Conflict-Oriented Windowed Hierarchical Cooperative A*

Zahy Bnaya and Ariel Felner
Information Systems Engineering Department, Ben Gurion University, Israel
{zahy,felner}@bgu.ac.il

***Abstract*— In the Multi-Agent Path Finding problem (MAPF), we are given a map and a set of agents with distinct source and goal locations. The task is to compute a path for each agent from its initial location to its goal location without conflicting with other agents.**

MAPF solvers can be divided into classes based on their purpose. One of these classes is the class of online MAPF algorithms, in which the search for paths is interleaved with the actual physical moves of the agents. A prominent algorithm in this class is the *Windowed Hierarchical Cooperative A* algorithm* (WHCA*) where paths are planned for each agent individually and cooperation is obtained using a reservation table.

A number of extensions for WHCA* already exist. In this paper we propose a general approach for the baseline WHCA* algorithm which is orthogonal to all other existing extensions. We improve WHCA* by introducing the *Conflict Oriented* (CO) principle for focusing the agent coordination around conflicts. In addition, we provide a conflict-oriented prioritization mechanism that intelligently chooses which agent should act next. Experimental results demonstrate the advantage of our approach over WHCA*.

I. Introduction

Recently, there has been a growing interest in the *multi-agent path finding* (MAPF) problem. MAPF consists of a graph and a number of agents. For each agent, a unique start and goal states are given, and the task is to find paths for all agents from their start states to their goal states, under the constraint that agents must not collide. Usually, the aim is to minimize a cumulative cost function such as the sum of the time steps for all agents to reach their goals. MAPF has many practical applications in robotics, video games, and other domains [1], [2], [3], [4].

In this work we assume a *centralized* setting with a single CPU that controls cooperative agents. The case where agents are non-cooperative was discussed in [5].

A. Classes of MAPF

MAPF solvers can be divided into classes. *Offline solvers* first perform all computations in a single continuous interval and return a *plan* - a set of paths for the agents to follow. The agents begin to follow the plan only after the computation is over and no additional computation is performed. Therefore, the plan cannot contain parts where agents collide. Some offline solvers aim to find the optimal solutions according to some predefined cost function. In the past, variants of the A* algorithm were used on the state-space induced by placing agents into locations [6]. Recently more advanced algorithms were developed either for the A* framework [7], or by using unique search trees and novel search algorithms [8], [9]. As optimal solvers they do not scale up. Other offline algorithms sacrifice optimality or completeness in order to reduce computation. For example, the HCA* algorithm [1] calculates paths for each agent individually. Once a path for an agent is found, that path is written (*reserved*) into a global spatio-temporal *reservation table* where subsequent agents cannot conflict with the reserved paths. The concept of using such a reservation table to coordinate agents was also used in [2] to enable multiple vehicles to move on a shared crossroad.

In this paper we focus on *online* solvers for MAPF that interleave computation and physical moves. They sacrifice optimality and may not be complete, but as any online algorithm, they respond quickly (sometimes in realtime) and agents do not spend a lot of time calculating paths. Some online algorithms are *constructive* or *rule-based* algorithms. These algorithms provide special rules or macros for the agents which prevent them from colliding and finally getting them to their goal locations. Usually, the solution quality is very far from optimal and many times no bounds on optimality are given. A prominent example of such algorithms is the *Push and Swap* algorithm [10]. Other rule-based algorithms were suggested ([11], [12]).

A straightforward approach for an online search solver for MAPF is to use the concept of *Local-Repair A** (LRA*) [13]. Here, paths are planned for each agent while ignoring other agents. Then, agents start moving. When a conflict is about to occur it is locally resolved. For example, by constructing detours (repairs) for some of the agents.

