Agent Subset Adversarial Search for Complex Non-cooperative Domains

Viliam Lisý, Branislav Bošanský, Roman Vaculín and Michal Pěchouček

***Abstract*— We investigate reduction of the complexity of a multi-agent adversarial search in the domain of n-player games. We describe a method that decomposes the game into a set of smaller overlapping sub-games, solves each sub-game separately, and then combines the results into a global solution. This way, the exponential dependence of the adversarial search complexity on the number of agents can be reduced to polynomial. Still, the proposed method performs well in the domains with sparse agents' interactions. The method can be used with a generic adversarial search algorithm. For the experimental evaluation, we implement it on top of an existing adversarial search algorithm designed for complex domains and we evaluate its performance on a game, which is a model of a humanitarian relief operation in conflict environment. The results show that the method often finds the same solutions as the complete search, but the search efficiency is significantly improved.**

I. Introduction

Recently, there has been a growing interest in studying complex systems, in which a number of agents concurrently pursue their goals while being engaged in complicated patterns of mutual interaction. Examples can be found in real-world systems, such as information and communication networks or social networking applications, as well as simulations, including models of societies, economies and/or warfare. Agents operating in these systems are part of a single shared environment; hence the outcomes of their actions depend on actions chosen by other agents. Such scenarios are often termed *games* and have been of an interest of AI research from its very beginning.

With the increasing complexity of environments in which the agents interact, classical AI algorithms, such as max^n search [1], become unusable due to the huge branching factor and the size of the state space. One of the key problems is the consideration of all possible interactions of all agents involved in the game – in general, the search complexity is exponential in the number of agents involved. Therefore, usually in game-playing only a very small number of agents is considered. The inter-agent interactions cannot be completely neglected as the interactions substantially change the development of the game. However, in many domains each agent interacts only with a small number of agents in a limited time frame, and the collisions and interactions with other agents are rather sparse. We have therefore developed a method that applies this insight to make search-based game-playing algorithms more efficient.

Agent Technology Center, Dept. of Cybernetics, FEE, Czech Technical University, Technická 2, 16627 Prague 6, Czech Republic, {lisy, bosansky, vaculin, pechoucek}@agents.felk.cvut.cz

The main idea of the presented approach is to substitute one global search in the search space of all agents with multiple sub-searches in the search spaces of small overlapping *subsets* of the agents, and use the information obtained through the sub-searches to synthesize a global solution. We term this method *Agent Subset Adversarial Search (ASAS)*.

We show that if the size of the subsets is limited, the ASAS method can be polynomial in the number of agents involved. The ASAS method is generic and it can be used with multiple known search-based game playing algorithms, such as max^n [1], or its extensions (e.g., [2], [3]).

The major contribution of this paper is the ASAS method, which substantially improves the search efficiency with a relatively small decrease of quality of produced plans and without requiring any background knowledge about the domain. The method is suitable for domains where the most of significant changes in the state of the game need only relatively small number of agents to interact. These changes can be either of positive or negative nature for an agent, i.e., some agents want to achieve the change which other agents want to prevent. In the ASAS method, the agent subsets do not have to be defined explicitly. Instead, the "correct subsets" leading to high-quality plans emerge from the sub-searches for many different subsets.

The remainder of the paper is structured as follows. Section II introduces preliminaries and Section III presents the proposed ASAS method. Section IV presents an experimental evaluation of the ASAS method. Section V reviews the related work and Section VI concludes the paper and outlines the future work.

II. Preliminaries

The problem tackled in this paper is to determine a suitable course of actions (a *plan*) for an agent in complex multi-agent environments that can be modeled as *n-player non-zero-sum games*. We consider games, in which all agents are self-interested, i.e., each agent maximizes only its own utility. We focus on the environments whose complexity prohibits the application of classical game-playing algorithms based on the exhaustive state space search because of its prohibiting size.

The domains for the proposed algorithm can be formalized as a tuple $(\mathcal{I}, \mathcal{A}, \mathcal{W}, \tau, \mathcal{U})$.

- $\mathcal{I}$ is a finite set of agents (players) – indexed by $1, \ldots, n$ – capable of performing actions in the world.
- $\mathcal{A}_i$ is a set of actions an agent $i \in \mathcal{I}$ can perform, and $\mathcal{A} = \mathsf{X}_{i \in I}\, \mathcal{A}_i$ is a set of joint actions where each $\vec{a} \in \mathcal{A}$ ($\vec{a} = \langle a_1, \ldots, a_n \rangle$, $a_i \in A_i$) denotes a joint action.

- $\mathcal{W}$ is the set of possible world states
- τ: $\mathcal{W} \times \mathcal{A} \rightarrow \mathcal{W}$ is the transition function realizing one *move* of the game, where the game world is changed via joint action of agents and the world's own dynamics.
- $\mathcal{U} = (u_1, \ldots, u_n)$ denotes the global utility function, where $u_i : \mathcal{W} \mapsto \mathbb{R}$ is a real-valued utility function for player i on world states $\mathcal{W}$.

