Adversarial Cooperative Path-finding: Complexity and Algorithms

Marika Ivanová and **Pavel Surynek**
Charles University Prague, Malostranské náměstí 25, Praha, 118 00, Czech Republic
MarikaIvanova@seznam.cz, pavel.surynek@mff.cuni.cz

Abstract— **The paper addresses a problem of adversarial cooperative path-finding (ACPF) which extends the well-studied problem of cooperative path-finding (CPF) with adversaries. In addition to cooperative path-finding where non-colliding paths for multiple agents connecting their initial positions and destinations are searched, consideration of agents controlled by the adversary is included in ACPF. This work is focused on both theoretical properties and practical solving techniques of the considered problem. We study computational complexity of the problem where we show that it is PSPACE-hard and belongs to the EXPTIME complexity class. Possible methods suitable for practical solving of the problem are introduced and thoroughly evaluated. Suggested solving approaches include greedy algorithms, minimax methods, Monte Carlo Tree Search, and adaptation of an algorithm for the cooperative version of the problem. Solving methods for ACPF were compared in a tournament in which all the pairs of suggested strategies were compared. Surprisingly frequent success rate of greedy methods and rather weaker results of Monte Carlo Tree Search were indicated by the conducted experimental evaluation.**

Keywords: cooperative path-finding, adversaries, Monte Carlo Tree Search, complexity, PSPACE-hardness

I. Introduction and Motivation

The problem of *adversarial cooperative path-finding* (ACPF) can be regarded as a generalization of the well-studied *cooperative path-finding* (CPF) [1], [4], [5], [6], [8] which is extended by adversarial element. The standard cooperative path-finding is a path planning problem in a fully observable static environment where the task is to find non-colliding routes for agents that lead them from their initial locations to given separate destinations. All the agents are controlled centrally while agents themselves make no decisions. The adversarial element consists in adding agents that are outside of the control of the central planning mechanism and that plays against it.

The environment where agents are moving is modeled as an undirected graph in CPF [1], [6]. The same abstraction is adopted in ACPF as well. Hence, CPF and ACPF are both considered as discrete combinatorial problems. The notion of adversarial agents has been first introduced in a short paper [3] where however solving techniques were not addressed due to space constraints. In this paper, the concept of ACPF is further elaborated in terms of solving techniques and a deeper theoretical analysis is given.

This research is supported by the Czech Science Foundation under the contract number GAP103/10/1287.

The classical CPF is motivated by problems arising in both real environments and as well as in virtual worlds (multi-robot navigation, container relocation, path-finding in computer games [8]). Note that, agents do not need to be represented by autonomous movable units. They can be passive objects or even pieces of certain commodity – the only important property is localization and space (virtual) occupation by agents.

The classical CPF has however limited expressive power for real world situations, as not all the environments are fully cooperative. That is, we cannot regard all the agents as controllable and the environment as static any more. Dealing with such adversarial or hostile elements in the environment beyond the standard CPF is thus desirable.

Our suggestion is to introduce two or more teams of agents that compete in finding paths to target destinations to model the adversarial element in CPF. The objective in the ACPF problem is to control agents of one selected team so that its agents reach their destinations before agents of adversarial teams. The adversarial agents observe the analogical objective – they also want to reach their destinations as first. The task is thus to find a winning strategy which means a decision mechanism that is able to react on all possible actions of adversaries.

There is a variety of possibilities how teams of agents can harm each other in the effort to reach their destinations as first – occupying target destination, blocking of narrow passage, or preventing agents from moving.

In the classical CPF, the combinatorial difficulty arises from the need of avoidance between agents [4]. The situation in ACPF is even more complex as the planning mechanism must consider all possible acting of the adversarial teams. Hence, combinatorial difficulty in ACPF comes not only from the need of avoidance but also from the need to consider possible harmful actions of adversaries.

There are many situations in the real world that can be abstracted as ACPF. Police actions or tactical military maneuvers such as blockade, encirclement, flanking, or their preventing can be planned through means of ACPF. These days it turns out to be extremely important to move troops to strategic locations as first without contact with the enemy. Another obvious application of the ACPF concept is game industry, particularly real-time strategic games.

The paper is organized as follows. We first concentrate on the formal definition of ACPF. Subsequently we will study

theoretical properties of the problem, particularly its computational complexity. It is shown that ACPF is PSPACE-hard. The first sketch of the proof of PSPACE-hardness has been published in [3], but the proof is very brief and does not contain technical details. Here, an alternative proof based on different techniques is shown with all the necessary details.

Investigation of possible solving methods and their experimental evaluation follows in the next part. As the ACPF problem is related to n-player games and to the classical CPF, we will focus on methods developed for these problems and consider their adaptation and application for ACPF. Conducted experiments should reveal how suitable are the suggested methods for solving various types of ACPF instances. We will also compare all the pairs of suggested solving methods against each other in a kind of strategy tournament in the final part.

II. Formal Definition of Adversarial Cooperative Path Finding

An abstraction of the studied problem is necessary for further processing. We adopted usual terminology known from CPF. The adversarial element present in ACPF requires extension and adaptation of existing definitions. The environment is modeled as an undirected graph, where vertices represent locations that can be occupied by agents. There is at most one agent located in each vertex, which models the spatial constraints. Edges represent passable regions. That is, an agent can relocate from one vertex to a neighboring vertex provided no collision occurs from this relocation.

