A New Approach to Security Games

Jan Karwowski(✉) and Jacek Mańdziuk

Faculty of Mathematics and Information Science, Warsaw University of Technology,
Koszykowa 75, 00-662 Warsaw, Poland
{jan.karwowski,j.mandziuk}@mini.pw.edu.pl

Abstract. The paper proposes a new approach to finding Defender's strategy in Security Games. The method is based on a modification to the Upper Confidence bound applied to Trees (UCT) algorithm that allows to address the imperfect information games. The key advantage of our approach compared to the ones proposed hitherto in the literature lies in high flexibility of our method which can be applied, without any modifications, to a large variety of security games models. Furthermore, due to simulation-based nature of the proposed solution, various Attacker's profiles (e.g. non-completely rational behavior) can be easily tested, as opposed to the methods rooted in the game-theoretic framework.

Keywords: Machine learning · UCT · Security games · Imperfect information · Bounded rationality

1 Introduction

Terrorism threats in 21st century raised many concerns in security of crowded public places like airports or trains stations. Governments are spending large amounts of financial resources on creating better technology for scanning, surveillance, weapon production as well as on increasing number of security officers. Even vast amount of all those resources is still insufficient to protect all vulnerable targets all the time. Hence, tactical planning remains a crucial aspect of national security and one way of its addressing is by means of Security Games (SG). The goal of SG is to provide optimal Defender strategy for protecting several targets. In such games it is usually assumed that Defender's resources are insufficient to protect all targets all the time (e.g. the number of security patrols is lower than the number of buildings to protect).

Many real life security problems are modeled via SG. The list of problems includes: securing airports and plane flights [4,8], US Coast Guard patrol planning [11], protecting forests from illegal cut [5] or protecting ferries on sea [3]. The first two works were deployed into real life and are used in LAX airport and Boston Coast, resp. The authors of the above listed papers use Stackelberg Game model to express SG problem. This is a simple non-zero-sum finite matrix game. Defender's moves represent targets to be protected and Attacker's moves represent the targets to be attacked. The game lasts only one round and the Defender seeks Stackelberg Equilibrium [12] to define their optimal strategy.

L. Rutkowski et al. (Eds.): ICAISC 2015, Part II, LNAI 9120, pp. 402–411, 2015.
DOI: 10.1007/978-3-319-19369-4_36

There are three main assumptions made with respect to the game model: (1) all rounds are independent (e.g. Defender's moves do not affect Attacker's tactics); (2) Attacker knows Defender's strategy; (3) Attacker makes perfectly rational decisions.

Moreover finding Stackelberg Equilibrium requires solving Mixed Integer Linear Programming (MILP) problem, which is in general NP-hard. All presented papers propose polynomial time solution for some subset of the problem or approximate solution.

Another, more flexible, model used for SG is Pursuer-Evader game of a graph. We are given a graph, where some vertices are targets. Attacker and Defender move their units between vertices connected with an edge. Attackers goal is to reach target vertex and not get caught. Defender's goal is to prevent Attacker from reaching the targets. When the Defender and the Attacker are in the same vertex at the same time, the Attacker is caught and loses the game. The game is divided into discrete time steps. In each step players can move each of their units to any vertex that is adjacent to the current unit's position. This game model is considered in the paper.

The remainder of the paper is structured as follows: Section 2 provides a detailed description of a game model and Section 3 gives a brief overview of the UCT method, which is the main SG playing engine proposed in this paper. Experimental setup, results and conclusions are presented in the last two sections.

2 Game Definition

Two players, Defender and Attacker, are taking part in the game. Let $G = (V, E)$ be a directed simple graph. V is a vertex set and E is an edge (arc) set. Let $T \subset V$ be a set of targets (resources that the Defender has to protect), $S \subset V$ be a set of spawns (vertices where the Attacker can enter the graph). Let $b \in V$ be a base (vertex where Defender units are in the first round). Let D be set of Defender units (patrols). Defender has a fixed number of units, no Defender's unit can be added or removed during the game. Let A_σ be a set of Attacker units being present in given a game state σ. The number of Attacker's units changes during the game as Attacker can introduce new unit in each round as well as some units can get caught. Game is divided into rounds (time steps). In each round players make decisions about new positions of their units in the next round.