A prominent online search MAPF solver is the *Windowed Hierarchical Cooperative A** (WHCA*) algorithm [1] which is an online variant of HCA*. WHCA* only reserves paths of length W (window size parameter). The agents then follow the partially-reserved paths and a new cycle begins from the agents' current locations. In each cycle a different ordering of agents is used to allow a balanced distribution of the reservation table between the agents. A number of variants and enhancements of WHCA* were introduced. They all combine reservation tables with online moves. Other works use map abstractions in order to compute less conflicted paths [14], [15], [16]. However, these algorithms still need some form of coordination. Usually, this coordination is achieved using the WHCA* window principle. For example, Jansen et al. [15] uses a shared data structure called *direction maps* to achieve better coordination between the agents. Although this method encourages cooperative behavior, agents can still collide. Another map abstraction method is used in the *Flow Annotation Replanning* (FAR) algorithm [16]. The

ideas presented in this paper are orthogonal to all of these variants. Therefore, for simplicity, we describe and compare our approach to the baseline WHCA* variant as was first presented by [1].

B. Contribution of the paper

WHCA* does not consider the locations and times where conflicts are likely to occur and uses a fixed size reservation window. This can lead to inefficient behavior on some cases. In this paper we aim to remedy these drawbacks. We make the following two contributions.

First we introduce the notion of *Conflict Oriented reservation* and the resulting *Conflict Oriented-WHCA** algorithm (CO-WHCA*). CO-WHCA* focuses the reservation windows on areas and time-steps where conflicts between agents occur. Additionally, with this new approach, reservations are made most of the time only by a single agent. This reduces the number of reservations and avoids many expensive integrations in the search process. CO-WHCA* uses a parameter to determine the start of the reservation window and the point up to which agents should move, given a conflict which is about to occur. In the extreme case, where this parameter is set to 0, CO-WHCA* boils down to an offline algorithm that can be called CO-HCA*. CO-HCA* improves HCA* on the same issues. Although not discussed in this paper, a non-parametric approach is also possible where the length and position of the reservation window is determined by a heuristic function. We leave this direction for future research.

Our second contribution is a mechanism called *Conflict Oriented prioritization* where we prioritize agents on the reservation table based on the likelihood of the agents to make efficient detours.

We provide experimental results that show that CO-WHCA* achieves considerably better execution times than WHCA* while keeping the solution quality and the success rates. We also demonstrate the advantage of using our conflict-oriented prioritization to achieve better solution qualities on expanse of execution time.

II. Definitions and terminology

The input to MAPF is a graph, $G(V, E)$ and a set of agents labeled $a_1 \dots a_k$. Each agent a_i has a start position $s_i \in V$ and goal position $g_i \in V$. Time is discretized into time steps. At each time step an agent can either *move* to a neighboring location or *wait* in its current location. Both actions cost 1. The task is to find a collision free path for each agent while minimizing a global cost function. We allow agents to move simultaneously but restrict moving on the same edge in opposite directions. The cost function we use is the *sum of time steps* [17], [8], [9] which is the sum of time steps required to reach the goal location for all agents. We use the term *path* in the context of a single agent. We define a *space-time point*, or in short st-point as a pair (v, t) where v is a vertex of the graph and t is a time point. A path $p_i = \langle (s_i = v_0, 0), (v_1, 1) \dots (g_i = v_g, g) \rangle$ for agent a_i is a sequence of st-points that brings the agent from the start to the goal where agent a_i is located at location v_t at time step t. We use the term *plan* to denote a set of paths for the given set of agents. Formally, a *conflict* is defined as a tuple (A, v, t) where A is a set of agents that use the (v, t) st-point. A *valid plan* is a plan that its paths have no conflicts.

III. Online MAPF Algorithms

Unlike offline algorithms that find an entire *valid-plan* before agents start to navigate, online algorithms work in cycles which interleave a *planning phase* and a *moving phase*. In the planning phase, partial paths are computed. In the moving phase agents physically move along these paths. Online algorithms may be required in realtime environments when moving must be done on a regular, time limited intervals.

The local repair A* (LRA*) [13] is a general online search approach, originally designed for a single agent that has only partial knowledge on the environment (i.e., some edges or vertices might turn out to be missing or blocked when the agent physically arrives at these locations). To find a path, LRA* uses an optimistic view of the environment where unknown parts of the environment are assumed traversable (commonly known as the *Free Space Assumption* [18]). The agent starts to follow a path which might contain untraversable or missing edges. In this case the moving phase halts and a new planning phase starts from the current location, with the new knowledge.