Definition 1 (Adversarial Cooperative Path-Finding). An instance of *adversarial cooperative path-finding* problem (ACPF) is a 7-tuple $\Sigma = (G, A, \mathcal{T}, t^*, \lambda_0, \lambda_+, \hat{\alpha})$ where $G = (V, E)$ is an undirected graph, $A = \{a_1, a_2, \dots, a_k\}$ is a set of agents, $\mathcal{T} = \{T_1, T_2, \dots, T_t\}$ is a set of teams with $t \leq k$. Teams are disjoint sets of agents and every agent belongs to exactly one team (that is, $A = \bigcup_{i=1}^{t} T_i$). t^* denote an index of the selected team for that play; other teams are our adversaries. $\lambda_0 \colon A \to V$ is an injective mapping that assigns an initial vertex to each agent (a starting position). $\lambda_+ \colon A \to \mathcal{P}(V)$ assigns a target set of vertices to each agent. Finally, $\hat{\alpha}$ represent as an adversarial control mechanism that determines the next placement of all agents belonging to adversarial teams. It takes a sequence of all the previous placements of all the agents as its input to be able to decide according to the past. □

The continuous time is divided into discrete time steps in APCF. The placement of agents at the i-th time step is an injective mapping $\lambda_i \colon A \to V$. The moves are taken as instantaneous transformations of agents' placement. Agent movements must observe several rules that correspond to real physical limitations. That is, λ_i can be transformed to λ_{i+1} if and only if following conditions hold:

(i) Agents move along edges or stay at a vertex.

(ii) An agent a can move to an unoccupied vertex v or if v was occupied at time step i by agent a' different from a, agent a' must move away. The second condition allows agents to move around a cycle without any free vertex. A trivial case of position swapping along an edge is an exception and is forbidden.

(iii) Teams alternate in their moves. That is, only agents of team $T_{i+1 \,(mod\ t)}$ moves between times steps i and $i+1$.

$$\Sigma = (G, A, \mathcal{T}, t^*, \lambda_0, \lambda_+, \hat{\alpha})$$

$A = \{1,2,3,4,5,6\}$
$\mathcal{T} = \{T_1, T_2\}$
$T_1 = \{1,2,3\}$
$T_2 = \{4,5,6\}$
$t^* = 1$ (green)

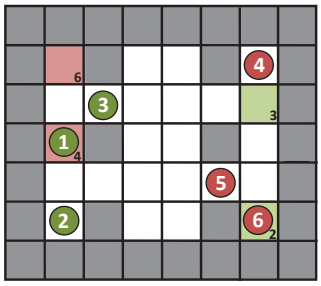

Figure 1. *An example of **ACPF instance**.* Agents of two teams are depicted as green and red circles. The graph is represented by a 4-connected grid with obstacles. Initial positions λ_0 correspond to current locations of agents and goals λ_+ are depicted as squares of color corresponding to the given team. Note, that agent 1 of team T_1 (selected green team) and agent 5 of team T_2 (adversarial red team) do not have any particular targets.

The task in ACPF is to find a winning strategy for agents of the selected team T_{t^*}. The winning strategy tells us how to move agents of the team T_{t^*} in reaction to movements of adversarial agents so that agents of T_{t^*} reach their destinations as first.

Definition 2 (Solution of Acpf Instance). A solution to an ACPF instance $\Sigma = (G = (V, E), A, \mathcal{T}, t^*, \lambda_0, \lambda_+, \hat{\alpha})$ is a *winning strategy* constituted by a mapping $\hat{\tau}$ that determines moves of team T_{t^*} at time steps when it is on turn so that T_{t^*} reaches its destinations before other teams whatever they do. That is, for any sequence of mappings of agents to vertices $[\lambda_0, \lambda_1, \dots, \lambda_{t*}, \dots, \lambda_{t*+t}, \dots, \lambda_{t*+2t}, \dots, \lambda_{t*+lt}]$ with $l \in \mathbb{N}$ that originate in λ_0 where λ_{t*+it} is determined by $\hat{\tau}$ from $[\lambda_0, \lambda_1, \dots, \lambda_{t*}, \dots, \lambda_{t*+t}, \dots, \lambda_{t*+2t}, \dots, \lambda_{t*+it-1}]$ for every $i = 0,1, \dots, l$ and all the consecutive mappings satisfy conditions (i)-(iii) from definition 1, one of the following two conditions hold:

(1) $\lambda_{t*+lt}(a) \in \lambda_+(a)$ for each $a \in T_{t^*}$ and there exists $a \in A$ such that $\lambda_j(a) \notin \lambda_+(a)$ for $j < t^* + lt$
(T_{t^*} wins at $(t^* + lt)$-th step and all the preceding placements of agents are not winning for any team)

(2) there exists $a \in A$ such that $\lambda_j(a) \notin \lambda_+(a)$ for any $j \in \{0,1, \dots, t^* + lt\}$

(all the placements of agents in the sequence are not winning for any team). □

The problem definition does not guarantee the existence of a solution since the graph is not required to be connected or the agents can block each other so they can never reach their target locations. Hence, in practice we focus on weaker objectives such as relocating a certain number of agents to their targets and/or restrict the time limit. A winning team either manages to relocate its agents into their targets before an adversary within the time limit, or achieves a position, that is closest to the desired target position among all the teams when the time is up. In order to be able to compare the teams' positions, we introduce following definition.

