**18.68**
	40	56	60 **(56)**	**62 (56)**	0.85	0.80	**0.75**	1,194	1,099	**1,026**	44.47	40.79	**38.85**
	44	45	49 **(45)**	**50 (45)**	1.08	1.06	**0.87**	1,389	1,343	**1,055**	54.49	50.82	**49.75**
	Improvement over CBSH2				0	10.3%	**18.3%**	0	13.0%	**24.4%**	0	6.3%	**8.2%**
Random 	18	95	**95 (95)**	94 (94)	0.32	**0.23**	0.31	5032	**3,105**	4,148	5.32	**5.27**	6.29
	20	88	**91 (88)**	**91 (88)**	0.43	**0.30**	0.36	7,834	**3,829**	4,595	12.38	**9.37**	9.48
	23	74	**80 (74)**	**80 (74)**	0.96	**0.56**	0.78	17,952	**8,118**	11,555	26.71	**20.60**	20.81
	26	39	**48 (39)**	45 **(39)**	1.27	**0.87**	1.24	19,236	**8,053**	13,301	61.50	**52.75**	55.82
	29	17	**27 (17)**	24 **(17)**	4.04	**2.74**	3.39	63,661	**35,485**	44,179	83.69	**74.07**	77.02
	Improvement over CBSH2				0	**33.4%**	17.6%	0	**49.3%**	35.8%	0	**15.1%**	10.3%
City	180	78	**85** (76)	84 (75)	3.53	**2.43**	2.46	859	**468**	476	134.99	**93.04**	99.87
	200	76	82 (75)	**83** (75)	4.78	5.08	**4.13**	849	702	**490**	147.96	113.53	**106.78**
	230	57	**68** (56)	64 (54)	4.86	4.26	**4.24**	835	**444**	449	261.36	**196.99**	220.50
	260	44	**54** (44)	**54** (43)	11.69	9.75	**9.55**	1,883	**1,178**	1,219	341.37	**282.00**	282.58
	290	18	27 (16)	**28** (17)	11.65	**8.45**	8.75	1,966	**1,372**	1,429	494.10	441.87	**436.70**
	Improvement over CBSH2				0	24.0%	**25.2%**	0	**47.3%**	46.4%	0	19.3%	**20.2%**
Game	100	68	**77 (68)**	75 **(68)**	6.76	**5.49**	5.94	4,100	**3,114**	3,341	196.66	**145.09**	156.74
	110	59	**67 (59)**	**67 (59)**	6.58	6.03	**5.92**	3,978	3,652	**3,596**	249.89	202.61	**202.39**
	120	35	**44 (35)**	**44** (34)	9.59	**8.76**	8.80	5,351	**4,643**	4,691	393.27	**341.63**	341.82
	125	34	41 **(34)**	**42 (34)**	9.32	7.77	**7.58**	5,145	4,153	**4,054**	399.18	358.91	**353.32**
	130	19	**26 (19)**	25 (18)	**4.83**	5.00	4.85	2,486	2,498	**2,338**	487.01	**447.22**	453.05
	Improvement over CBSH2				0	16.6%	**17.3%**	0	22.7%	**23.3%**	0	16.1%	**16.4%**

Table 4: Success rates, average runtimes and CT sizes of instances solved by all solvers and PAR10 scores for different numbers of agents k on 6 maps. For the success rates of ML-S and ML-O, the percentages of instances solved by both our solvers and CBSH2 are given in parentheses (bolded if they solve all instances that CBSH2 solves). For each map, we report the percentages of our improvement over CBSH2 on the runtime and CT size on instances solved by all solvers and the PAR10 score.