The set of the world states is represented by values of variables $\{x_1, \ldots, x_m\} = D$, i.e., $\mathcal{W}$ is the set of all valuations of these variables. We denote the set of all valuations of a set of variables S by $\bar{S}$ (i.e., $\bar{D} = \mathcal{W}$). Each action $a_i \in A_i$ uses a subset $\mathsf{pre}(a_i) \subseteq D$ of variables in its preconditions and it modifies a subset $\mathsf{eff}(a_i) \subseteq D$ of variables by its execution. Finally, we define a plan of an agent i as a finite sequence of actions, i.e., $p = (a_1, \ldots, a_d)$, $a_j \in A_i$, $j \in \{1, \ldots, d\}$.

We already mentioned that the main idea of the ASAS method is to substitute the global search by multiple sub-searches in small agent subsets. We employ the notion of *plan independence* in splitting the whole set of agents into subsets and, later, for merging the partial solutions into a global solution. For simplicity, assume that we have two agents 1 and 2. Let $P_1 = \{p_1, p_2, \ldots, p_l\}$ denote the set of some plans of the first agent, where $p_i = (a_1^i, \ldots, a_d^i)$, $i = 1, \ldots, l$ are plans composed of actions of the first agent. For such plans of agent 1, we define the sets of variables occurring in the plans preconditions and effects as follows:

$$\mathsf{pre}(p_i) = \bigcup_{j=1}^{d} \mathsf{pre}(a_j^i)\ , \qquad \mathsf{pre}(1) = \bigcup_{i=1}^{l} \mathsf{pre}(p_i) \tag{1}$$

$$\mathsf{eff}(p_i) = \bigcup_{j=1}^{d} \mathsf{eff}(a_j^i)\ , \qquad \mathsf{eff}(1) = \bigcup_{i=1}^{l} \mathsf{eff}(p_i) \tag{2}$$

For the second agent, P_2, $\mathsf{pre}(2)$ and $\mathsf{eff}(2)$ can be defined in a similar fashion.

Definition 1 *We say that sets of plans P_1 and P_2 of agents 1 and 2 are* independent *iff*

$$(\mathsf{pre}(1) \cup \mathsf{eff}(1)) \cap \mathsf{eff}(2) = \emptyset \ \ \& \ \ (\mathsf{pre}(2) \cup \mathsf{eff}(2)) \cap \mathsf{eff}(1) = \emptyset$$

Agents' plan independence, as illustrated in Figure 1, expresses that the plans of agents 1 and 2 cannot interfere. Clearly, if plans of agents 1 and 2 are not independent, the success and the utility of execution of such plans of one agent can depend on the plan that is selected by the other agent. On the other hand, if plans of agents 1 and 2 are independent and if P_1 and P_2 contains *all possible plans* of agents 1 and 2 respectively, we can be sure, that agents 1 and 2 can never interfere. Therefore, in such a case, the search for an optimal plan for agents 1 and 2 can be performed independently.

If more than two agents are considered, we are interested in dividing the set of agents to disjoint subsets, such that the plans for the agents in each subset can be computed independently from the plans of the agents in any other subset. We use the notion of pairwise agents' plans independence to generalize the agent's plan independence to agent subsets.

Definition 2 *Assume an undirected graph $G = (I, E)$, where the set of nodes corresponds to the individual agents and the edge $\{i, j\}$ is present if the plans of agents i and j are not independent. We call G the* graph of agents' plan dependence.

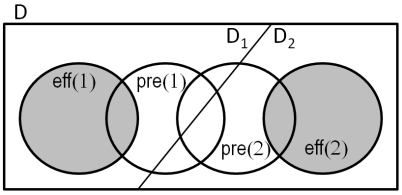

Fig. 1. The general case of inclusion of the variable sets defining plan independence.

If graph G is not connected, then the subsets of nodes corresponding to individual connected components of the graph create a decomposition of the set of agents to the parts that can be evaluated independently.

The benefits of decomposing the global game into subgames can be expressed in terms of computational complexity of the search process. Let n be the number of agents in the game, b be the number of different actions applicable by an agent in single move (branching factor), and d be the desired look-ahead, i.e., the number of moves in the future, which we want to consider for determining the plan. If we consider a game progressing in time with simultaneous actions of all agents, the size of the search space is approximately

$$b^{n*d} \tag{3}$$

If the set of all agents can be decomposed into disjoint independent subsets, such that the optimal plan of the agent we plan for depends on plans of at most $(k-1)$ other agents in the game, the complexity of planning for the agent becomes b^{k*d}.

The major problem of the described notion of agents' plan independence is that in realistic environments we almost never encounter the situation of complete agents' plans independence. In other words, while in a small local neighborhood of an agent the agents' plans independence would be a reasonable criterion for splitting agents into subsets, if a more global planning perspective is needed, the agents' plans independence cannot be used directly.

III. ASAS: The Method

The ASAS method relies on the observation that explicit coordination of *all* agents in the game is often not necessary in order to find high-quality plans that reach agents' goals (e.g., in some domains the agents are specialized and interact only with the agents of similar role and/or task group). Therefore, the set of all agents can be decomposed into small subsets, and planning can be performed independently for each subset. We use the term *active agents* for such agents that are members of the subset for which the sub-search is currently performed. The remaining agents, which are not in the subset, are called *inactive agents* (with respect to the specific sub-search).