Silver [1] reports that a basic variant of LRA* was the main common approach for many years in industrial applications for online MAPF solvers. In the planning phase, all agents compute an "optimistic" path to their goals, ignoring the existence of other agents. Agents then start moving. When two or more agents are about to move to the same location, thus causing a conflict, one of the agents is allowed to use the conflicted location while the others are forced to replan, thus, locally resolving the conflict. The basic implementation of LRA* for MAPF suffers from infinite-loops and cycles on narrow or bottleneck environments [1]. To address this, Sliver suggested the HCA* algorithm (offline) and the WHCA* algorithm (online) which are covered next.

Algorithm 1: WHCA* pseudo-code

```
1 Input: MAPF,W,K
2 reset T
3 while some agents are not at their goal do
4     for each agent a_i do
5         findPath(a_i, T)
6         Reserve first W steps
7     for each agent a_i in parallel do
8         Move a_i K time steps
9     reset T
```

A. HCA* and WHCA*

HCA* is a suboptimal and incomplete offline algorithm. HCA* calculates individual paths for each agent according to some predefined order. After each path calculation, the algorithm reserves the entire series of st-points of that path in the reservation table. When searching for a path for any given

agent, st-points already reserved by previous agents cannot be used, and this assures that no conflicts will occur. Agent prioritization in HCA* is determined arbitrarily. However, this order is known to be very sensitive. It is very easy to think of an example with two agents where a given order will find a solution but reversing the order will prevent the algorithm from finding a solution.

The prioritization issue was partially mitigated by the *Windowed Hierarchical Cooperative A** algorithm (WHCA*) [1]. WHCA* is an online variant of HCA* where *plan - move* cycles are activated. WHCA* uses two parameters, W for the planning phase and K for the moving phase. In the planning phase, agents are ordered arbitrarily and each agent in turn searches for a path to its goal. However, unlike HCA*, in WHCA* the agents only make reservations for the next W time steps. Therefore, the paths that are generated are only guaranteed to be conflict-free for the next W steps. In the moving phase, each agent follows its reserved path up to time step K (another parameter, the length of the moving phase). Since the paths calculated are only conflict-free up to W time steps, the equation $K \leq W$ must always hold. After these K steps are taken, the reservation table is erased (even for time steps t where $k < t \leq W$) and a new cycle begins at time step k from the current location of the agents. The pseudo code of WHCA* is given in algorithm 1. First, a reservation table $\mathcal{T}$ is initialized to empty (line 1). Then, paths up to length W are computed and stored in the reservation table for each of the agents. In the moving phase, agents move K steps each in parallel.

WHCA* has several important advantages over HCA*. First, Silver suggested that in each cycle, a different prioritization order will be used. In particular, Silver suggested to iterate through the agents in a round-robin fashion. Second, agents only reserve W steps. Compared to HCA*, this saves memory of the reservation table. Additionally, the planning time is reduced as less st-points are reserved. In practice, W and K should be determined carefully. A window which is too large, demands more time and space in the reservation table. Using large W or small K is more likely to result in futile planning since the plan might not be relevant on later stages. On the other hand, a window which is too small can lead to many computation phases and cycles due to short sighted decisions. Likewise, the value of the K parameter also impacts the algorithm behavior. Silver only used one value for K: $K = W/2$.

Silver observed that WHCA* outperforms HCA* in the distance traveled by the agents. In addition, neither HCA* nor WHCA* guarantee completeness. Silver showed that the success rate (i.e., the proportion of instances where a solution was found and the agents arrived at their goals) of WHCA* was larger than that of HCA*. This advantage is attributed to the balanced prioritization and the fact that none of the agents had better privileges on the reservation table. Moreover, both HCA* and WHCA* outperform LRA* in their success rates, solution quality and the number of *search - move* cycles.

WHCA* does not consider conflicts that occur between agents and thus has a number of drawbacks. First, for each agent, WHCA* (and HCA*) reserves W st-points for the next W time steps, regardless of whether any of the st-points is required by another agent. One might consider an example where the individual optimal paths of all agents have no conflicts. In this case, reserving any of the st-points is useless. Furthermore, assume that the first conflict between two agents occurs at time step $W+1$. These two agents will calculate their full paths (that conflicts at time point $W+1$) but reserve only the first W st-points. These two agents then move along these W st-points unaware of the conflict. Only when a new cycle begins, the conflict at time step $W+1$ will be detected. Agents might find it hard to overcome a conflict so close to their physical position.