Definition 3 (WINNING TEAM). For a mapping $\lambda: A \to V$ and a team $T \in \mathcal{T}$ we define $\gamma_\lambda(T)$ as the number of agents from team T placed at their target vertices. Further we $\delta_\lambda(T)$ as a sum of lengths of the shortest paths between current positions of agents from T and their targets. We can consider either the shortest path including only unoccupied vertices, or the shortest path in graph without agents. Former option is always of at least the same length or longer than the latter one. Now we can define an ordering of a pair of teams $T_1, T_2 \in \mathcal{T}$ with respect to given mapping λ:

$$T_1 <_\lambda T_2 \equiv \gamma_\lambda(T_2) < \gamma_\lambda(T_1) \vee \left(\gamma_\lambda(T_1) = \gamma_\lambda(T_2) \wedge \delta_\lambda(T_1) < \delta_\lambda(T_2)\right)$$

This ordering allows us to determine a winning team, even if the run was terminated before any target position were reached.

III. COMPUTATIONAL COMPLEXITY OF ACPF

We are interested in the decision variant of the problem with respect to computational complexity: given an instance of ACPF problem, we want to decide whether there exists a winning strategy for a selected team. We will show that decision variant of ACPF is PSPACE-hard and belongs to the EXPTIME complexity class [1]. Whether ACPF belongs to PSPACE, that is, if it is PSPACE-complete is currently an open question. The problematic point is that we do not have any polynomial upper bound on the length of the sequence of moves in the winning strategy.

We will start with the proof of PSPACE-hardness. It will be accomplished by a reduction of TQBF – a variant of QBF [7]– to ACPF.

The following auxiliary lemmas will help us to construct needful reduction.

Lemma 1 (VERTEX BOOKING). Let $\Sigma = (G = (V, E), A, \mathcal{T}, t^*, \lambda_0, \lambda_+, \hat{\alpha})$ be an ACPF instance and $v \in V$ be so called *booked vertex*. Next, let $b \in \mathbb{N}$ be so called *booking time* and T_{t_b} with $t_b \in \{1, \dots, t\}$ be some team, for that the vertex should be booked. Then there exists a modified instance $\Sigma' = (G' = (V', E'), A', \mathcal{T}, t^*, \lambda'_0, \lambda'_+, \hat{\alpha}')$ such that the booked vertex v is not accessible by any other agent before time-step b. ■

Proof. Σ' is derived from Σ by adding an extra vertex v' that is connected by an edge to the vertex v. Alongside we put a new agent $a' \in T_{t_b}$ at the booked vertex v. Agent a' will enter the vertex v' at booking time b. Hence any teammate located at a vertex adjacent to v can enter v at time step b. Figure 2 depicts the situation. ■

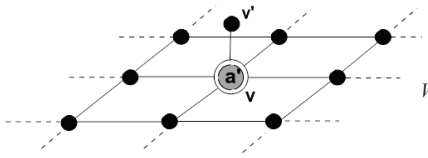

Figure 2. *Vertex booking.*

Lemma 2 (HAZARDOUS VERTEX). Let $\Sigma = (G = (V, E), A, \mathcal{T}, t^*, \lambda_0, \lambda_+, \hat{\alpha})$ be an ACPF instance and T_{t_h} with $t_b \in \{1, \dots, t\}$ be some team. Next let $v \in V$ be so called *hazardous vertex* and $s \in \mathbb{N}$. Then there exists a modified instance $\Sigma' = (G' = (V', E'), A', \mathcal{T}, t^*, \lambda'_0, \lambda'_+, \hat{\alpha}')$ such that the hazardous vertex v can be occupied by an agent $a' \in A \setminus T_{t_h}$ at time step s, if v is not entered by any other agent $a \in T_{t_h}$ at some former time step $s' < s$.

Proof. Σ' is derived from Σ by attaching a path $v'_0, v'_1, \dots, v'_{s-1}, v$. Vertex v'_0 contains an agent $a' \in A \setminus T_{t_h}$. This agent can approach the vertex v every time step and enter it in the s-th step, unless any other agent do it earlier. Situation is depicted in the Figure 3. ■

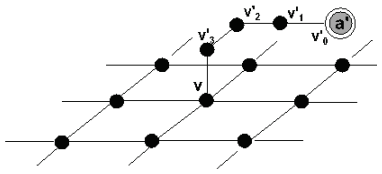

Figure 3. *Hazardous vertex.*

Preposition 1. *Decision problem whether there exists a winning strategy for selected team in a given ACPF instance is PSPACE-hard.* ■

Proof plan. We will provide a polynomial-time reduction of TQBF [7] to ACPF. Without loss of generality we can assume, that given TQBF φ is in 3CNF and each variable is present both as a positive and negative literal. If these properties do not hold, we can transform φ into an equivalent φ' fulfilling these requirements. For given formula φ we construct an ACPF instance, where the movement of some agents will simulate gradual valuation of variables appearing in φ. Consequently, we will show that φ is valid if and only if there exists a winning strategy for the selected team. The

construction is inspired on the proof of NP-completeness of the optimization variant of CPF [10].