2.1 Game State

Current positions of all units in game optionally accompanied by some variant-specific information form a game state. Let Σ be a set of all possible game states. We will usually use $\sigma \in \Sigma$ to denote some game state. Besides A_σ introduced earlier, each game state has information about units' positions. Let $P_\sigma : A_\sigma \cup D \to V$ be a function describing positions.

In the initial game state σ_0, there are no Attacker units and all Defender's are in the base: $A_{\sigma_0} = \emptyset$, $(\forall d \in D) P_{\sigma_0}(d) = b$.

The game does not provide perfect information to the players. Defender knows only his own units' positions. Attacker knows his units' positions and can see Defender units if they are in the targets.

2.2 Movement

A Defender move consists of decisions on each Defender unit position in the next round. A new unit's positions must be either the same vertex (unit is not moved) or be a vertex adjacent to the current position.

Attacker moves are more complex. Besides making analogous move decision to all Attacker units in the game, Attacker can also introduce a new unit, which will be available since the beginning of the next round. If Attacker decides to introduce a unit he must also set its initial position s, choosing it from the spawns set: $s \in S$.

The players make their decisions simultaneously, not knowing the opponent's plans. When decisions are made, units are moved and new positions are checked for *Attack and Defense Situations*. A **Defense Situation** is when at least one Attacker and one Defender unit are in the same vertex. In such a case all Attacker units in this vertex are caught (removed from a game) and players are given appropriate pay-offs. An **Attack Situation** is when Attacker is in a target vertex $t \in T$ (and no Defender's units are present in that vertex). This Situation is considered a successful attack - players are given appropriate pay-offs and Attacker unit is removed from the game.

The order of checking Situations is important. Defense Situations are checked in the first place (so Defender can catch Attacker in a target). After checking and possibly removing some Attackers a new state with units' positions is set. The transition is discrete and no intermediate positions on edges are considered. In particular when a, b are adjacent vertices, Defender moves from a to b, and Attacker moves from b to a, they will not meet and Attacker will not be caught.

2.3 Pay-Offs

At the end of each round after all requested units have been moved and all Defense and Attacks situations have been evaluated each player is given some pay-off according to encountered Situations. For each Attacker Situation, Attacker is rewarded according to $R_A : T \to \mathbb{Z}^+$ reward function and Defender gets penalty $P_D : T \to \mathbb{Z}^-$. In case of Defense Situations we have similarly: $R_D : V \to \mathbb{Z}^+$ to describe Defender's reward and $P_A : V \to \mathbb{Z}^-$ for Attacker's penalty. Each round pay-off is a sum of rewards and penalties for all Situations that happened during that round. The reward/penalty functions are game parameters.

3 UCT Applied to Security Games

Upper Confidence Bounds (UCB) was introduced in [2] to optimize a long-term pay-off from playing a number of Single-Armed Bandit (Slot) Machines in a

casino. It is assumed that each machine has some unknown stationary probability distribution of pay-offs. UCB starts with random sampling the machines. Once some initial information is gathered the algorithm tries arms with higher average pay-off more frequently while still sampling the rest of arms from time to time. The algorithm was proven to converge to the optimal playing policy (maximal expected pay-off in the long run) [2].

Upper Confidence Bounds applied to Trees (UCT) [6], is an extension of UCB approach applied to games, whose states are represented in the form of a (game) tree. In the current game state (root of a tree) UCT performs massive simulations traversing the game tree until the leaf nodes. In each such simulation (playout), in each game state, the algorithm performs, similar to UCB, sampling of the child nodes, chooses one of them, moves to this node, etc. Once the simulation is completed (reaches the terminal state) the game outcome is propagated back from that node all the way to the root node, updating the quality statistics of all nodes on a path that was traversed in this playout. After finalizing the simulation phase, a move with the highest average reward (game outcome) is selected in the actual play (in our case the SG). A pseudocode of the UCT method is presented in Algorithm 1. It is worth mentioning that UCT converges to min-max strategy when the number of samples increases to infinity [6]. For the sake of space limits we are not able to go into more details. For a detailed description of the UCT method please consult, for instance, [6,13,15].

UCT method was successfully applied to various problems, in particular: Go game [17], combinatorial optimization expressed as a game [9,10], General Game Playing [13,14,15], project scheduling [16], dynamic vehicle routing [7] or problems which involve MILP [1].