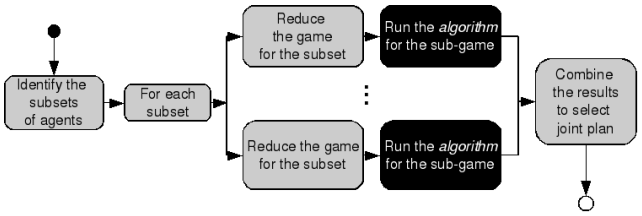

Fig. 2. The scheme of the ASAS method using a generic search-based algorithm.

The ASAS method is designed to be combined with some generic search-based game playing algorithm, such as max^n , and tries to find the *optimal plan*, i.e., the one that would be found by underlying search algorithm without using the ASAS method. Figure 2 shows the overall scheme of the method, which in essence consists of the following steps:

1) **Subsets selection**: Create a set of sub-games involving a small number (k) of active agents in each sub-game (Section III-A). These subsets of active agents do not need to be independent and can be overlapping.
2) **Sub-searches**: For each sub-game use the underlying search algorithm to compute the plans which the active agents should pursue in the sub-games. In sub-searches the problem arises of how to reason about the game development without considering the inactive agents (Section III-B)
3) **Merging**: Combine the results from the sub-searches into a single global solution (Section III-C)

An agent uses ASAS in the same way as the underlying adversarial search algorithm. It computes its optimal plan centrally (on its own), without interaction with other agents, based on his model of the game and the utility functions of all agents.

A. Subsets Selection

There are several options of how to choose a set of subsets in the first step of the algorithm. A rather simple method is to consider all possible subsets of k agents. Even such basic selection method reduces the search complexity substantially. The reason stems from the complexity of computing plans in all sub-games. If all possible subsets of k agents are considered, the number of such subsets is given by the binomial coefficient. Hence, the sum of sizes of all the sub-search spaces can be approximated (with respect to formula 3) as

$$\left(\begin{array}{c} n \\ k \end{array} \right) b^{k*d} < (n^k) b^{k*d} \tag{4}$$

Note that this number is still polynomial in the number of agents in the game (n) and it form the upper bound on complexity of the method.

Alternatively, different (more sophisticated) approaches could be used for subsets selection, in order to decrease the number of subsets that needs to be explored. For example, the agents' plan dependence graph as defined in Def. 2 can be employed. Using this graph, such minimal set of subsets of k agents can be selected, for which each clique of size k or smaller is included in at least one of the subsets. This would be less than the number of all subsets of k agents if the graph is not complete.

Although selecting only some of the subsets would further reduce the search space, we use all subsets of specified size as a proof of concept in our experiments, which still allows substantial computation time reduction.

B. Reasoning about Inactive Agents

Another key issue is what to do with the agents that are not currently active in the given sub-search. We consider the following three options.

(1) Removing inactive agents from the environment can be used in domains where pure presence/absence of an agent does not substantially change the environment state. Such situations may occur, e.g., in domains, where some form of *locality* (e.g., geographical, functional) does not allow many agents to interact with each other even when a long look-ahead is considered. In our domain, as in many adversarial domains, this is not the case. For example, removing a military brigade from an unstable region would change the game completely, and plans optimal for such situation would not make sense in the game with all agents.

(2) Using predefined heuristic behavior. Heuristic behaviors, employing background knowledge of the domain, can represent tasks for inactive agents that are necessary for reasonable consideration of the future development. Conservative heuristic behavior of inactive agents will allow more realistic assessment of future game states during sub-searches. However, creating heuristic behaviors manually for all agents and all situations is non-trivial and time consuming.

(3) Reusing results from the previous search: Instead of creating the heuristics manually, the results from the *previous search* can be used as a heuristic in the current search. Specifically, for every inactive agent we use its best plan resulting from ASAS for the agent in the previous execution of the algorithm (usually in the previous move) as a heuristic approximation of its behavior. Since such plans are sequences of actions (the length corresponds to the size of the planning look-ahead), these plans can be used as a heuristic in the search for several subsequent moves, and such plans will most likely still be reasonable. If the past plan for some inactive agent is not applicable at a certain time because of a major deviation from the past game state development, the inactive agent will move randomly further on. Importantly, random moves have to be used mostly only in the later stages of the search, hence the effect of random moves does not affect the outcome of the sub-search for active agents too much.

For our experimental domain evaluation we follow the third option, which showed to suit best the used domain.