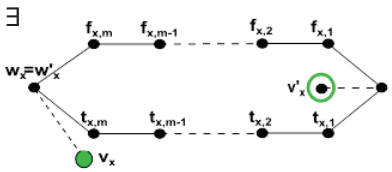

Figure 4. ***Existentially*** *quantified variable gadget.*

Construction. Let $\varphi: Q_1 x_1 \cdots Q_n x_n \psi$ be the given formula, n be the number of its variables, and m be the number of clauses in φ. For every **existentially** quantified variable x we will construct a gadget consisting of two parallel paths of the length m, joining at a vertex w_x (see Figure 4).

There is also a path connected to w_x, ended by vertex v_x, where sits an agent a_x from T_{t^*}. The length of the path from w_x to v_x corresponds to the order of x in the quantifier part of φ. Target vertex of the agent a_x is placed at the vertex v'_x.

For **universally** quantified variable y, there is a similar gadgets (see Figure 5). Agent a_y placed at the vertex v_y belongs to the adversarial team. Vertex v'_y is a target vertex of a_y. Additionally, this gadget contains a path $w'_y \ldots u_y$ with an agent from T_{t^*} (u-agent), that begins at u_y and its target is at the vertex w'_y.

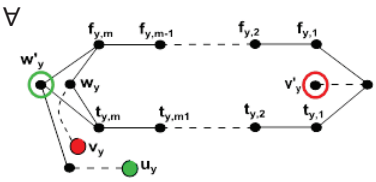

Figure 5. ***Universally*** *quantified variable gadget.*

For every clause c_j there will be a path of the length $n/2 + 1$ with an agent placed at $v_{c_j,1}$ (see Figure 6) and an extra target vertex v'_{c_j} for the agent.

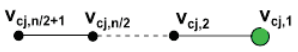

Figure 6. *Clause gadget*

Now we will put all these gadgets together. Consider following example. For the given formula

$$\varphi: \exists x \forall a \exists y \forall b \exists z \forall c$$
$$(b \vee c \vee x) \wedge (\neg a \vee \neg b \vee y) \wedge$$
$$(a \vee \neg x \vee z) \wedge (\neg c \vee \neg y \vee \neg z)$$

we construct ACPF instance depicted in Figure 7. Clause gadget is connected to variable gadgets of that corresponding variables appearing in the clause. If x is the i-th variable and appears in the j-th clause as a positive literal, then there is an edge between $v_{c_j,\lfloor i/2 \rfloor+1}$ and $f_{x,j}$. If x is a negative literal, then the edge connects $v_{c_j,\lfloor i/2 \rfloor+1}$ and $t_{x,j}$. Note that paths from initial to target vertex for all agents except those starting from vertices u are of the equal length.

Now we can finish the proof of proposition 1.

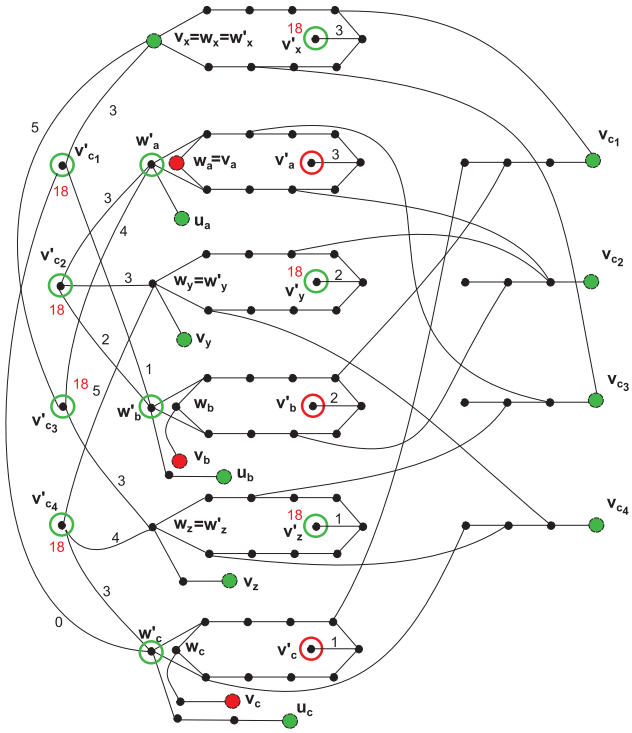

Figure 7. *Example of* ***reduction****.* Selected and adversarial teams are associated with green and red color respectively. Filled circles represent agents, unfilled are their targets. Numbers near some vertices indicate, that the vertex can be entered by an adversarial gent in certain time step (hazardous vertex). Arcs labeled with a number contain that many vertices that are not explicitly displayed. Selected team's turn comes first.