Algorithm 2: CO-WHCA* : MAPF

```
1  reset T
2  while some agents are not at their goal do
3      for each agent a_i do
4          planPath(a_i, T)
5      c = ⟨A, v, t⟩ ← findFirstConflict
6      a_c ← conflictOwner(c)
7      w_start ← movePosition(c)
8      w_end ← reservePosition(c)
9      reserve w_start ... w_end st-points of a_c in T
10     for each agent a_i in parallel do
11         Move a_i to w_start
```

IV. The Conflict-Oriented WHCA*

Our new algorithm, Conflict Oriented WHCA* (CO-WHCA*) aims to address these drawbacks by combining ideas from both WHCA* and local-repair A*. Similar to WHCA*, CO-WHCA* uses a reservation table with a limited length. CO-WHCA* is flexible about the window position and places it only around conflicts that were found in the planning phase. As LRA*, CO-WHCA* is flexible in its switching between the moving phase and the planning phase.

Algorithm 2 describes CO-WHCA*. Similar to WHCA*, initially (line 1), a reservation table is set to empty. Then, *plan-move* cycles are repeated until a solution is found.

In CO-WHCA*, the location of the window where paths are reserved is flexible. Unlike WHCA*, where the reservations are erased between cycles, in CO-WHCA*, reservations that were made in planning phase of cycle i are kept for the planning phase of cycle $i+1$ if the moving phase of cycle i stopped before the relevant time steps. Such reserved st-points are denoted as *leftovers*. The *planning phase* of CO-WHCA* consists of 3 steps: **(1)** A path is found for each agent individually (line 4) from its current location to its goal. This path does not conflict with the st-points that are leftovers from previous cycles in reservation table. Naturally, in the first cycle, there are no leftovers. **(2)** These paths are simulated until the earliest conflict $c = \{A, v, t\}$ (i.e., a set of agents A are using the (v, t) st-point) is found (line 5). This conflict is, of course, beyond the time steps of any leftovers from previous cycles. **(3)** Finally, a reservation mechanism similar to the one used by WHCA* is used. However, in CO-WHCA* we use additional information (lines 6-8) in

order to position the reservation window around the conflict c (line 9). This is described next.

Given a conflict c, one of the agents is chosen as the *conflict-owner* of c, denoted as a_c (line 6). Only a_c is allowed to use the conflicting st-points in its path. We set two time points, w_{start} and w_{end} (lines 7-8). These two time points constitute the reservation window around conflict c. We then reserve the st-points in the path that was calculated by a_c between w_{start} and w_{end} (Line 9).

In our implementation, the methods $movePosition(c)$ and $reservePosition(c)$ that decide on w_{start} and w_{end} use a predetermined fixed length W for the reservation window. We position the reservation window such that the conflict time is located in the center of the reservation window. Thus, for a conflict $c = \langle A, v, t\rangle$ and a reservation window size W, $movePosition(c)$ set w_{start} to be $t - w/2$ and $reservePosition(c)$ sets w_{end} to be $t + w/2$. Although we do not cover any of them in this work, we can think of several extensions to the above implementations. For example, a non-parametric approach that decides on w_{start} and w_{end} using heuristics.

In the moving phase of CO-WHCA* (Line 10), all agents move up to the last step before to w_{start}. That is, agents move exactly to the last time step that was not reserved in the preceding planning phase. Then, a new *plan-move* cycle begins when the agents are located at their new locations. In the next *plan* phase all agents search for paths from their current location to their goal, considering the modified reservation table. That is, the reservation around the conflict is passed as a leftover for the next planning phase.

Agents are now located rather close, but a little before the conflict ($W/2$ steps away). Thus, those who cannot step into the conflict will have enough time to find a meaningful detour. This is the main advantage of CO-WHCA* over WHCA*, as in WHCA*, agents might start to plan right before the conflict.