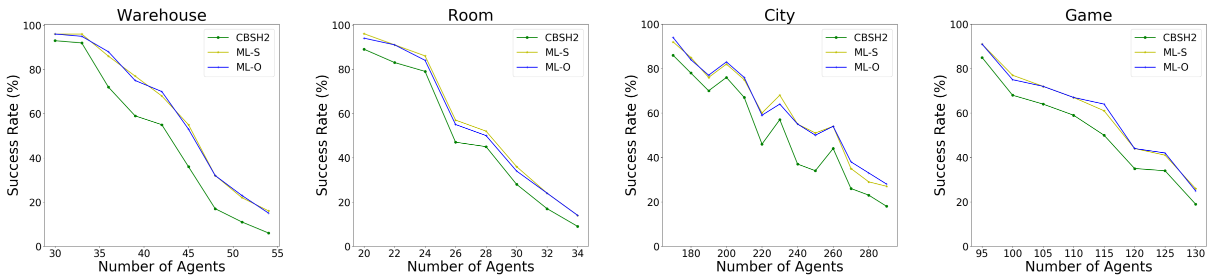


Figure 1: Success rates within the runtime limit.

tions: i) If the graph underlying the instances is known in advance, can we learn a model that performs well on unseen instances on the same graph with different numbers of agents? ii) If the graph underlying the instances is unknown in advance, can we learn a model from other graphs that performs well on instances on that graph?

We use a set of six four-neighbor grid maps $\mathcal{M}$ of different sizes and structures as the graphs underlying the instances and evaluate our algorithms on them. $\mathcal{M}$ includes (1) a warehouse map (Li et al. 2020), which is a 79×31 grid map with 100 6×2 rectangular obstacles; (2) the room map "room-32-32-4" (Stern et al. 2019), which is a 32×32 grid map with 64 3×3 rooms connected by single-cell doors; (3) the maze map "maze-128-128-2" (Stern et al. 2019), a 128×128 grid map with two-cell-wide corridors; (4) the random map; (5) the city map "Paris_1_256" (Stern et al. 2019), which is a 256×256 grid map of Paris; (6) the game map. The figures of the maps are shown in Table 4. For each map $M \in \mathcal{M}$, we collect data from randomly generated training instances $\mathcal{I}^{(M)}_{\mathsf{Train}}$ and test instances $\mathcal{I}^{(M)}_{\mathsf{Test}}$ on M with a fixed number of agents, where $|\mathcal{I}^{(M)}_{\mathsf{Train}}| = 30$ and $|\mathcal{I}^{(M)}_{\mathsf{Test}}| = 20$. We learn two ranking functions for map M: one ranking function that is trained using 5,000 CT nodes i.i.d. sampled from the training dataset collected by solving instances $\mathcal{I}^{(M)}_{\mathsf{Train}}$ on the same map and another one that is trained using 5,000 CT nodes sampled from the training dataset collected by solving instances $\cup_{M' \in \mathcal{M}} \mathcal{I}^{(M')}_{\mathsf{Train}} \setminus \mathcal{I}^{(M)}_{\mathsf{Train}}$ on the other maps, namely 1,000 i.i.d. CT nodes sampled from each of the five other maps. For each map $M \in \mathcal{M}$, we denote our solver that uses the ranking function trained on the same map by ML-S and the solver that uses the one trained on the other maps by ML-O. We set the regularization parameter $C = 1/100$ to train an SVM^{rank} (Joachims 2002) with a linear kernel to obtain each of the ranking functions. We varied $C \in \{1/10, 1/100, 1/1000\}$ and achieved similar results. We test the learned ranking functions on the test dataset collected by solving $\mathcal{I}^{(M)}_{\mathsf{Test}}$. The numbers of agents in the instances used for data collection, the test losses and the test accuracies of selecting one of the conflicts labeled as 1 are reported in Table 3. We varied the numbers of agents for data collection and found that they led to similar performance. In general, the losses of the ranking functions for ML-O are larger and their accuracies of selecting "good" conflicts are lower than those for ML-S.