3.1 Imperfect Information Games

The UCT algorithm cannot be straightforwardly applied to imperfect information games. The problem lies in line 4 of the Algorithm 1 where multiple simulations from a given state are performed. In imperfect information games, this state

Algorithm 1. The UCT algorithm

```
1 State ← InitialState // Starting state of game
2 while not IsTerminal(State) do
      // Game is not over
3     for i ← 1...simCount do
          // Number of times each node on path is evaluated - alogrithm
             parameter
4         SingleRun(State) // Performs a game simulation from given state
             and updates payoff statistics for all moves after State
5     Best ← UCTBestMove(State) // Choose best move from State
         according to pay-off statistics gathered in simulations
6     MakeMove(State, Best)
```

is not fully observable (only the Defender's units' locations are available - for Defender). The information about Attacker's positions constitutes an unknown context of a given state (UCT tree node). Therefore we propose a modification of Algorithm 1, which is presented in Algorithm 2. The baseline idea is to evaluate the n-th move in a game which may lead to different game states when invisible information is included. In addition the modified algorithm introduces the move limit of SG (variable *maxDepth*).

Algorithm 2. UCT variant for SG (imperfect information games)

```
1  for depth ← 1 ··· maxDepth do
       // Limit depth (number of rounds) of evaluation (game is now
          finite)
2      for i ← 1 ... simCount do
           // simCount - the same parameter as in Algorithm 1
3          State ← InitialState
4          for d ← 1 ··· depth − 1 do
               // Play given number of steps, so we will be evaluating
                  depth-th move in game
5              Best ← UCTBestMove(State)
6              MakeMove(State, Best)
7          SingleRun(State)// State may be different at each simulation
              now
```

3.2 Game Model Implementation

Two variants of states and move modeling were proposed and compared in this paper. Both of them implement a model defined in Section 2 and strategy defined on one of them can be transformed the equivalent strategy on the other one.

In the first variant, the game state is encoded as a vector of length $k = |D|$, where each element describes state of one Defender's unit. The state maintains one of two kinds of values: $\bullet v$ – the unit is in vertex v, $\rightarrow v$ – the unit was requested to go to v, i.e. it was sent to v in one of the previous moves and is on his way to v (information about current position is discarded). In a given round Defender can move any number of his $\bullet u$ units to any of their adjacent nodes (reachable by a single directed edge of a graph). All units $\rightarrow v$ move one step forward towards their destination vertex.

The second variant implements a more straightforward idea. Defender's state is represented as a numerical vector in which the i-th element denotes the vertex number of the current location of the i-th unit. A Defender's move is a vector of the requested unit's positions in the next round, resp. Again, the requested position of a unit can only be either its current position (no movement) or a vertex adjacent to this position.

The game ends after a pre-defined number of rounds.

3.3 Attacker Implementation

UCT simulation phase requires simulating an opponent (Attacker) to play against. In our implementation the opponent choose target to attack based on the expected pay-off in that vertex.

Please recall, that due to a game definition Attacker can see positions of Defender's units when they are located in any of the targets. The Attacker player, therefore, gathers statistics of Defender visits in each target and calculates the expected pay-off for each target vertex using formula (1), where: c_v is the number of times vertex v was covered by Defender, t is the number of rounds in which the statistics were collected.

$$EP(v) = \frac{c_v P_A(v) + (t - c_v) R_A(v)}{t} \tag{1}$$

Furthermore, Attacker decides to start an attack with probability 0.2 (see Algorithm 3).

Algorithm 3. Single Attacker move

```
if UniformRandom(0, 1) < 0.2 then
    // decided to start an attack
    Start ← UniformRandom(S)                    // Random starting vertex
    Destination ← Roulette(T, EP)        // Destination from distribution
    where probability is proportional to value of EP function
    ScheduleAttack(Start,Destination)// This function puts new Attacker
       unit on Start vertex and makes that during consequent rounds
       it moves to Destination using the shortest path
```

4 Experimental Design

The algorithm was tested on a few different games with several parameter sets. For each pair $(game, parameterset)$21 repetitions of the training process followed by a game playing phase were performed. In the game playing phase payoffs and Situation counts from 40 games each of length 10 000 rounds were used. The UCT configuration used in the experiments is presented in Table 1.

4.1 Test Games

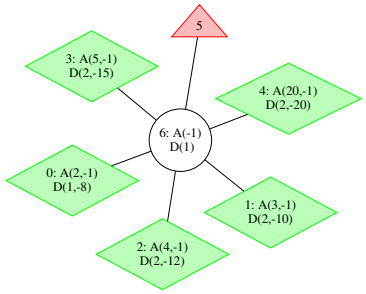
For each of the graphs presented in Graphs 1.1,1.2,1.3 one game was defined. In all test games a winning strategy that allows protection of all the targets all the time exists for the Defender. In all graphs, the targets (elements of T) are denoted as green diamonds, and spawns (elements of S) as red triangles.