C. Sub-results Merging

The purpose of merging sub-results is to select the final plan for each agent based on information found in sub-

Input: W: current world state, I: the current set of agents, D_{util}: the part of domain to compute utility, $P[I]$: the set of plans to choose from for each agent
Output: an array of values of the world state (one value for each agent)

```
1  Compute the graph of agents' plan dependence G
2  if G is not connected then
3  |  V⃗ = 0⃗
4  |  foreach G_i connected component of G do
5  |  |  Let I_i be the agents in component G_i
6  |  |  Let D_i be the domain partition corresponding to G_i
7  |  |  V⃗ = V⃗ + Merge(W|(D_i ∩ pre(I_i)),I_i,D_i, P[I_i])
8  |  end
9  |  return V⃗
10 end
11 foreach agent i in I do
12 |  Let First_i be the set of all actions that are first at some
   |  plan from P[i]
13 end
14 foreach a⃗ = (a_1, a_2, ..., a_n) ∈ X_{i=1}^n First_i do
15 |  curW = τ(W, a⃗)
16 |  foreach agent i in I do
17 |  |  P'[i] = {p ∈ P[i] : p starts with a_i}
18 |  |  Remove first action from each plan in P'[i]
19 |  end
20 |  ActionValues[a⃗] = Merge(curW, I, D_util, P'[I])
21 end
22 return ValueBackup(ActionValues)
```

Fig. 3. Merge(W, I, D_{util}, $P[I]$) – the main procedure plan merging algorithm.

searches. For each subset the sub-search produces:

- an optimal plan for each agent in the sub-game
- a resulting utility achievable in the sub-game if all active agents apply the plans optimal for the subset
- other characteristics of the resulting plans, such as their resource requirements, preconditions, and effects

If the plans generated in some sub-game are used in the complete game, conflicts between plans originating from different sub-games may occur, and the utility of the resulting game may not be the same as in the sub-game. We developed two merging techniques that address this issue. The *search-based plan selection* achieves better performance, but re-introduces the exponential dependence of computational complexity on the number of agents. Alternatively, the *rule-based plans selection* approach guarantees the polynomial time complexity, but the quality of the selected plans in lower.

1) Search-based plan selection: In essence, the search-based plan selection tries to find the best possible combination of plans (1 plan for each agent) by exploring all combinations of agents' plans that were found as the best plan in at least one of the sub-searches. The combinations are explored by searching the search space (game tree) that is generated only by the plans of each agent produced by the sub-searches. The algorithm realized by a recursive method showed as the pseudocode in Figure 3, consists of two parts:

1) the *split* part (lines 1–10) partitions the set of agents and their corresponding plans found in sub-searches into independent agent sets (Defs. 1 and 2), for which the best combination of plans can be found separately,
2) the *selection* part (lines 11–22) considers every partition from the split part and explores all plan combinations for the agents of the partition by simulating the plans simultaneously in the shared environment and by computing the utility of every such combination of plans.

The combination of plans with optimal utility values in the sense of the underlying search method equilibrium is returned.

While we discussed the fact that agents' plan independence properties are very unlikely to hold globally for all possible agents' plans, these properties are very useful in the *split* part of the *Merge* procedure. In particular, the set of agents for which we evaluate different combinations of plans, can be split in case their remaining actions in plans are independent (in terms of Def. 2) and evaluated separately. Note, that the *split* part is an optimization of the overall search-based plan selection method that can be omitted.

In order to find the best combination of plans in the *selection* part of the procedure, one of its plans is selected for each agent, and the first actions of such selected plans are executed simultaneously (lines 14, 15). Since some of the plans of each agent can have a shared prefix (i.e., start with the same sequence of actions), as an optimization, the procedure always considers plans with shared action prefixes instead of considering only single plans at a time (lines 17, 18). The *Merge* procedure recursively traverses the tree structure formed by the shared prefixes in depth-first manner. With each descent in the tree, less and less candidate plans share the same prefix with the already executed actions, i.e., the sets $P[i]$, and the variable sets $\mathsf{pre}(i)$ and $\mathsf{eff}(i)$ are getting smaller. Subsequently, in each recursive descent of the procedure the *split* part is more likely to produce an independent partitioning of the agents into smaller subsets.

In the recursive call on line 7, the term $W|S$ means the world state W restricted to the variables in set S. The vector $\vec{V}$ represents utility values of optimal plans for each agent (not just agents in I_i), and the plus sign means vector addition. Function τ on line 15 is the state transition function. The *ValueBackup* function computes the value of the best combination of agents' possible actions (computed in the leaf nodes of the tree traversal), in the same way as the underlying search algorithm of ASAS. The utility values for each action are stored in the $ActionValues$ vector. For the sake of simplicity, the presented algorithm does not include the structures and procedures to store the plans that correspond to the selected solution.

In order to use the *split*, we assume that the utility functions of agents are *decomposable* with respect to the domain D. It means that the value of each utility function u on D can be obtained by computing some derived functions on arbitrary partitions of the domain and then combined at the end, i.e., $\forall D_1, D_2$ s.t. $D_1 \cup D_2 = D$, $D_1 \cap D_2 = \emptyset$ $\exists u_1 : \bar{D}_1 \to \mathbb{R}, u_2 : \bar{D}_2 \to \mathbb{R} \quad (u = u_1 + u_2)$.

The decomposability property of the utility functions is needed in order to allow the utility of the plans to be computed on subdomains of D in the *split* part. Specifically, for independent plans, each agent can evaluate its options independently just on the subset of the domain that is relevant for him $(\mathsf{pre}(i) \cup \mathsf{eff}(i))$. However, if we need to know the utility of performing the plans, working with just the sets pre and eff might not be sufficient. The utility function must be computed on the complete domain. That is why the agents split the domain to two disjoint parts D_1, D_2, so that $D_1 \cup D_2 = D$, such that $\mathsf{eff}(1) \subseteq D_1$ and $\mathsf{eff}(2) \subseteq D_2$ (see Figure 1). The agent 1 evaluates its plans on domain $D_1 \cup \mathsf{pre}(1)$ and computes the utility only on D_1. The agent 2 computes the utility on D_2 in a similar way. The final utility is then just sum of the utilities computed by individual agents thanks to the assumption of decomposable utility.