Proof. Suppose that φ to be valid. Variables are assigned their truth value gradually one by one according to the order in the quantifier part of φ. Every sequence of valuation showing validity can be considered as a guideline for agents of the selected team and lead them to a winning position. Evaluation of a variable corresponds to one time step. Every time a variable is valuated, another agent in the constructed ACPF instance is ready to enter upper or lower path within the variable gadget. If some variable x is evaluated as true, corresponding agent a_x enters the lower path (vertex $t_{x,m}$), otherwise the agent goes to the upper path (vertex $f_{x,m}$).

Since the valuation satisfies the formula, every clause c_j has at least one variable x that causes satisfaction of c_j. In such case, the clause gadget is connected to the variable gadget through a vertex in the other path than the one through which agent a_x advances towards its target. Clause agent can then enter the variable gadget without encountering a_x. Both agents can continue undisturbed. A purpose of u-agents is to prevent opponent's agents from unwanted behavior. For example u-agent is able to ensure, that no

opponent's agent will occupy any clause agent's target. Whenever any opponent's agent moves unexpectedly, u-agent from the same gadget works as a booking vertex from the booking lemma. Hence, opponent's team has no chance to avert victory of the selected team.

Whenever there exists a winning strategy for the constructed ACPF instance, variable and clause agents must reach their targets on time. This is possible only in case variable agents and clause agents do not meet on the horizontal path. They must use different paths. The variable agents' selection of upper or lower paths determines the evaluation of corresponding variables. If a variable agent and clause agent pass by each other on the parallel horizontal paths, corresponding variable causes satisfaction of the clause. ■

Now we will show the upper bound of complexity we have found so far. Let us recall that $EXPTIME = \bigcup_{k\in\mathbb{N}} DTIME\left(2^{n^k}\right)$. As a byproduct, the membership to the $EXPTIME$ class shows that the decision version of the ACPF problem is **decidable**.

Preposition 1. *Decision problem whether there exists a winning strategy for selected team in a given ACPF instance belongs to EXPTIME complexity class.* ■

Proof. If n denote number of vertices and k denote total number of agents, then the number of distinct placements is $2\binom{n}{k}k!$ with its maximum for $k = n$ we get up to $2n!$ possible placements. Search space of an ACPF instance can be regarded as an AND-OR tree. From the Stirling's approximation we infer $DTIME(n!) \in DTIME\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right) \in EXPTIME$, thus an algorithm with detection of already visited states that traverses entire tree runs in exponential time. ■

IV. Solving Approaches

Practical ACPF solving represents a challenging problem due to its high complexity. In this section, we suggest several solving approaches that are inspired in game theory and in CPF without adversaries (adversarial agents may be considered merely as movable obstacles).

A. Game Theoretic Approach

If ACPF is regarded as an n-player game [9], then utilization of algorithms and domain independent heuristics known from board games such as *Go*, *Chess* or other combinatorial games is applicable. These algorithms involve *minimax* with *alpha-beta pruning* [12] or variants of *Monte Carlo Tree Search*, that were recently successfully applied in *Go* playing program [11].

Common property that ACPF shares with many board games is that agents move in the environment and take turns to move. The most significant difference consists in generality: ACPF defines neither any particular graph nor number of agents and locations of their initial and target vertices. In addition, every time a team in on turn, all its members can be relocated simultaneously. This causes that the size of the search space grows exponentially with the number of agents and makes the usage of search algorithms impossible for even relatively small number of agents.

B. Greedy Methods

The most straightforward approach to solving ACPF is the use of greedy algorithms. In this case, we need to be able to generate all the available moves from a current position and select the one that leads to a position with highest score. As the space of all possible moves from a certain position is exponential in the number of agents, an ordering of agents is considered. Then agents are taken one by one in the given ordering – the set of possible moves in considered for a single agent only.

C. CPF-based Methods

We also consider methods designed for CPF. In particular, we propose an extension and adaptation of Cooperative A* (CA*) algorithm introduced in [6]. CA* works as follows: the graph is expanded into the 3rd dimension, such that t copies (levels) of the original graph are piled up and represent time dimension of the movement of the agents. The edges between vertices within the level are removed, while neighboring levels are connected according to the edges in the original graph. The algorithm processes agents one by one and finds paths to their target vertices in the time-expansion graph. Vertices that belong to a path are reserved and paths for every next processed agent must not contain any reserved vertex.

Algorithm 1. Adversarial Cooperative A*
Input: ACPF instance $\Sigma = (G, A, \mathcal{T}, t^*, \lambda_0, \lambda_+, \hat{\alpha})$

```
1:  paths ← initialize empty paths each member of T_{t*}
2:  do
3:      move ← select next move from paths for agents of T_{t*}
4:      if move is currently illegal then
            assign new targets to agents without pre-defined targets
5:          paths ← re-plan paths using CA* from current position
6:          move ← select next move from paths for all agents
7:      play(move)
8:      opponent's turn
        while terminal condition not satisfied
```

We extend described algorithm by the adversarial element and introduce *adversarial cooperative A** (ACA*). At the beginning, agents from the selected team are sorted, so that agents with lower number of accessible adjacent vertices should be processed before the agents with larger freedom. The next step assigns targets. If agent a has nonempty set of target vertices $\lambda_+(a)$, its path will be planned to the nearest vertex from $\lambda_+(a)$. In case $\lambda_+(a) = \emptyset$, a's path will be planned to the nearest target vertex of some adversarial

agent. When all the targets are determined, the search in the extended graph begins. We have to check, whether the next move according to the planned paths is still legal at every time step. Since the adversarial agents also change their locations, it is possible that the scheduled movement of the selected team is no longer feasible. In such case we have to re-plan. Algorithm 1 summarizes the described process.