We run CBSH2, ML-S and ML-O on randomly generated instances on each of the six maps and vary the number of agents. The runtime limits are set to 60 minutes for the two largest maps (the city and game maps) and 10 minutes for the other maps. In Table 4, we report the success rates, the average runtimes and the average CT sizes of instances solved by all solvers and the PAR10 scores (a commonly used metric to score solvers where we count the runs that exceed the given runtime limit as 10 times the limit when computing the average runtimes) (Bischl et al. 2016) for some numbers of agents on each map. We plot the success rates on the warehouse, room, city and game maps in Figure 1. ML-S and ML-O dominate CBSH2 in all metrics on all maps for almost all cases. Overall, CBSH2, ML-S and ML-O solve 3,326 (55.43%), 3,779 (62.98%) and 3,758 (62.63%) instances out of 6,000 we tested, respectively. The improvement of ML-S and ML-O over CBSH2 on instances commonly solved by all solvers is 10.3% to 64.4% for the runtime and 13.0% to 68.2% for the CT sizes across different maps. For ML-S, even though we learn the ranking function from data collected on instances with a fixed number of agents (listed in Table 3), the learned function generalizes to instances with larger numbers of agents on the same map and outperforms CBSH2. ML-O, without seeing the actual map being tested on during training, is competitive with ML-S and even outperforms ML-S sometimes on the warehouse, city, maze and game maps. The results suggest that our approach, when focusing on solving instances on a particular map, can outperform CBSH2 significantly and, when faced with new maps, still has an advantage.

Next, we look at the feature importance of the learned ranking functions. For ML-O, the six ranking functions have nine features in common among their eleven features with the largest absolute weights. Thus, they are similar when looking at the important features. We take the average of each weight and sort them in decreasing order of their absolute values. The top eight features are (1) the weight of the edge between agents a_i and a_j in the weighted dependency graph (WDG); (2) the binary indicator for non-cardinal conflicts; (3) the maximum of the differences of the cost of the path of agent a_i (a_j) and t; (4) the binary indicator for cardinal conflicts; (5) the minimum of the numbers of conflicts that agent a_i (a_j) is involved in; and (6-8) the minimum, the maximum and the sum of the numbers of conflicts involving agent a_i (a_j) that have been selected and resolved. Those features mainly belong to three categories: features related to the conflict type, the WDG and the number of conflicts having been resolved for agents, where the first one is commonly used in previous work on CBS and the third one is an analogue of the branching variable pseudocosts (Achterberg, Koch, and Martin 2005) in MILP solving. For more experimental results, we refer readers to the full version of the paper[1].

Conclusions and Future Directions

In this paper, we proposed the first ML framework for conflict selection in CBS. Our extensive experimental results showed that our learned ranking function can generalize across different numbers of agents on both a fixed graph and unseen graphs. Our objective was to imitate the decisions made by the oracle that picks the conflict that produces the tightest lower bound on the optimal cost in its child nodes. We are also interested in discovering a better oracle for conflict selection from which we can learn. We expect our method to work well with other newly-developed techniques, such as symmetry breaking techniques (Li et al. 2020), and it remains future work to incorporate these techniques into the framework of CBSH2 to work with our ML-guided conflict selection.

[1]The full version of the paper can be found at https://arxiv.org/abs/2012.06005.

Acknowledgments

We thank Jiaoyang Li, Peter J. Stuckey and Danial Harabor for helpful discussions. The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189, 1837779 and 1935712, the U.S. Department of Homeland Security under Grant Award No. 2015-ST-061-CIRC01 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 necessarily representing the official policies, either expressed or implied, of the U.S Department of Homeland Security.

References

Achterberg, T.; Koch, T.; and Martin, A. 2005. Branching rules revisited. *Operations Research Letters* 33(1): 42–54.

Balakrishnan, H.; and Jung, Y. 2007. A framework for coordinated surface operations planning at Dallas-Fort Worth International Airport. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, 6553.

Barer, M.; Sharon, G.; Stern, R.; and Felner, A. 2014. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Annual Symposium on Combinatorial Search*, 19–27.