With respect to the complexity of the *Merge* procedure, in the phase of sub-searches, each agent is active in $\binom{n}{k-1}$ searches. In the worst case, each sub-search can produce a new plan for the agent, i.e., the final search tree would contain at most $\binom{n}{k-1}^n$ leaves. Although this number does not depend on the look-ahead factor (see formula (3)), it re-introduces the exponential dependence on the number of agents. However, our experiments show that for an agent the same plan often results from multiple sub-searches, which reduces the size of the search space.

2) Rule-based plan selection: In order to avoid the possibly exponential time complexity of the search-base plan selection, the rule-based plan selection process introduces simple rules for selecting final plan for each agent out of all best plans found in corresponding sub-searches. The rules can employ various domain-independent factors of plans in order to select a good one, such as:

- **Utility**: the maximal absolute value of the plan utility, or the maximal average utility of the plan (one plan can be found in several sub-searches with different utilities)
- **Occurrences**: the number of occurrences within the candidate plans, i.e., the plan that was returned from most sub-searches for an agent will be selected.

Such rules can be determined automatically, e.g., by using machine learning algorithms on a training set created by running the full search algorithm as well as the sub-searches for all the agent subsets on a set of scenarios. However, the focus of this paper is on the ASAS method as such, therefore we have constructed several rules combining both of these ideas manually and used them in our experimental evaluation to assess the feasibility of this approach (see Section IV).

IV. Experimental Evaluation

The ASAS method can be applied on top of various search algorithms. For the experimental evaluation, we use the goal-based game-tree search (GB-GTS) [3] as the underlying search method. This algorithm was chosen since it allows search in significantly larger search-spaces compared to more standard game-tree search algorithms.

The results for the rule-based plan selection method presented in this section were obtained by the heuristic rule that chooses the plan that is maximal in lexicographical order given firstly by the average utility value of the plan and secondly by the number of occurrences of the plan within the set of all candidate plans. This rule was the most successful rule from all the fixed rules we considered.

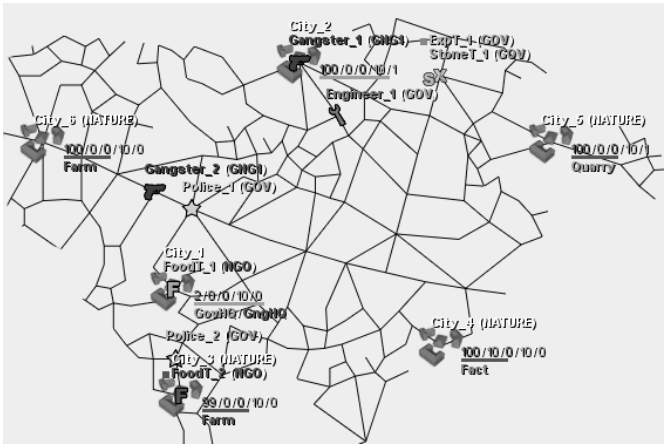

Fig. 4. One of the sample situations from the experimental scenario. Lines represent the edges in the graph, agents are represented by appropriate symbols in the vertex, and cities are shown as a group of buildings in the vertex.

A. Experimental Scenario

The game we use as a test case is a model of a humanitarian relief operation in a conflict environment with three groups of agents – government agents, humanitarian organization agents, and separatists. Agents have different capabilities and they are placed in the game world represented by a planar graph forming a network of roads. Any number of agents can be located in each vertex of the graph and each agent can change its position to an adjacent vertex in one game move. Some of the vertices of the graph contain cities, which can take in commodities the agents use to construct buildings or to produce other commodities. The utilities representing the objectives of the agents are expressed as weighted sums of components, such as the number of cities with sufficient food supply, or the number of cities under the control of the government. Because of the limited space, we explain only the key factors of the scenario used in the experiments. The detailed description of the game can be found in [4].

Figure 4 depicts the schema of the scenario used for the experiments with all important details. The scenario includes 6 cities, 2 of which can be controlled by the government as well as the separatist team (City 1 and City 2). At the beginning of the scenario, both of these cities are under the control of the government team. Government headquarters (HQ) that help assuring government's control over a city are build only in City 1 and the food reserves in this city are very small.