On cases where more than one conflict occurs at time point t, we repeat lines (6–9) for each of the conflicts (omitted from Algorithm 2 for clarity). Thus, the algorithm proceeds to the *move* phase only when all conflicts up to time t had been assigned reservation windows.

A. Running example

To demonstrate CO-WHCA*, consider Figure 1. Initially, agents A and B (agents' locations are marked with white circles), each need to cross the entire grid to get to their goals while agent C is one step away from its goal and the reservation table is empty. The first *plan-move* cycle begins and each of the agents find a path to their goal (Figure 1(a)). By simulating the paths, the first conflict is found between the paths of agents A and B at time point t (marked with a red X). In this example, agent B is chosen arbitrarily as the *conflict master*. We assume a fixed window size of size W and that the reservation window should be positioned evenly around the conflict. Thus, we select w_{start} and w_{end} such that W st-points around the conflict are reserved by agent B (Figure 1(b)). Next, the move phase begins (Figure 1(c)) and the agents proceed to their locations at w_{start}. A new plan phase begins at step (d). Since agent B reserved the interval adjacent to conflict c, agent A plans a detour - a path that conflicts with agent C. Now, C gets to reserve the interval (Step (e)) and the next move phase advances agent A near the location of c (Figure 1(f)). Next, a new plan is found and all agents. Agent C again reserves the interval around the conflict and agent A plans an alternative path to its destination. This time, none of the agents have conflicts.

The space requirements of the reservation table used in WHCA* and HCA* depends on the number of agents ($W \cdot |A|$ in WHCA*) since all agents reserve st-points at each phase. In CO-WHCA* , the reservation table is usually a vector of size W since most of the time, only one agent uses the reservation table for a limited duration. One exception to this is when multiple conflicts occur at the same time.

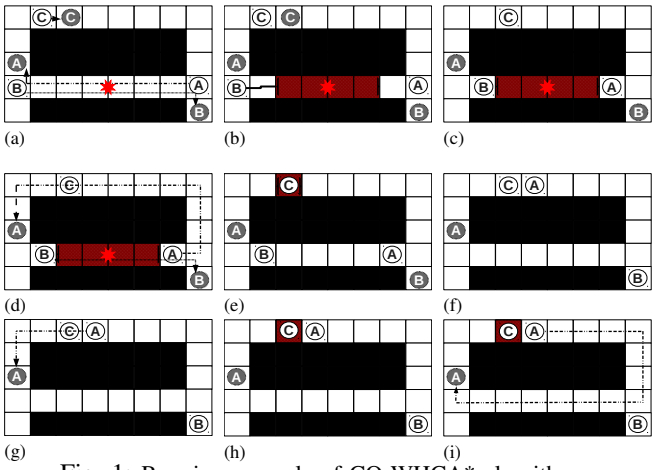

Fig. 1: Running example of CO-WHCA* algorithm.

B. Offline version

CO-WHCA* can also be easily modified to be an offline MAPF solver. To do so, we set $w_{start} = 0$. That is, all the *moving phases* are trivial and agents do not move until all the *planning phases* complete. Therefore, at each phase a conflict is identified and one of the conflicting agents is set to be the conflict-master. This is repeated until no more conflicts are found and each agent has a full path to its goal. Then, a single *moving-phase* is activated and agents move to their goals. This variant of CO-WHCA* is denoted by CO-WHCA*(0) or simply CO-HCA*. When using CO-HCA* in the example of Figure 1, after the first conflict is found between A and B, the agents start their second round of planning from their initial locations. Then the conflict between A and C is found and resolved. Only then, the agents start to move.