Bischl, B.; Kerschke, P.; Kotthoff, L.; Lindauer, M.; Malitsky, Y.; Fréchette, A.; Hoos, H.; Hutter, F.; Leyton-Brown, K.; Tierney, K.; et al. 2016. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence* 237: 41–58.

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Tolpin, D.; Betzalel, O.; and Shimony, E. 2015. ICBS: Improved conflict-based search algorithm for multi-agent pathfinding. In *International Joint Conference on Artificial Intelligence*, 442–449.

Dresner, K.; and Stone, P. 2008. A multiagent approach to autonomous intersection management. *Journal of Artificial Intelligence Research* 31: 591–656.

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

He, H.; Daume III, H.; and Eisner, J. M. 2014. Learning to search in branch and bound algorithms. In *Advances in Neural Information Processing Systems*, 3293–3301.

Hönig, W.; Kiesel, S.; Tinka, A.; Durham, J. W.; and Ayanian, N. 2019. Persistent and robust execution of MAPF schedules in warehouses. *IEEE Robotics and Automation Letters* 4(2): 1125–1131.

Joachims, T. 2002. Optimizing search engines using clickthrough data. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 133–142.

Joachims, T. 2006. Training linear SVMs in linear time. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 217–226.

Khalil, E. B.; Dilkina, B.; Nemhauser, G. L.; Ahmed, S.; and Shao, Y. 2017. Learning to run heuristics in tree search. In *International Joint Conference on Artificial Intelligence*, 659–666.

Khalil, E. B.; Le Bodic, P.; Song, L.; Nemhauser, G. L.; and Dilkina, B. 2016. Learning to branch in mixed integer programming. In *AAAI Conference on Artificial Intelligence*, 724–731.

Li, J.; Felner, A.; Boyarski, E.; Ma, H.; and Koenig, S. 2019a. Improved heuristics for multi-agent path finding with conflict-based search. In *International Joint Conference on Artificial Intelligence*, 442–449.

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*, 193–201.

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 *International Conference on Automated Planning and Scheduling*, 279–283.

Li, J.; Surynek, P.; Felner, A.; Ma, H.; Kumar, T. S.; and Koenig, S. 2019c. Multi-agent path finding for large agents. In *AAAI Conference on Artificial Intelligence*, 7627–7634.

Ma, H.; Li, J.; Kumar, T. S.; and Koenig, S. 2017a. Lifelong multi-agent path finding for online pickup and delivery tasks. In *International Conference on Autonomous Agents and Multi-Agent Systems*, 837–845.

Ma, H.; Yang, J.; Cohen, L.; Kumar, T. S.; and Koenig, S. 2017b. Feasibility study: Moving non-homogeneous teams in congested video game environments. In *Artificial Intelligence and Interactive Digital Entertainment Conference*, 270–272.

Morris, R.; Chang, M. L.; Archer, R.; Cross, E. V.; Thompson, S.; Franke, J.; Garrett, R.; Malik, W.; McGuire, K.; and Hemann, G. 2015. Self-driving aircraft towing vehicles: A preliminary report. In *AI for Transportation Workshop at the AAAI Conference on Artificial Intelligence*.

Sartoretti, G.; Kerr, J.; Shi, Y.; Wagner, G.; Kumar, T. S.; Koenig, S.; and Choset, H. 2019. PRIMAL: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters* 4(3): 2378–2385.

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

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; Boyarski, E.; and Bartak, R. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks 151–158.

Sturtevant, N. R. 2012. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games* 4(2): 144–148.

Wolsey, L. A.; and Nemhauser, G. L. 1999. *Integer and combinatorial optimization*, volume 55. John Wiley & Sons.

Yu, J.; and LaValle, S. M. 2013. Planning optimal paths for multiple robots on graphs. In *IEEE International Conference on Robotics and Automation*, 3612–3617.