There are 9 agents operating in this scenario – two food trucks delivering food to City 1 from two farms (Cities 3

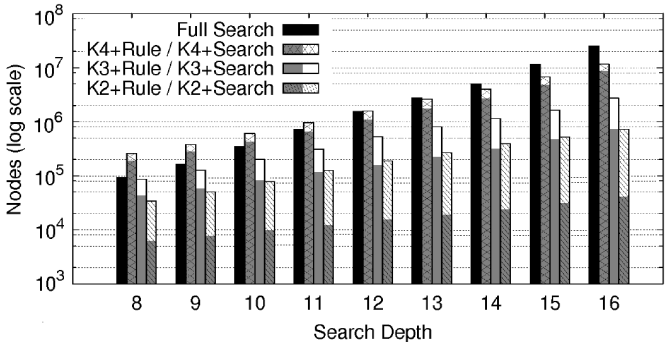

Fig. 5. The number of nodes explored by the full search compared with the ASAS method for different sizes of agent subsets and various look-ahead values. The nodes axis is in the logarithmic scale.

and 6), a truck transporting explosives from City 4 to City 5 (the explosives are used to mine stone in City 5), a truck transporting stones from City 5 to City 2 and City 1 if they do not contain HQ (in order to build it). Next agents are an engineer that uses the stone to build HQs, two police cars that can escort trucks and gain control in cities and finally two gangsters that can either gain control for separatists in a city without HQ, destroy food in order to cause riots that lead to the destruction of an HQ in a city, or steal explosives and create a suicide bomber in their cell in City 1. The suicide bomber is an agent that can destroy HQ by a suicide attack. There are many possible runs of this scenario. For example the police agents have to deal with several threats (e.g. protecting two food transports, explosives transport, and control of City 2). The gangster agents try to gain control of the cities they do not control and the humanitarian organization team tries to provide supplies efficiently without being robbed.

In our experiments, we compare the ASAS method with the original search method (GB-GTS in our case) without the ASAS being applied (termed *full search*). We measure the complexity (expressed as the number of explored search nodes), and optimality of the found plans. In our basic evaluation scenario, the game is played on a graph that consists of several hundred nodes. Thanks to the background knowledge in GB-GTS, the average branching factor of each agent is approximately 2.5 (minimal branching factor is 1, maximal is equal to 4), and each agent has to decide about it next course of action approximately every 5 moves.

The experiments were performed for different size of subsets $k = 1, \ldots, 4$, different look-ahead values $d = 8, \ldots, 16$, and different number of agents present in the scenario $n = 7, \ldots, 12$. The lower bound of the look-ahead was chosen as a minimal look-ahead that produces reasonable behavior in our scenario. The upper bounds were given by a practical limit of computational resources, especially when comparing to the full search. In each experiment, we ran the game on 42 situations based on the described scenario.

B. Search Reduction

We evaluate *search space reduction* by comparing the number of search nodes explored by the ASAS method and by the full search. In results regarding the ASAS method, we distinguish between the nodes evaluated in the sub-searches phase and the nodes evaluated during the search-based plan selection. The number of nodes evaluated in the sub-searches is the same for both plan-selection methods, however for the rule-based plan selection, this number is the final one. For the search-based plan selection, the overall complexity equals to the sum of nodes explored in the sub-searches and the nodes explored in the plan selection search. Furthermore, the complexity of computations performed in nodes of sub-searches might differ from computations in plan selection search nodes. In the presented experiments, the complexity of computation of all three kinds of nodes is the same, which allows us to compare these nodes and to depict them in the same graph.

The search space reduction for varying look-ahead values is shown in Figure 5. For the ASAS method, the bars show the overall sum of all nodes explored with the search-based plan selection method applied, while the number of nodes evaluated only in sub-searches is highlighted by the gray filling.

As expected, the search space size grows exponentially with the increasing look-ahead in all cases (note that the figure scale is logarithmic). However, the number of nodes explored in sub-searches is a rather small portion of the overall number of explored nodes. For example, for the look-ahead 16 the full search explored $25,429,065$ nodes, while the ASAS for subsets of size $k = 3$ with search-based plan selection method explored $2,728,847$ nodes (i.e., 9.32 times less than the full search) and the ASAS with the rule-based plan selection explored only $722,743$ nodes (i.e., 35.19 times less than the full search).

The search space reduction for varying number of agents is depicted in Figure 6(a). In this experiment we evaluated the dependence of complexity on the number of agents involved in the search. For the look-ahead 16 and two different sizes of subsets ($k = 2, 3$) we measured the number of evaluated nodes. For the full search and the ASAS with search-based plan selection, the number of nodes grows exponentially (the figure scale is again logarithmic). However, the number of nodes evaluated in sub-searches (i.e., if the rule-based plan selection was used) remains polynomial in the number of agents.

C. Accuracy: Preservation of Optimal Plans

Next, we focus on the accuracy of the ASAS method. We show that in spite of the significant reduction of the search space, the ASAS algorithm finds the same solution as the full search in most cases. We evaluate the accuracy by matching the plans returned by both methods for each agent. Two plans are matching, if they are exactly the same. This criterion is stronger than just comparing the utilities of returned plans – matching plans imply matching utilities.

