Goal-Based Game Tree Search for Complex Domains

Viliam Lisý, Branislav Bošanský, Michal Jakob, and Michal Pěchouček

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

Abstract. We present a novel approach to reducing adversarial search space by employing background knowledge represented in the form of higher-level goals that players tend to pursue in the game. The algorithm is derived from a simultaneous-move modification of the max^n algorithm by limiting the search to those branches of the game tree that are consistent with pursuing player's goals. The algorithm has been tested on a real-world-based scenario modelled as a large-scale asymmetric game. The experimental results obtained indicate the ability of the goal-based heuristic to reduce the search space to a manageable level even in complex domains while maintaining the high quality of resulting strategies.

1 Introduction

Recently, there has been a growing interest in studying complex systems, in which larger numbers of agents concurrently pursue their goals while engaging in complicated patterns of mutual interaction. Examples include real-world systems, such as various information and communication networks or social networking applications, as well as simulations, including models of societies, economies and/or warfare. Because in most such systems the agents are part of a single shared environment, situations arise in which their actions and strategies interact. Scenarios in which the outcome of agent's actions depends on actions chosen by others are often termed *games* and have been a topic of AI research from its very beginning. With the increasing complexity of environments in which the agents interact, however, classical game playing algorithms, such as minimax search, become unusable due to the huge branching factor, size of the state space, continuous time and space, and other factors.

In this paper, we present a novel game tree search algorithm adapted and extended for use in large-scale multi-player games with asymmetric objectives (non-zero-sum games). The basis of the proposed algorithm is the max^n algorithm [1] generalized to simultaneous moves. The main contribution, however, lies in a novel way in which background knowledge about possible player's goals and the conditions under which they are adopted is represented and utilized in order to reduce the extent of game tree search. The background knowledge contains:

– **Goals** corresponding to basic objectives in the game (goals represent elementary building blocks of player's strategies); each goal is associated with an algorithm which decomposes it into a sequence of actions leading to its fulfilment.

J. Filipe, A. Fred, and B. Sharp (Eds.): ICAART 2009, CCIS 67, pp. 97–109, 2010.

- **Conditions** defining world states in which pursuing the goals is meaningful (optionally, representing conditions defining when individual players might pursue the goals).
- **Evaluation functions** assigning to each player and world state a numeric value representing desirability of the game state for the player (e.g. utility of the state for the player).

The overall background knowledge utilized in the search can be split into a player-independent part (also termed *domain knowledge*) and a player-specific part (further termed *opponent models*).

The proposed approach builds on the assumption that strategies of the players in the game are composed from higher-level goals rather than from arbitrary sequences of low-level actions. Adapted in game tree search, this assumption results in considerably smaller game trees, because it allows evaluating only those sequences of low-level actions that lead to reaching some higher-level goal. However, as with almost any kind of heuristics, the reduction in computational complexity can potentially decrease the quality of resulting strategies, and this fundamental trade-off is therefore an important part of algorithm's evaluation described further in the paper.

The next section introduces the challenges that complicate using game-tree search in complex domains. Section 3 describes the proposed algorithm designed to address them. Search space reduction, precision loss and scalability of the algorithm are experimentally examined in Section 4. Section 5 reviews the related work and the paper ends with conclusions and a discussion of future research.

2 Challenges

The complex domains of our interest include real-world domains like network security or military operations. We use the later for intuition in this section. The games appearing there are often *n*-player non-zero-sum games with several conceptual problems that generally prohibit using classic game-tree search algorithms, such as max^n.

- **huge branching factor (BF)** – In contrast to many classical games, in military operations the player assign actions simultaneously to all units it controls. Together with a higher number of actions (including parameters) which the units can perform, this results in branching factors several magnitudes bigger than in games such as Chess (BF ≈ 35) or Go (BF ≈ 361).
- **importance of long plans** – In many realistic scenarios, long sequence of atomic actions is needed before a significant change to the state of world/game is produced. A standard game tree in such scenarios needs to use a correspondingly high search depth, further aggravating the effect of huge branching factor mentioned earlier.
- **simultaneous moves** – The absence of alternation of different players' moves makes the original unmodified max^n algorithm inapplicable.

Let us emphasis also the advantages that using game tree search can bring. If we successfully overcome the mentioned problems, we can reuse the large amount of research in this area and further enhance the searching algorithm with many of existing extensions (such as use of various opponent models, probabilistic extensions, transposition tables, or other shown e.g. in [2]).