Algorithm 3: $select_{wd}$: A,v,t

```
1 for each agent a_i ∈ A do
2     reset T_tmp
3     reserve st-points for agent a_i
4     for each agent a_j ∈ A do
5         findPath(a_j, T_tmp)
6         Estimation[a_i]+ = cost of path for agent a_j
7 return arg min_{a∈A} Estimation[a]
```

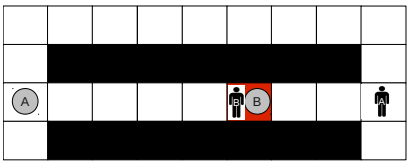

Fig. 2: WHCA* prioritization sensitivity

V. Conflict-oriented prioritization

The order by which the agents get to use the reservation table is known to be important for all algorithms that use a reservation table. HCA* uses a random prioritization of agents that may lead to inefficiency. Consider the example in Figure 2. Agent A needs to cross the map to its goal while agent B needs to stay at its starting position. If each agent moves according to its best path, a conflict occurs at time step 3 in the goal location of agent B. Assume a fixed window size 8. By giving agent A the highest priority, it reserves the path in the corridor up to time step 8 forcing agent B to circle around the wall back to its home location. However, it is much more efficient to give agent B the highest priority. In this case, B reserves its current location for the first time steps, forcing agent A to take the bypass around the wall. Thus, achieving a better total cost.

Several suggestions have been previously proposed for agent prioritization [19], [20], [21]. Van der berg et al. [20] considers continuous environments for multi-robot navigation and suggests to prioritize by decreasing order of the shortest path length for each agent. Note that in our example of Figure 2 it is best to give the agent with the shorter path a higher priority. This is a counter example to the heuristics used in [20]. In this example, HCA*, WHCA* and CO-WHCA* will all fail to achieve the optimal solution unless agent B gets the highest priority. Therefore, although the windowed reservation of WHCA* makes the prioritization less critical, it still plays a major role in the success of the algorithm, especially as the number of agents grows.

A. The winner determination prioritization scheme

We propose a general prioritization scheme for CO-WHCA* . We call this scheme the *winner-determination* of a conflict. The number of possible agent-prioritizations for a MAPF instance is $k!$ where k is the number of agents. However, when agent-prioritization is limited to a single conflict $c = \{v, t, A\}$, there are $|A|!$ possible prioritizations, where $|A|$ is usually a small number. Moreover, since in CO-WHCA* only a single agent is set to be the conflict master, we only need to choose one agent out of A.

Our scheme estimates all possible orders and selects the best one as described in Algorithm 3. We iterate over all agents $a_i \in A$ and make each of them in turn the conflict master. We reserve a_i's relevant st-points in a temporary reservation table $\mathcal{T}_{tmp}$ (line 3). Then, each of the other agents in A finds a path that does not conflict with the temporary reservation table (line 5). We ignore all future conflicts and sum the costs for the entire group of agents A. This summation is used as heuristics and we choose the agent that resulted in the minimal sum of costs. The variant of CO-WHCA* that uses our *winner-determination* conflict master selection is denoted as CO-WHCA*P (P for prioritized) and replaces the arbitrary selection of agent that appeared at line 6, with the procedure $select_{wd}(A)$ (Algorithm 3).

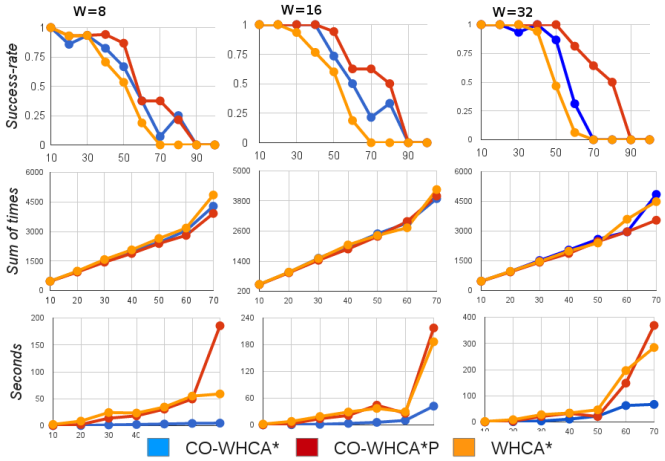

Fig. 3: Evaluation of our methods on different number of agents.