V. Experimental Evaluation

We have performed an experimental evaluation in order to determine how suggested strategies perform in scenarios, which are expectable in practical situations. The definition of the ACPF problem is very general and allows large variety of instances, while many of them do not represent any practical situation. Hence, we focused on few types of instances only, which we call *instance scenarios*.

A. Experimental Setup

An instance scenario defines a graph with two subsets of vertices – an initial region and goal region for each team. The initial region of team t determines all possible starting positions for members of t, while goal region determines possible target vertices for agents of team t. We considered three instance scenarios: *exchange*, *race* and *mingled* (see Figure 8). All the instance scenarios take place on 4-connected grids with regularly placed obstacles. Initial locations and goals of individual agents are selected randomly within corresponding initial and goal regions respectively.

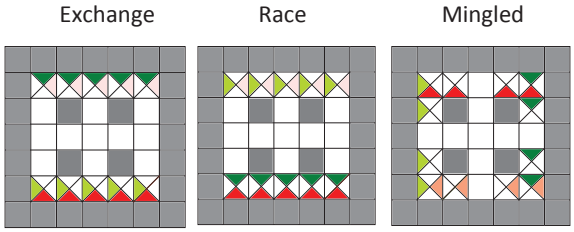

Figure 8. *Testing **instance scenarios** - exchange, race and mingled.* Green color is associated with the selected team, red color belongs to the opponent. Dark and light shades represent initial and target locations respectively. Agents and their targets are placed randomly within areas of the same color. Instance patterns also include number of agents and their targets.

B. Greedy Strategy Tournament

The first collection of experiments is aimed on revealing which greedy strategy is the most suitable in suggested ACPF instance scenarios. We focused on greedy strategies with different ways of selection of the next move. The greedy strategy always makes a locally optimal step, which means that a move that leads to a placement of agents with the best value is always performed. The value of a placement of λ_i at time step i for the selected team is calculated as follows:

$$\mathrm{val}_{t^*}(\lambda_i) = \sum_{a \in T_{t^*}} \mathrm{distance}_G(\lambda_i(a), \lambda_+(a))$$

Where $\mathrm{distance}_G(x, y)$ is a length of the shortest path connecting vertices x and y in graph G while the use of certain vertices is forbidden. We tried various restrictions of what vertices are allowed.

(a) all the vertices are allowed
(b) vertices that does not contain agents of T_{t^*} are allowed
(c) vertices that does not contain agents from $A \setminus T_{t^*}$
(d) unoccupied vertices only are allowed
(e) vertices that does not contain an agent that is at its target

Tables with results are roughly symmetric. The small observable amount of asymmetry can be attributed to the fact, that it is more advantageous for certain strategies to take either the first or the second turn.

Table 1. ***Small tournament*** *of greedy strategies.* Testing scenarios over a 4-connected grid of size 5×5 (5×6 in race scenario) with various placements of initial and goal positions of agents as depicted Figure 8 in were used. Both teams have 3 agents with a single target. All the possible pairs greedy of strategies playing against each other were tried. Score after playing 100 ACPF instances for each test case is reported. Greedy strategies (a), (b) and (e) tend to outperform the other strategies.

Small exchange

row:column	(a)	(b)	(c)	(d)	(e)
(a)	-	29:71	70:30	72:28	48:52
(b)	50:50	-	74:26	73:27	52:48
(c)	28:72	23:77	-	42:58	29:71
(d)	34:66	26:74	55:45	-	29:71
(e)	51:49	34:66	74:26	72:28	-

Small race

row:column	(a)	(b)	(c)	(d)	(e)
(a)	-	34:57	49:51	54:46	40:60
(b)	47:53	-	51:49	50:50	46:54
(c)	39:61	38:62	-	51:49	33:67
(d)	40:60	44:56	43:57	-	37:63
(e)	42:58	48:52	59:41	53:47	-

Small mingled

row:column	(a)	(b)	(c)	(d)	(e)
(a)	-	51:49	74:26	73:27	53:47
(b)	44:56	-	63:37	65:35	52:48
(c)	36:64	38:62	-	55:45	42:58
(d)	33:67	39:61	58:42	-	27:73
(e)	56:44	56:44	71:29	75:25	-

Described five modification of the greedy strategies played against each other. Table 1 and Table 2 show the results. It is noticeable, that strategies that rather disregard vertices occupied by opponent – that is strategies (a), (b) and (e) – outperform those, that take agents into consideration in most cases.

Table 2. ***Big tournament*** *of greedy strategies.* The same testing scenarios were used but the size of grids increased to 9×10 with 6 agents in each team. Again, greedy strategies (a), (b) and (e) dominate over the remaining strategies.