3 Goal-based Game-Tree Search

In this section, we present the Goal-based Game-tree Search algorithm (denoted as GB-GTS) developed for game playing in the complex scenarios and addressing the listed challenges. We describe the problems of simultaneous moves, present our definition of goals, and then follow with a description of the algorithm and how it can be employed in a game-playing agent.

3.1 Domain

The domains supported can be formalized as a tuple $(\mathcal{P}, \mathcal{U}, \mathcal{A}, \mathcal{W}, \mathcal{T})$, where $\mathcal{P}$ is the set of players, $\mathcal{U} = \bigcup_{p \in \mathcal{P}} \mathcal{U}_p$ is a set of units/resources capable of performing actions in the world, each belonging to one of the players. $\mathcal{A} = \mathsf{X}_{u \in \mathcal{U}} \mathcal{A}_u$ is a set of combinations of actions the units can perform, $\mathcal{W}$ is the set of possible world states and $\mathcal{T} : \mathcal{W} \times \mathcal{A} \rightarrow \mathcal{W}$ is the transition function realizing one move of the game where the game world is changed via actions of all units and world's own dynamics.

The game proceeds in moves. At the beginning of a move, each player assigns actions to all units it controls (forming the action of the player). Function $\mathcal{T}$ is subsequently invoked (taking the combination of assigned actions as an input) to modify the world state.

3.2 Simultaneous Moves

There are two ways simultaneous moves can be dealt with. The first one is to directly work with joint actions of all players in each move, compute their values and consider the game matrix (normal form game) they entail. The actions of individual players can then be chosen based on a game-theoretical equilibrium (e.g. Nash equilibrium in [3]). The second option is to fix the order of the players and let them choose their actions separately in the same way as in max^n, but using the unchanged world state from the end of the previous move for all of them and with the action execution delayed until all players have chosen their actions. This method is called *delayed execution* in [4]. In our experiments, we have used the approach with fixed player order, because of its easier implementation, allowing us to focus on core issue of utilizing the background knowledge.

3.3 Goals

For our algorithm, we define a goal as a pair (I_g, A_g), where $I_g(\mathcal{W}, \mathcal{U})$ is the initiation condition of the goal and A_g is an algorithm that, depending on its internal state and the current state of the world, *deterministically* outputs the next action that leads to fulfilling the goal.

A goal can be assigned to one unit and it is then pursued until it is successfully reached or dropped because its pursuit is no longer practical. Note that we do not specify any dropping or succeeding condition, as they are implicitly captured in the A_g algorithm. We allow the goal to be abandoned only if A_g is finished; furthermore, each unit can pursue only one goal at a time. There are no restrictions on the form of

algorithm A_g, so it can represent any type of goal from the taxonomy of goals presented in [5] (e.g. maintain, achieve) and any kind of architecture (e.g. BDI, HTN) can be used to describe it.

The goals in GB-GTS serve as building blocks for more complex strategies that are created by combining different goals for different units and then explored via search. This contrasts with HTN-based approaches used for guiding the game tree search (see Section 5), where the whole strategies are encoded using decompositions from the highest levels of abstraction to the lower ones.

3.4 Algorithm Description

The main procedure of the algorithm (outlined in Figure 1 as procedure *GBSearch()*) recursively computes the value of a state for each of the players assuming that all the units will rationally optimize the utility of the players controlling them. The inputs to the procedure are the world state for which the value is to be computed, the depth to which to search from the world state and the goals the units are currently pursuing. The last parameter is empty when the function is called for the first time.

The algorithm is composed of two parts. The first is the simulation of the world changes based on the world dynamics and the goals that are assigned and pursued by the units, and the second is branching on all possible goals that a unit can pursue after it is finished with its previous goal.

The first part – *simulation* – consists of lines 1 to 15. If all units have a goal they actively pursue, the activity in the world is simulated without any need for branching. The simulation runs in moves and lines 3-10 describe the simulation of a single move. At first, for each unit, an action is generated based on the goal g assigned to this unit (line 5). If the goal-related algorithm A_g has finished, the goal is removed from the map of goals (lines 6-8) and the unit that was previously assigned to this goal becomes idle. The generated actions are then executed and the conflicting changes of the world are resolved in accordance with the game rules (line 10). After this step, one move of the simulation is finished. If the simulation has reached the required depth of search, the resulting state of the world is evaluated using the evaluation functions of all players (line 13).