#agents	HCA		HCA$_{pr}$		CO-HCA*		CO-HCA*P	
	cost	time	cost	time	cost	time	cost	time
10	467	2.6	475	1.3	470	**0.13**	**464**	0.33
20	959	12.7	1,004	7.6	985	**2.7**	**946**	7.4
30	1,489	34.6	1,556	29.4	1,480	**10.5**	**1,416**	27.5
40	1,933	54.5	2,054	33.8	1,923	**23.4**	**1,891**	61.8
50	2,514	92.1	2,670	69.9	2,421	**64.9**	**2,222**	93.2
60	3,100	97.8	3,246	128.8	2,948	**87.1**	**2,919**	134.2

TABLE I: Costs and execution times (seconds) for offline versions

VI. Empirical results

We compared our new algorithms to WHCA* and HCA*. We used 50 instances of 32x32 four-connected grids and 20 maps from the game **Dragon Age Origin** [22]. The number of agents varies from 10 to 70. 20% of the cells in each instance are randomly selected as un-traversable. Our selection method of start and goal states was biased towards generating instances where many conflicts occur. For each traversable cell l we calculate a *density coefficient* which is the number of cells located in proximity r from l that are used as sources or destinations for other agents or that are un-traversable. When randomizing source or destination we give higher probability to cells with high *density coefficient*.

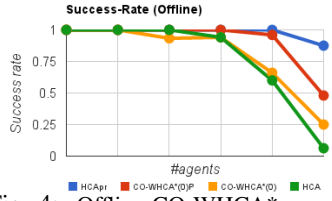

Fig. 4: Offline CO-WHCA* success rates

A. Offline versions

In Table I we compare our offline methods to the basic HCA* algorithm and to HCA*$_{pr}$. HCA*$_{pr}$ prioritizes agents according to the length of their shortest-path [20]. Figure 4 demonstrates the percentage of instances that each of the algorithms was able to solve (success rate). For HCA*, unsolvable instances are such that the algorithm fails to find a path for some of the agents. For CO-WHCA* and CO-WHCA*P we limit the number of cycles to 100, after which

we consider the instance unsolvable. Both CO-HCA* and CO-HCA*P achieve better success rates than basic HCA* with randomized prioritization. $HCA*_{pr}$, however, is able to solve more instances than our algorithms, especially with a large number of agents. But on the other hand, its solution costs are worse. Our algorithms also have advantage in execution times. Especially when there are small number of agents. CO-HCA*P execution time increases when the number of agents is large and therefore many calculations of agent prioritization are needed. However, usually CO-HCA*P achieves better solution costs than CO-HCA*. Thus there is a time-quality tradeoff. As the number of agents and conflicts increases, CO-HCA*P might be impractical.

B. Online versions

We used several window sizes for WHCA* (8,16,32). We set w_{start} and w_{end} at a distance of $W/2$ from the conflict time. Figure 3 show the success rates, solution costs and running times. Both CO-WHCA* and CO-WHCA*P outperform WHCA* in their success rate. CO-WHCA* achieves a dramatic improvement over WHCA* in execution time without sacrificing the solution qualities. CO-WHCA*P usually achieves better solution qualities than CO-WHCA* but requires longer calculation of many incremental searches.

Both of our algorithms achieve better results than WHCA*. Nevertheless, these two algorithms have an internal time-quality tradeoff. We suggest to use our CO-WHCA*P algorithm if time permits. Otherwise, we suggest to use CO-WHCA*. In our implementation we recalculated the alternative paths from scratch. However, a more efficient calculation could be made by using suitable algorithms such as D^*Lite [23]. Note that in our experiments, most algorithms scaled to at most 70 agents. Thus, we report our findings only up to 70 agents.

	CO-WHCA*	CO-WHCA*P	WHCA*
Success rate	1	1	0.85
Solution cost	767	744	785
Execution time	31	2,854	1,082

TABLE II: Results for Dragon Age Origin

Table II shows the results from the game *Dragon Age Origin* (map den520d) over 20 instances and 10 agents. CO-WHCA* outperformed WHCA* in all three aspects making it the algorithm of choice for this domain. However, CO-WHCA*P required a considerable amount of time and achieved only a slight improvement in solution quality.

VII. Conclusion and Future Work

In this work we investigated two techniques that improve over the WHCA* algorithm, both as an online and as an offline algorithm. The first technique focuses the reservations on areas around a potential conflict and allows only a single agent to use the reservation table. The second technique determines efficient prioritization of agents, given a detected conflict. Experimental results shown in this paper demonstrate that both these techniques are helpful. We showed that both our algorithms can achieve better results than WHCA* on a random 32x32 domain and a considerable advantage on the *Dragon Age Origin* maps. Also, although CO-WHCA*P achieves better solution quality, it should be carefully used since it is time demanding.