Large exchange

row:column	(a)	(b)	(c)	(d)	(e)
(a)	-	18:82	69:31	78:22	49:51
(b)	74:26	-	77:12	88:12	68:32
(c)	30:70	17:83	-	58:42	22:78
(d)	26:74	9:91	38:62	-	19:81
(e)	45:55	57:43	80:20	78:22	-

Large race

row:column	(a)	(b)	(c)	(d)	(e)
(a)	-	59:41	65:35	67:33	45:55
(b)	50:50	-	68:32	73:27	39:61
(c)	38:62	39:61	-	61:39	30:70
(d)	38:62	35:75	43:57	-	27:63
(e)	51:49	54:46	74:26	72:28	-

Large mingled

row:column	(a)	(b)	(c)	(d)	(e)
(a)	-	56:44	68:32	75:25	49:51
(b)	45:55	-	64:36	73:27	43:57
(c)	33:67	31:69	-	72:28	31:69
(d)	26:74	16:84	24:76	-	21:79
(e)	53:47	59:41	75:25	83:17	-

This finding might seem counterintuitive, however the explanation is simple: when a strategy takes opponent's agents into account, it tends to select moves try to avoid the adversaries, which clears the path for them. Adversarial agents can then reach their targets easier. This property is apparent in all scenarios although it is less significant in the race scenario than in the other two. The interaction among agents (an agent wants to enter occupied vertex) is less likely to occur in the race scenario, which partly reduces the differences between considered greedy approaches.

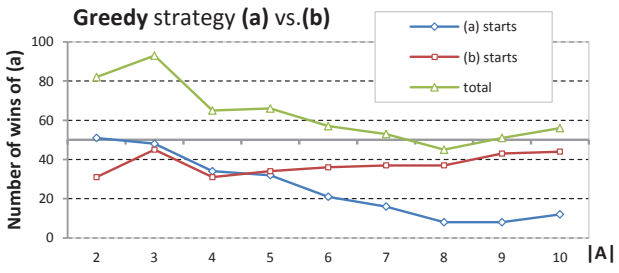

Figure 9. *Comparison of the most successful* ***greedy strategies****.* The number of wins of the team playing the greedy strategy (a) against a team playing the greedy strategy (b) in instances over a grid of size 11×12 is shown. In total, 200 instances of the type exchange in which 100 times (a) starts and 100 times (b) starts were used. It can be observed that the performance of the (a) strategy degrades with the increasing number of agents.

The two most successful greedy strategies (a) and (b) were compared separately. The test was targeted on the performance for the increasing number of agents. Results are shown in Figure 9. It can be observed that the (a) greedy strategy has better performance for few agents and its performance degrades as the number of agents increases.

C. Full-Spectrum Strategy Tournament

Several conceptually different strategies played against each other a tournament in order to find out which strategy performs as best. Again all the pairs of competing strategies were evaluated. Four strategies participated in the tournament. Every pair of strategies was tested over 100 instances conforming to suggested instance scenarios. Results are summarized in Table 3 and Table 4.

Table 3. ***Small tournament*** *of all strategies.* Considered scenarios were tested on instances of size 5×5. 1 out of 3 agents in each team did not have any particular target. Classic alpha-beta algorithms seem to be more successful than others.

Small exchange

row:column	**Greedy**	**ACA***	**α-β**	**MCTS**
Greedy	-	40:60	44:56	48:52
ACA*	53:47	-	51:49	48:52
α-β	60:40	65:35	-	64:36
MCTS	47:53	35:65	41:59	-

Small race

row:column	**Greedy**	**ACA***	**α-β**	**MCTS**
Greedy	-	27:73	40:60	35:65
ACA*	67:33	-	60:40	58:42
α-β	51:49	29:71	-	65:35
MCTS	49:51	35:65	39:61	-

Small mingled

row:column	**Greedy**	**ACA***	**α-β**	**MCTS**
Greedy	-	19:81	31:69	37:63
ACA*	76:24	-	74:26	68:32
α-β	75:25	43:57	-	58:42
MCTS	50:50	23:77	39:61	-

It can be observed that alpha-beta algorithm wins in majority of cases in small instances. The large size of the search space in case of game-based strategies becomes evident in large instances where these strategies are often defeated. Success rate of the cooperative approach (ACA*) is besides other things caused by agents without targets, that are used for capturing opponent's targets.

Number of agents in instances, where these methods can be employed is very limited due to the exponential search space growth. Instances containing tens of agents must be solved by either greedy or cooperative methods.

Like in the case of greedy strategies, a more detailed comparison of two best strategies overall has been conducted. Greedy strategy (a) and the ACA* strategy were compared for the increasing number of agents without targets.

Results are shown in Figure 10. The performance of ACA* degrades for higher number of agents without targets.

Table 4. ***Large tournament*** *of all strategies. Again, c*onsidered scenarios were tested on instances of size 9×10, while 2 out of 6 agents in each team were without target. Classic alpha-beta algorithms seem to be more successful than others. Cooperative approach (ACA*) algorithm is more successful than other methods.