The second part of the algorithm – *branching* – starts when the simulation reaches the point where at least one unit has finished pursuing its goal (lines 16-22). In order to ensure fixed order of players (see Section 3.2), the next processed unit is chosen from the idle units based on the ordering of the players that control the units (line 16). In the run of the algorithm, all idle units of one player are considered before moving to the units of the next one. The rest of the procedure deals with the selected unit. For this unit, the algorithm sequentially assigns each of the goals that are applicable for the unit in the current situation. The applicability is given by the goal's I_g condition. For each applicable goal, the algorithm assigns the goal to the unit and evaluates the value of the assignment by recursively calling the whole *GBSearch()* procedure (line 19). The current goals of the units are cloned because the state of the already started algorithms generating actions from goals for the rest of the units (A_g) must be the same for all the considered goals of the selected unit. After computing the value of each goal assignment, the one that maximizes the utility of the owner of the unit is chosen (line 21) and the values of this decision for all players are returned by the procedure.

Input: $W \in \mathcal{W}$: current world state, d: search depth, $G[\mathcal{U}]$: map from units to goals they pursue

Output: an array of values of the world state (one value for each player)

```
1  curW = W
2  while all units have goals in G do
3  |  Actions = ∅
4  |  foreach goal g in G do
5  |  |  Actions = Actions ∪ NextAction(A_g)
6  |  |  if A_g is finished then
7  |  |  |  remove g from G
8  |  |  end
9  |  end
10 |  curW = T(curW,Actions)
11 |  d = d − 1
12 |  if d=0 then
13 |  |  return Evaluate(curW)
14 |  end
15 end
16 u = GetFirstUnitWithoutGoal(G)
17 foreach goal g with satisfied I_g(curW, u) do
18 |  G[u] = g
19 |  V[g] = GBSearch(curW, d,Copy(G))
20 end
21 g = arg max_g V[g][Owner(u)]
22 return V[g]
```

Fig. 1. GBSearch(W, d, G) – the main procedure of GB-GTS algorithm

3.5 Game Playing

The pseudo-code on Figure 1 shows only the computation of the values of the decisions; it does not deal with how the algorithm can be employed by a player to determine its next actions in the game. In order to do so, the player needs to extract a set of goals for its units from the searched game tree.

Each node in the search procedure execution tree is associated with a unit – the unit for which the goals are tried out. During the run of the algorithm, we store the maximizing goal choices from the top of the search tree representing the first move of the game. The stored goals for each idle unit of the searching player are the main output of the search.

In general, there are two ways the proposed goal-based search algorithm can be used in game-playing.

In the first approach, the algorithm in started in each move and with all units in the simulation set to idle. The resulting goals are extracted and the first actions generated for each of the goals are played in the game. Such an *eager* approach is better for coping with unexpected events should also be more robust in case the background knowledge does not exactly describe the activities in the game.

In the second approach the player using the algorithm maintains a list of current goals for all units it controls. If none of its units is idle (i.e. has no goal assigned), the player uses the goals to generate next actions for its units. Otherwise, the search algorithm is started with goals for the player's non-idle units pre-set and all the other units idle. The goals generated for the previously idle are assigned and pursued in following moves. This *lazy* approach is significantly less computationally intensive.

3.6 Opponent Models

The search algorithm introduced in this section is very suitable for the use of opponent models, that are proved to be useful in adversarial search in [6]. There are two types of opponent models in our approach. We now describe how they can be utilized in the algorithm. The first type, the evaluation function capturing preferences of each player is already an essential part of the max^n algorithm.

The other type of the opponent model can be used to reduce the set of all applicable goals (iterated in Figure 1 on lines 18-21) to the goals a particular player is *likely* to pursue. This can be done by adding player-specific constrains to conditions I_g defining when the respective goal is applicable. These constrains can be hand-coded by an expert or learned from experience; we call them *goal-restricting opponent models*.

The role of goal-restricting opponent model can be illustrated on a simple example of the goal representing loading a commodity to a truck. The domain restriction I_g could be that the truck must not be full. The additional player-specific constraint could be that the commodity must be produced locally at the location, because the particular opponent never uses temporary storage locations for the commodity and always transports it from the place where it is produced to the place where it is consumed.