Future work will proceed in the following directions. Integrate the conflict-oriented principle with existing online algorithms such as the FAR algorithm [16]. Propose other winner-determination methods. For example, based on abstractions and shared data-structures such as *direction maps* [15]. Develop a non-parametric approach that automatically determines the start and end of the reservation window. These could be determined by heuristics.

VIII. Acknowledgements

This research was supported by the Israeli Science Foundation (ISF) under grant number 417/13 to Ariel Felner.

References

[1] D. Silver, "Cooperative pathfinding," in *AIIDE*, 2005, pp. 117–122.
[2] K. Dresner and P. Stone, "A multiagent approach to autonomous intersection management," *JAIR*, vol. 31, pp. 591–656, March 2008.
[3] K.-H. C. Wang and A. Botea, "Scalable multi-agent pathfinding on grid maps with tractability and completeness guarantees." in *ECAI*, 2010, pp. 977–978.
[4] A. Bleiweiss, "Gpu accelerated pathfinding," in *23rd ACM SIGGRAPH/EUROGRAPHICS symposium*, 2008, pp. 65–74.
[5] Z. Bnaya, R. Stern, A. Felner, R. Zivan, and S. Okamoto, "Multi-agent path finding for self interested agents," in *SOCS*, 2013.
[6] M. Ryan, "Constraint-based multi-robot path planning," in *ICRA*, 2010, pp. 922–928.
[7] T. Standley and R. Korf, "Complete algorithms for cooperative pathfinding problems," in *IJCAI*, 2011, pp. 668–673.
[8] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, "The increasing cost tree search for optimal multi-agent pathfinding," in *IJCAI*, 2011, pp. 662–667.
[9] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent path finding," in *AAAI*, 2012.
[10] R. Luna and K. E. Bekris, "Push and swap: Fast cooperative pathfinding with completeness guarantees," in *JAIR*, 2011, pp. 294–300.
[11] M. M. Khorshid, R. C. Holte, and N. R. Sturtevant, "A polynomial-time algorithm for non-optimal multi-agent pathfinding," in *SOCS*, 2011.
[12] K.-H. C. Wang and A. Botea, "MAPP: a Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees," *JAIR*, vol. 42, pp. 55–90, 2011.
[13] A. Zelinsky, "A mobile robot exploration algorithm," *Robotics and Automation, IEEE Transactions on*, vol. 8, no. 6, pp. 707–717, 1992.
[14] N. R. Sturtevant, "Memory-efficient abstractions for pathfinding," in *AIIDE*, 2007, pp. 31–36.
[15] M. Jansen and N. Sturtevant, "Direction maps for cooperative pathfinding," in *AIIDE*, 2008.
[16] K. C. Wang and A. Botea, "Fast and memory-efficient multi-agent pathfinding," in *ICAPS*, 2008, pp. 380–387.
[17] T. Standley, "Finding optimal solutions to cooperative pathfinding problems," in *AAAI*, 2010, pp. 173–178.
[18] S. Koenig and Y. Smirnov, "Sensor-based planning with the freespace assumption," in *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, vol. 4. IEEE, 1997, pp. 3540–3545.
[19] J. Barraquand and J.-C. Latombe, "Robot motion planning: A distributed representation approach," *The International Journal of Robotics Research*, vol. 10, no. 6, pp. 628–649, 1991.
[20] J. P. van den Berg and M. H. Overmars, "Prioritized motion planning for multiple robots," in *IROS 2005*, 2005, pp. 430–435.
[21] M. Bennewitz, W. Burgard, and S. Thrun, "Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots," *Robotics and Autonomous Systems*, vol. 41, no. 2, pp. 89–99, 2002.
[22] N. Sturtevant, "Benchmarks for grid-based pathfinding," *Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144 – 148, 2012.
[23] S. Koenig and M. Likhachev, "D^* lite," in *Proceedings of the national conference on artificial intelligence*, 2002, pp. 476–483.