Large exchange

row:column	Greedy	ACA*	α-β	MCTS
Greedy	-	27:73	55:45	73:27
ACA*	67:33	-	72:28	70:30
α-β	54:46	23:77	-	74:26
MCTS	38:62	30:70	27:73	-

Large race

row:column	Greedy	ACA*	α-β	MCTS
Greedy	-	26:74	37:63	72:28
ACA*	82:18	-	84:16	85:15
α-β	43:57	28:72	-	64:36
MCTS	28:72	13:87	21:79	-

Large mingled

row:column	Greedy	ACA*	α-β	MCTS
Greedy	-	8:92	50:50	59:41
ACA*	75:25	-	72:28	65:35
α-β	44:56	14:86	-	70:30
MCTS	36:64	31:69	33:67	-

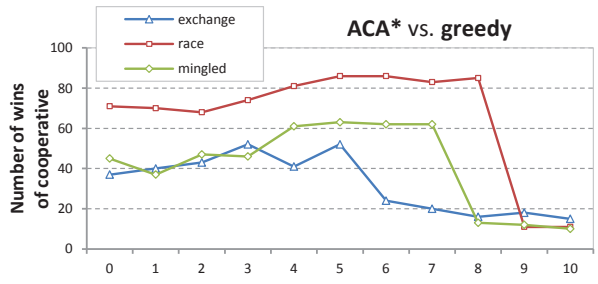

Figure 10. *Comparison of the overall most* ***successful strategies****.* The ACA* strategy and the greedy strategy (a) are compared in instances over a grid of size 11×12 with 11 agents on both sides. The performance of strategies is shown for the increasing number of agents without any targets (on each side). The performance of the cooperative strategy degrades with increasing number of agents without targets.

VI. Conclusions, Discussion, and Future Work

We have studied the problem of adversarial cooperative path-finding (ACPF). It can be regarded as the extension of the standard cooperative path-finding (CPF) with adversarial agents. The task in ACPF is to control movements of agents so that they reach their targets before the adversarial agents.

The major challenge in ACPF consists in designing an efficient decision strategy that leads controlled agents to the victory. Designing such a strategy is a difficult task. Note that the strategy needs to take into account all the possible actions of adversarial agents and their reactions.

The ACPF problem is studied theoretically as well as from the practical point of view. Regarding the theory, we studied the complexity of the decision version of ACPF where we are interested in the question whether there exists a winning strategy. We have shown that decision version of ACPF is *PSPACE*-hard. It remains an open question whether it belongs to *PSPACE*. We are not aware of any polynomial upper bound on the size of the solution of ACPF, which would allow us to prove membership of ACPF into *PSPACE* (note that in case of CPF such bound exists). Next, we have shown that the problem is in *EXPTIME*, which by the way also show that it is decidable.

Regarding practical solving of ACPF we suggested several approaches – simple greedy method, game theoretical methods, and methods based on CPF solving. Game theoretical methods include Monte-Carlo tree search and alpha-beta tree search. CPF-based solving is in fact a re-planning method. All these methods were compared in a tournament from which a quite counter-intuitively greedy methods and CPF-based methods came out as winners defeating game-based methods.

References

[1] Hearn, R. A., Demaine, E. D. *PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation.* Theoretical Computer Science, Volume 343(1-2), Elsevier, 2005, pp. 72-96.

[2] Kornhauser, D., Miller, G. L., Spirakis, P. G., "Coordinating Pebble Motion on Graphs, the Diameter of Permutation Groups, and Applications", Proceedings of the 25th Annual Symposium on Foundations of Computer Science (FOCS 1984), IEEE Press, 1984, pp. 241-250.

[3] Ivanová, M., Surynek, P., "Adversarial Cooperative Path-Finding: A First View", Late-Breaking Developments in the Field of Artificial In telligence (AAAI 2013 Late-Breaking Developments), AAAI Press 2013.

[4] Ratner, D., Warmuth, M. K. "Finding a Shortest Solution for the N × N Extension of the 15-PUZZLE Is Intractable", Proceedings of AAAI 1986, Morgan Kaufmann, 1986, pp. 168-172.

[5] Ryan, M. R. K. "Exploiting Subgraph Structure in Multi-Robot Path Planning", Journal of Artificial Intelligence Research (JAIR), Volume 31, AAA Press, 2008, pp. 497-542.

[6] Silver, D. "Cooperative Pathfinding", Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2005), AAAI Press, 2005, pp. 117-122.

[7] Sipser, D. "Introduction to the Theory of Computation", CENGAGE Learning Custom Publishing, 2012.

[8] Wang, K. C., Botea, A. "MAPP: a Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees", JAIR, Volume 42, AAAI Press, 2011, pp. 55-90.

[9] Watson, J. "Strategy: An Introduction to Game Theory", W W Norton & Company Incorporated, 2012.

[10] Yu, J., LaValle, S. M. "*Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs",* Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI 2013), AAAI Press, 2013.

[11] Browne, C., Powley, E. J., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. & Colton, S. "A Survey of Monte Carlo Tree Search Methods", *IEEE Trans. Comput. Intellig. and AI in Games* 4 (1) , 27-29 .

[12] Poole, P. C., Fuller, S. H., Gaschnig, J. G., Gillogly, J. J. "Analysis of the Alpha-Beta pruning algorithm", Technical report, Carnegie-Mellon University, Pittsburgh, PA, July 1973.