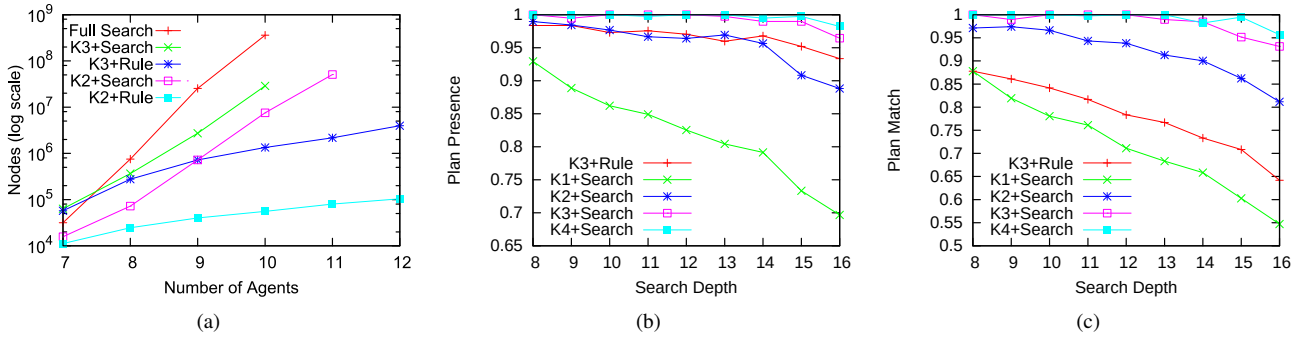

Fig. 6. (a) The number of nodes (the axis is in logarithmic scale) explored by the full search compared with the ASAS method for different number of agents, fixed look-ahead (16), and different size of subsets $k = 2, 3$; (b) The percentage of optimal plan existence for all agents among the candidate plans resulting from the sub-searches in ASAS for various look-ahead values; (c) The percentage of optimal plans returned by the ASAS method for all agents for various look-ahead values.

We use plans matching since it captures very well the loss of quality of plans returned by the ASAS method as compared to plans found by the original adversarial search algorithm.

First, we measured the *presence rate*, i.e., how often the optimal plans (the plans found by the original full search algorithm) are present among the candidate plans found by the sub-searches. The presence rate corresponds to the maximal possible precision of the ASAS method in case we had an ideal plan selection strategy (i.e., the one that always selects the optimal plan from sub-search results if it is present). However, in our case the presence rate is also influenced by the used plan selection method, because the results of sub-searches depend on behavior of inactive agents, which in our case are approximated by reusing the plans from previous searches (see Section III-B).

Figure 6(b) depicts the presence rate for varying look-ahead and different size of agent subsets k. We emphasize the search-based plan selection method, but for a comparison we performed the experiments with heuristic rule-based plan selection as well. With the increasing look-ahead, the presence rate gets worse, which is due to the exponential expansion of the search space. However, for $k = 2$ the presence rate is almost always more than 90%, and for higher values of k the presence grows towards 100%. Specifically, for $k = 3$ the presence rate is almost always more than 95%. The presence rate for the rule-based plan selection heuristic is slightly worse.

Next, we analyzed the overall accuracy of the ASAS method. For each agent, we compare the plan selected by the particular selection strategy with the plan assigned to the agent by the full search. The results depicted in Figure 6(c) show that for subsets size $k = 3$ or $k = 4$ and the search-based plan selection we obtain results almost as good as the in full search. For $k = 3$ the accuracy is always higher than 95% for all evaluated look-ahead values. For the rule-based plan selection, the results are worse than search-based plan selection, however, still applicable in real game-playing situations. For both $k = 2$ and $k = 3$ we obtained the accuracy of 64% for the look-ahead of 16. Notice, that for $k = 1$, which corresponds to the approach presented in [5] (see Section V), the accuracy dropped to 54%.

The employed plans comparison is quite strict. The plans selected by the two algorithms must be exactly the same. Such precision is often unnecessary, since in most game playing scenarios only the first action or goal of the plan is used in every move. In the next move (or in next couple of moves) the planning is performed again. Therefore, we are also interested in examining the match of only the first goals of the plans returned by the full search and by the ASAS. In this modified experiment, the results improve significantly even for the rule-based plan selection. For $k = 2$ and $k = 3$ the first goal in the plans matched in 86.7% and 85.6% cases respectively with the rule-based selection method.

V. Related Work

We focus on the environments whose complexity prohibits the application of classical game-playing algorithms based on complete state space search. Consequently, heuristic game playing and multi-agent planning techniques are predominately applied in these environments and as such they are the closest competitor to the presented approach.

The heuristic techniques used in the search often utilize background knowledge to substantially reduce the search space. Examples can be found in GO [6], where the knowledge is in a form of a hierarchical task network, in card game of bridge [7], that uses domain-specific approach, or in the goal-based game tree search (GB-GTS) algorithm [3], which uses procedural knowledge in a form of higher level goals to reduce the search. These heuristics do not tackle the complexity dependence on the number of agents and they are complementary to the presented ASAS method.

The ASAS method is similar to the work of Mock [5]. Mock developed a version of game tree search, where action choices for only one of the agents are explored at a time and all the other agents behave heuristically. Our approach extends this idea and conducts searchers for subsets of more than one agent. Consequently, in our method a more complex sub-result merging strategy needs to be applied, but quality of the produced solutions is better.

In the field of game theory, the *graphical games* (GG) [8] consider situations where each player interacts only with

a small subset of all players in the game (termed agent's *neighborhood*). The algorithms for finding the optimal strategies for graphical games can run with time complexity exponential in the size of the largest neighborhood in the game [9]. However, as we are interested in the multi-stage games, the algorithms for solving GG cannot be applied because they need a fixed structure of agents' interactions, which is unknown in advance in our case as it can possibly change with respect to different game states.