Using a suitable goal-restricting opponent model can further reduce the size of the space that needs to be searched by the algorithm. A similar way of pruning is possible also in adversarial search without goals. We believe, however, that determining which goal a player will pursue in a given situation is more intuitive and easier to learn than to determine which low-level atomic action (e.g. going right or left on a crossroad) a player will execute.

4 Experiments

In order to practically examine the proposed goal-based (GB) adversarial search algorithm, we performed several experiments. Firstly, we compare it to the exhaustive search of the complete game tree performed by the simultaneous-move modification of max^n search (further called action-based (AB) search) in order to assess the ability to reduce the volume of search on one hand and to maintain the quality of resulting strategies on the other. Afterwards, we analyze the scalability of the GB algorithm in more complex scenarios.

Note that we use the eager, computationally more intensive version of the game playing algorithm in the experiments (see Section 3.5), in which all units are choosing their goals at the same moment in each move.

4.1 Example Game

The game we use as a test case is modelled after a humanitarian relief operation in an unstable environment, with three players - government, humanitarian organization, and separatists. Each of the players controls a number of units with different capabilities that are placed in the game world represented by a graph. Any number of units can be located in each vertex of the graph and 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 players use to construct buildings and produce other commodities.

The utilities (evaluation functions) representing the main objectives of the players 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. The government control is derived from the state of the infrastructure, the difference between the number of units of individual players in the city and the state of the control of the city in the previous move. Detailed description of the game can be found in [7].

The goals, used in the algorithm, are generated by instantiation of fifteen goal types. Each goal type is represented as a Java class. Only four of the fifteen classes are unique and the rest nine classes are derived from four generic classes in a very simple way. The actions leading towards achieving a goal consist typically of path-finding to a specific vertex, waiting for a condition to hold, performing a specific action (e.g. loading/unloading commodities), or their concatenation. The most complex goal is escorting a truck by cop that consists of estimating a proper meeting point, path planning to that point, waiting for the truck, and accompanying it to its destination.

Simple Scenario In order to run the standard AB algorithm on a game of this complexity, the scenario has to be scaled down to a quite simple problem. We have created a simplified scenario as a subset of our game with the following main characteristics (see also the scheme in Figure 2):

- only two cities can be controlled (Vertices 3 and 6)
- a government's HQ is built in Vertex 3
- two "main" units - police (cop) and gangster (gng) are placed in Vertex 3
- a truck is transporting explosives from Vertex 5 to Vertex 7
- another two trucks are transporting food from Vertex 1 to the city with food shortage Vertex 3

There are several possible runs of this scenario. The police unit has to protect several possible threats. In order to make government to lose control in Vertex 3, the gangster can either destroy food from a truck to cause starving resulting in lowering the wellbeing and consequent destroying of the HQ (by riots), or it can steal explosives and build a suicide bomber that will destroy the HQ without reducing wellbeing in the city. Finally, it can also try to gain control in city in Vertex 6 just by outnumbering the police there. In order to explore all these options, the search depth necessary is six moves.

Even such a small scenario creates a game tree too big for the AB algorithm. Five units with around four applicable actions each (depending on the state of the world) considered in six consequent moves results in $(4^5)^6 \approx 10^{18}$ world states to examine. Hence, we further simplify it for the AB algorithm. Only the actions of two units (cop

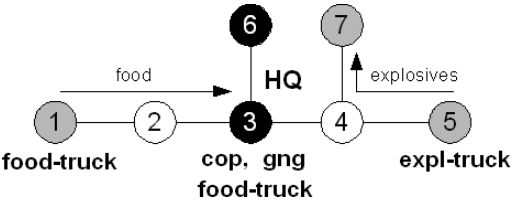

Fig. 2. A schema of the simple scenario. Black vertices represent cities that can be controlled by players, grey vertices represent cities that cannot be controlled, and white vertices do not contain cities.

and gng) are actually explored in the AB search and the actions of the trucks are considered to be a part of the environment (i.e. the trucks are scripted to act rationally in this scenario). Note that the GB algorithm does not need this simplification and actions of all units are explored in the GB algorithm.

4.2 Search Reduction

Using this simplified scenario, we first analyze how the main objective of the algorithm – *search space reduction* – is satisfied.