The results of experiments are presented in Table 2 as average values of 21 experiment runs. All experiments were computed on Intel Core i7-2600 3.4GHz CPU. A single experiment lasted approximately 14 minutes regardless of the

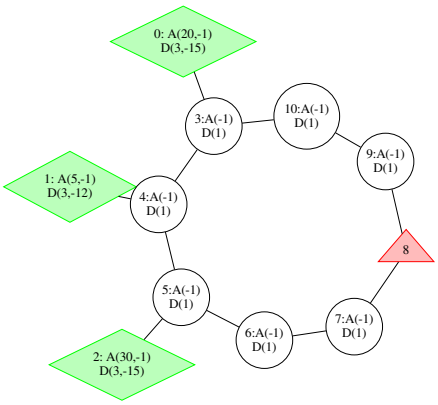
Table 1. UCT parameters for the experiments

Parameter	Value
Depth of full evaluation ($maxDepth$ in Algorithm 2)	20
Number of evaluation of each level ($simCount$)	6000
Number of rounds after which each simulation is stopped	20
Expansion coefficient (C in UCT formula)	See Table 2
Number of rounds each pay-off is back-propagated	See Table 2
Repetitions of the whole training phase before evaluation	20

Graph 1.1. A graph with single central vertex to protect. In a game the Defender had one unit at his disposal. The optimal strategy is to stay in node 6.

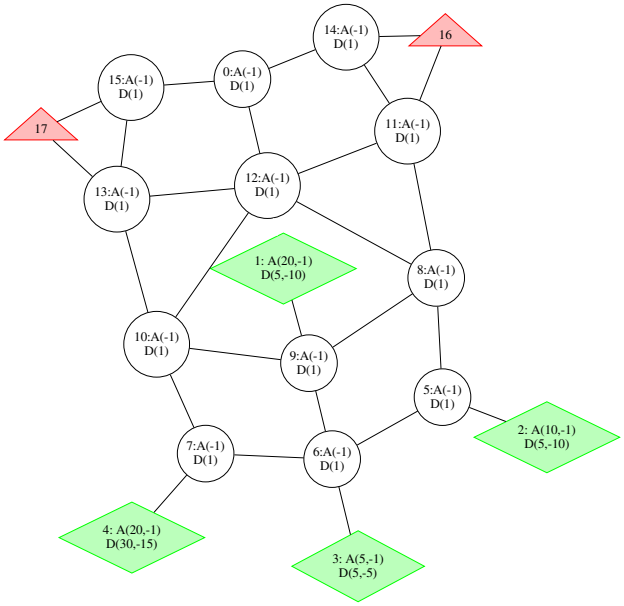


Graph 1.2. Graph based on cycle with two path to protect. In a game the Defender had 2 units at his disposal. The optimal strategy is to patrol the section of nodes $3, 9, 10$ (one unit) and the one of the nodes $5, 6, 7$ (the other unit).



game being played and the configuration set used. Please note the restriction imposed in the length of he path along which the results of simulations are back-propagated. Such a limited, local impact of simulation result introduces a significant difference compared to "typical" UCT implementation, in which the result would be back-propagated along the whole path, however, our preliminary tests proved the advantage of proposed implementation over the "regular" one.

Graph 1.3. More complex graph with two vertices to protect. In a game the Defender had 2 units at his disposal. Please note that in this game the timing is crucial since Defender's units (located in node 0) and Attacker's units (located in nodes 16 and 17) have the same distance to the targets.



5 Results and Discussion

On a general note, the results show that our modified-UCT algorithm was able to repeatedly find a winning strategy for all games tested. In many of the cases the Attacker did not make even a single attempt to attack, in many other the median value of Attacker Situations is equal to 0 which means a visible win of Defender. Also a comparison with random-walk strategy proves a clear upper-hand of proposed method.

An important difference between our algorithm and existing approaches is flexibility. Our game model can be easily extended without making any changes to training algorithm since the base of the UCT method is an abstract model of states and moves which can reflect virtually any Attacker profile. Replacing the current Attacker strategy with any other, e.g. game theory-based solutions, randomly or systematically biased, or the one based on bounded rationality, is straightforward and does not require any changes in Defender algorithm.