In [10] authors present an approach to multi-agent planning (MAP) that utilizes limited interactions among agents. It is based on a combination of single-agent planning and *constraint satisfaction programming*. It creates a multi-agent plan for fully cooperative agents with time complexity exponential in two sparsity measures, but not the number of agents in the environment. However, as other approaches based on MAP, the work considers only cooperative agents that work together to reach a common goal, which makes the combination of the single-agent plans possible using conflict-resolving. In adversarial situations, each agent seeks and exploits conflicts in order to get more in specific situations.

The limitation of the MAP is partially solved by Ephrati et. al. in [11], serving as an inspiration for the sub-result merging part of our method. The approach optimizes the utility and considers also the case of agents with conflicting preferences. The solution assumes that the agents agree on certain fairness criterion (e.g., egalitarian or social welfare) and that they fully cooperate to find a global common plan that optimizes it. Such approach is not usable in real-world conflict situations (e.g., military operations), because the parties involved in the conflict would not fully cooperate and they would never trust each other sufficiently.

VI. Conclusions

Interactions in multi-agent adversarial domains are crucial and cannot be simply neglected while creating plans for an agent. However in many cases, each agent interacts only with a small subset of other agents. Based on this observation, we have developed a method, termed agent subset adversarial search (ASAS), which decomposes the global adversarial search into multiple smaller overlapping sub-searches. In each sub-search only actions of a subset of all agents are searched, while the remaining agents behave heuristically based on the result from the search in the previous time-step. We proposed two different merging methods for construction of the final plan for each agent based on results from the sub-searches. The rule-based selection method, which reduces the computational complexity of the overall algorithm to polynomial in the number of agents, and the search-based method, for which the dependence on the number of the agents is still exponential, but the reduction of the search space is still significant.

The experimental evaluation proved that in spite of the reduced effort, ASAS achieves accuracy similar to the underlying full search. The subsets of three agents turned out to be a reasonable compromise between computational complexity and accuracy for our domain. With $k = 3$, the more precise search-based method produced the exact same plan as the full search in more than 90% and the faster rule-based method in 64% of cases. However, if we compare only the first actions produces by the searches (corresponding to the typical use of a game-playing algorithm), even the rule-based selection method agrees with the full search in over 85% of cases.

Future research will include a more thorough theoretical analysis of which domain properties underlie the applicability of the approach. We also plan to investigate different strategies for selecting the correct agent subsets, which would further reduce the search space. Specifically, we plan to investigate a closer integration of the principle of agents' plans independence into the construction of subsets, allowing different sizes of subsets, or possibly modifying the subsets during the sub-search process.

References

[1] C. Luckhardt and K. B. Irani, "An algorithmic solution of N-person games," in *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI)*, T. R. S. Kehler, Ed., vol. 1. Philadelphia, Pennsylvania: Morgan Kaufmann, Aug. 1986, pp. 158–162.

[2] N. Sturtevant, M. Zinkevich, and M. Bowling, "Prob-Maxˆ n: Playing N-Player Games with Opponent Models," in *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, vol. 21, no. 2. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006, p. 1057.

[3] V. Lisý, B. Bošanský, M. Jakob, and M. Pěchouček, "Adversarial search with procedural knowledge heuristic," in *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, 2009.

[4] E. Semsch, V. Lisý, B. Bošanský, M. Jakob, D. Pavlíček, J. Doubek, and M. Pěchouček, "Adversarial behavior testbed," CTU, FEE, Gerstner Lab., Technical Report GLR 90/09, 2009, http://agents.felk.cvut.cz/publications.

[5] K. J. Mock, "Hierarchical heuristic search techniques for empire-based games," in *Proceedings of the International Conference on Artificial Intelligence (IC-AI)*, 2002, pp. 643–648.

[6] S. Willmott, J. Richardson, A. Bundy, and J. Levine, "Applying adversarial planning techniques to Go," *Theoretical Computer Science*, vol. 252, no. 1–2, pp. 45–82, 2001. [Online]. Available: citeseer.ist.psu.edu/willmott01applying.html

[7] S. J. J. Smith, D. S. Nau, and T. A. Throop, "Computer bridge - a big win for AI planning," *AI Magazine*, vol. 19, no. 2, pp. 93–106, 1998. [Online]. Available: citeseer.ist.psu.edu/smith98computer.html

[8] M. Kearns, M. Littman, and S. Singh, "Graphical models for game theory," in *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*. Citeseer, 2001, pp. 253–260.

[9] C. Papadimitriou and T. Roughgarden, "Computing correlated equilibria in multi-player games," *Journal of the ACM (JACM)*, vol. 55, no. 3, p. 14, 2008.

[10] R. Brafman and C. Domshlak, "From one to many: Planning for loosely coupled multi-agent systems," in *ICAPS-08, 18th International Conference on Automated Planning and Scheduling, Sydney, Australia*, 2008.

[11] E. Ephrati and J. Rosenschein, "A heuristic technique for multi-agent planning," *Annals of Mathematics and Artificial Intelligence*, vol. 20, no. 1, pp. 13–67, 1997.