We run the GB and AB algorithms on a fixed set of 450 problems – world states samples extracted from 30 different traces of the game. On each configuration, we experiment with different values of the look-ahead parameter (1-6 for the AB algorithm and 1-19 for the GB algorithm). As we can see in Figure 3(a), the experimental results fulfilled our aim of substantial reduction of the search space. The number of nodes explored increases exponentially with the depth of the search. However, the base of the exponential is much lower for the GB algorithm. The size of the AB tree for six moves look-ahead is over 27 million, while GB search with the same look-ahead explores only

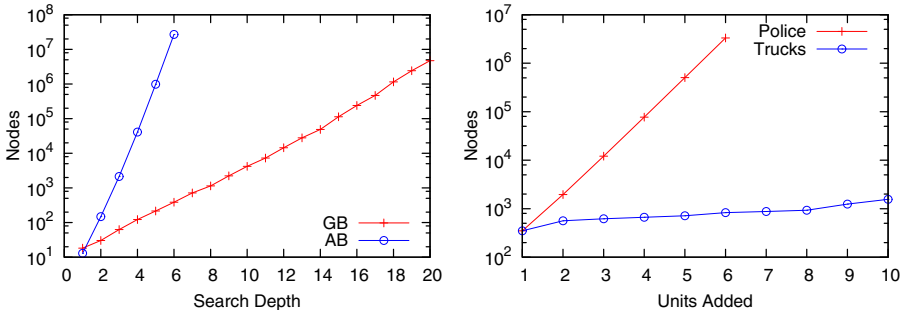

Fig. 3. (a) The search space reduction of the GB algorithm compared to the AB algorithm. An average number of the search tree nodes explored depending on the search depth is shown for both algorithms in a logarithmic scale. (b) Increase of the size of the searched tree when adding one to ten police units and explosives trucks to the simple scenario with a 6 move look-ahead.

385 nodes and even for the look-ahead of nineteen, the size of the tree was in average less than 5×10^6. These numbers indicate that using heuristic background knowledge can reduce the time needed to choose an action in the game from tens of minutes to a fraction of a second.

Our implementations of each of the algorithms processed approximately twenty five thousands nodes per second on our test hardware without any optimization techniques. According to [8], however, game trees with million nodes can be searched in real-time (about one second) when such optimization is applied and when efficient data structures are used.

4.3 Loss of Accuracy

With such substantial reduction of the set of possible courses of action explored in the game, some loss of quality of game-playing can be expected. Using the simplified scenario, we compared the actions resulting from the AB and the first action generated by the goal resulting from the GB algorithm. The action differed in 47% of cases. However, a different action does not necessarily mean that the GB search has found a sub-optimal move. Two different actions often have the same value in AB search. Because of the possibly different order in which actions are considered, the GB algorithm can output an action which is different from the AB output yet still has the same optimal value. The values of actions referred to in the next paragraph all come from the AB algorithm.

The value of the action resulting from the GB algorithm was in **88.1%** of cases exactly the same as the "optimal" value resulting from AB algorithm. If the action chosen by GB algorithm was different, it was still often close to the optimal value. We were measuring the difference between the values of GB and optimal actions, relative to the difference between the maximal and the minimal value resulting from the searching player's decisions in the first move in the AB search. The mean relative loss of the GB algorithm was 9.4% of the range. In some cases, the GB algorithm has chosen the action with minimal value, but it was only in situations, where the absolute difference between the utilities of the options was small.

4.4 Scalability

Previous sections show that the GB algorithm can be much faster than and almost as accurate as the AB algorithm with suitable goals. We continue with assessing the limits on the complexity of the scenario where GB algorithm is still usable. There are several possible expansions of the simple scenario. We explore the most relevant factor – number of units – separately and then we apply the GB algorithm on a bigger scenario. In all experiments, we ran the GB algorithm in the initial position of the extended simple scenario and we measured the size of the searched part of the game tree.

Adding Units. The increase of the size of the searched tree naturally depends on the average number of goals applicable for a unit when it becomes idle and the lengths of the plans that lead to their fulfilment. The explosives truck has usually only a couple of applicable goals. If it is empty, the goal is to load in one of the few cities where

explosives are produced and if it is full, the goal is to unload somewhere where explosives can be consumed. On the other hand, a police unit has many possible goals. It can protect any transport from being robbed or it can try to outnumber the separatists in any city. We were adding these two unit types to the simple scenario and computed the size of the search tree with fixed six moves look-ahead.

When adding one to ten explosives trucks to the simple scenario, each of them has always only one goal to pursue at any moment. Due to our GB algorithm definition, where goals for each unit are evaluated in different search tree node, even adding a unit with only one possible goal increases the number of evaluated nodes slightly. In Figure 3(b) are the results for this experiment depicted as circles. The number of the evaluated nodes increases only linearly with increasing the number of the trucks.

Adding further police units with four goals each to the simple scenario increased the tree size exponentially. The results for this experiment are shown in Figure 3(b) as pluses.

Complex Scenario. In order to test the usability of the GB search in a more realistic setting, we implemented a larger scenario within our test domain. We used a graph with 2574 vertices and two sets of units. The first unit set was composed of nine units, including two police units with up to four possible goals in one moment, two gangster units with up to four possible goals, an engineer with three goals, stone truck with up to two goals and three trucks with only one commodity source and one meaningful destination resulting to one goal at any moment. The second unit set included seven units – one police, one gangster unit and the same amount of units of the other types. The lengths of the plans to reach these goals is approximately seven atomic actions. There are five cities, where the game is played.

A major difference of this scenario to the simple scenario is, besides the added units, a much bigger game location graph and hence higher length of the routes between cities. As a result, all plans of the units that need to arrive to a city and perform some actions there are proportionally prolonged. This is not a problem for the GB algorithm, because the move actions along the route are only simulated in the simulation phase and do not cause any branching.

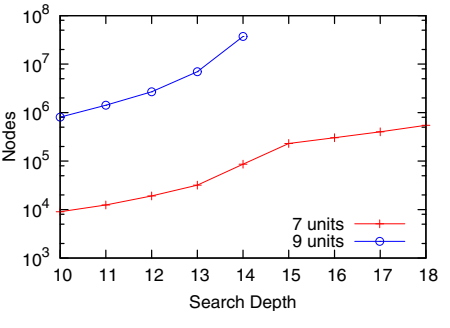

Fig. 4. Size of the trees searched by the GB algorithm in the complex scenario

In a simple experiment to prove this, we changed the time scale of the simulation, so that all the actions were split to two sub-actions collectively effecting the same change of the game world. After this modification, the GB algorithm explored exactly the same number of nodes and the time needed for the computation increased linearly, corresponding to more simulation steps needed.

If we assume that an optimized version of the algorithm can compute one million nodes in a reasonable time, then the look-ahead we can use in the complex scenario is 10 in the nine-unit case and 18 in the seven-unit case. Both values are higher than the average length of unit's typical plan, allowing it to find meaningful plans. If we wanted to apply the AB algorithm to the seven-unit case with the look-ahead of eighteen, considering only four possible move directions and waiting for each unit, it would mean searching through approximately $4^{7^{18}} \approx 10^{75}$ nodes of the game tree, clearly an impossible task.

5 Related Work

The idea to use domain knowledge in order to reduce the portion of the game tree that is searched during the play has already appeared in literature. The best-known example is probably [9], where authors used HTN formalism to define the set of runs of the game, that are consistent with some predefined hand-coded strategies. During game playing, they search only that part of the game tree.

A plan library represented as HTN is used to play GO in [10]. The searching player simulates HTN planning for both the players, without considering what the other one is trying to achieve. If one player achieves its goal, the opponent backtracks (the shared state of the world is returned to the previous state) and tries a different decomposition.

Both these works use quite detailed descriptions of the whole space of the meaningful strategies in HTN. Another approach for reducing the portion of the tree that is searched for scenarios with multiple units is introduced in [11]. The authors show successful experiments with searching just for one unit at a time, while simulating the movements of the other units using rule-based heuristic.

An alternative to using the game-tree search is summarized in [12]. They solve large scale problems with multiple units using, besides other methods, generation of the meaningful sequences of unit actions pruned according to various criteria. One of them is whether they can be intercepted (rendered useless or counterproductive) by traces of the opponent's units.

6 Conclusions

We have proposed a novel approach to introducing background knowledge heuristic to multi-player simultaneous-move adversarial search. The approach is particularly useful in domains where *long sequences* of actions are required to produce significant changes in the world state, each unit (or other resource) can only pursue a *few goals* at any time, and the decomposition of a each goal into low level actions is *uniquely defined* (e.g. using the shortest path to move between two locations).

We have compared the performance of the algorithm to a slightly modified exhaustive max^n search, showing that despite examining only a small fraction of the game tree (less than 0.002% for the look-ahead of six game moves), the goal-based search is still able to find the optimal solution in 88.1% cases; furthermore, even the suboptimal solutions produced are often close to the optimum. These results have been obtained with the background knowledge designed *before* implementing and evaluating the algorithm and without any further optimization to prevent over-fitting.

Furthermore, we have tested the scalability of the algorithm to larger scenarios where the modified max^n search cannot be applied at all. We have confirmed that although the algorithm's time complexity cannot escape exponential growth, this growth can be controlled by reducing the number of different goals considered for each unit and by making the action sequences generated by goals longer. Simulations on a real-world scenario modelled as a multi-player asymmetric game proved the approach viable, though further optimization and improved background knowledge would be necessary for the algorithm to discover more complex strategies.

An important advantage of the proposed approach is its compatibility with all existing extensions of general-sum game-tree search based on modified value back-up procedure and other optimizations. It is also insensitive to the granularity of space and time with which the game is modelled as long as the structure of the goals remains the same and their decomposition into low-level actions is scaled correspondingly.

In the future, we aim to implement additional technical improvements in order to make the goal-based search applicable to even larger problems. In addition, we would like to address the problem of the automatic extraction of goal-based background knowledge from game histories. First, we plan to learn goal initiation conditions for individual players and use them for additional search space pruning. Later, we plan to address a more challenging problem of learning the goal decomposition algorithms.

Acknowledgements. Effort sponsored by the Air Force Office of Scientific Research, USAF, under grant number FA8655-07-1-3083 and by the Research Programme No. MSM6840770038 by the Ministry of Education of the Czech Republic. The U.S. Government is authorized to reproduce and distribute reprints for Government purpose notwithstanding any copyright notation thereon.

References

1. Luckhardt, C., Irani, K.B.: An algorithmic solution of n-person games. In: Proc. of the National Conference on Artificial Intelligence (AAAI 1986), Philadelphia, Pa, August 1986, pp. 158–162 (1986)
2. Schaeffer, J.: The history heuristic and alpha-beta search enhancements in practice. IEEE Transactions on Pattern Analysis and Machine Intelligence 11(11), 1203–1212 (1989)
3. Sailer, F., Buro, M., Lanctot, M.: Adversarial planning through strategy simulation. In: IEEE Symposium on Computational Intelligence and Games (CIG), Honolulu, pp. 80–87 (2007)
4. Kovarsky, A., Buro, M.: Heuristic search applied to abstract combat games. In: Canadian Conference on AI, pp. 66–78 (2005)

5. van Riemsdijk, M.B., Dastani, M., Winikoff, M.: Goals in agent systems: A unifying framework. In: Padgham, Parkes, Müller, Parsons (eds.) Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, May 2008, pp. 713–720 (2008)
6. Carmel, D., Markovitch, S.: Learning and using opponent models in adversary search. Technical Report CIS9609, Technion (1996)
7. Semsch, E., Lisý, V., Bošanský, B., Jakob, M., Pavlíček, D., Doubek, J., Pechoucek, M.: Adversarial behavior testbed. Technical Report GLR 90/09, CTU, FEE, Gerstner Lab (2009), http://agents.felk.cvut.cz/publications
8. Billings, D., Davidson, A., Schauenberg, T., Burch, N., Bowling, M., Holte, R.C., Schaeffer, J., Szafron, D.: Game-tree search with adaptation in stochastic imperfect-information games. In: van den Herik, H.J., Björnsson, Y., Netanyahu, N.S. (eds.) CG 2004. LNCS, vol. 3846, pp. 21–34. Springer, Heidelberg (2006)
9. Smith, S.J.J., Nau, D.S., Throop, T.A.: Computer bridge - a big win for AI planning. AI Magazine 19(2), 93–106 (1998)
10. Willmott, S., Richardson, J., Bundy, A., Levine, J.: Applying adversarial planning techniques to Go. Theoretical Computer Science 252(1-2), 45–82 (2001)
11. Mock, K.J.: Hierarchical heuristic search techniques for empire-based games. In: ICAI, pp. 643–648 (2002)
12. Stilman, B., Yakhnis, V., Umanskiy, O.: 3.3. Strategies in Large Scale Problems. In: Adversarial Reasoning: Computational Approaches to Reading the Opponent's Mind, pp. 251–285. Chapman & Hall/CRC, Boca Raton (2007)