The detailed conclusions include the observation that depending on a particular game different values for expansion coefficient C were more suitable than the others. The investigation of the quasi-formal rule for C parameter selection for a given game is one of our current targets. Furthermore, we plan to test our method in the cases when Defender resources are insufficient to secure all targets all the time.

Table 2. Number of Attack Situations and pay-offs in test games with different UCT configurations. The exploration coefficient (C) was equal to $4, 7, 12, 15$ or 20 and the the length of back-propagation of results len was one of $2, 3$ or 4. Game variant A denotes the first implementation variant described in Section 3.2 and B means the second one. $Configset$ column presents values of C in UCT formula (a number placed after ex) and back-propagation length (a number placed after len). The results for those $(game, parameterset)$) pairs for which there were no attack attempts in all 21 tests are omitted. Such situations favor Defender's behavior since the reluctance to attack may often stem from efficient patrolling schedule implemented by Defender.

		Attacker Situations			Overall pay-off			Random player
Game	Config set	mean	median	stddev	mean	median	stddev	avg pay-off
Graph2A	ex20-len2	0	0	0	102455.8	101805	2218.7	-547124
Graph2A	ex15-len2	1	0	2.9	104847.4	101552	6519.9	-547124
Graph2A	ex12-len4	0.8	0	1.5	107871.6	103169	12187	-547124
Graph2A	ex7-len2	908.7	0	2863.4	82902.6	80123	48089.9	-547124
Graph2A	ex4-len3	2486.2	5	4569	52859.1	79923	62337.1	-547124
Graph2B	ex15-len2	0	0	0	102553.3	100966	3197.3	-544089
Graph2B	ex12-len4	0.2	0	1.1	106392.3	104069	10478.7	-544089
Graph2B	ex7-len2	557.7	4	2536.1	107031.5	103290	22436.9	-544089
Graph2B	ex4-len3	1052.5	4	3311.9	75123	80126	45902.9	-544089
Graph1A	ex12-len4	0	0	0	79981.9	79987	241.7	-674773
Graph1A	ex7-len2	1907.2	0	8739.8	51276	79944	131656.6	-674773
Graph1A	ex4-len3	3806.5	0	12022.3	19356	80125	192247.8	-674773
Graph1B	ex7-len2	0	0	0	79950.3	79942	239.5	-674925.6
Graph1B	ex4-len3	3829.9	0	12096	19511.1	79862	190970.3	-674925.6
Grap3A	ex20-len2	0.8	0	1.9	80010.1	80064	218.9	-522846.5
Grap3A	ex15-len2	1.2	0	2.1	80062	80118	274.5	-522846.5
Grap3A	ex12-len4	356.6	4	1618.9	76033	79885	17669.8	-522846.5
Grap3A	ex7-len2	1928	0	5852.2	59649.7	79951	60579.8	-522846.5
Grap3A	ex4-len3	9084.2	12	11624.6	-20358.3	79538	125375	-522846.5
Grap3B	ex20-len2	2089.6	4	6617.7	83586.1	79908	13466	-522190.5
Grap3B	ex15-len2	2	0	3.1	79936.9	79948	194	-522190.5
Grap3B	ex12-len4	1169	2	5341.7	81385.2	79980	6734.9	-522190.5
Grap3B	ex7-len2	1041.3	6	2596.7	68444.9	79836	28531.5	-522190.5
Grap3B	ex4-len3	8270.5	21	10791.5	-10813	79423	118137.9	-522190.5

Acknowledgment. The research was financed by the National Science Centre in Poland, grant number DEC-2012/07/B/ST6/01527.

References

1. Ahmadizadeh, K., Gomes, C.P., Sabharwal, A.: Game playing techniques for optimization under uncertainty (2010)
2. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning 47(2-3), 235–256 (2002)

3. Fang, F., Jiang, A.X., Tambe, M.: Optimal patrol strategy for protecting moving targets with multiple mobile resources. In: Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems, pp. 957–964. International Foundation for Autonomous Agents and Multiagent Systems (2013)
4. Jain, M., Tsai, J., Pita, J., Kiekintveld, C., Rathi, S., Tambe, M., Ordóñez, F.: Software assistants for randomized patrol planning for the lax airport police and the federal air marshal service. Interfaces 40(4), 267–290 (2010)
5. Johnson, M.P., Fang, F., Tambe, M.: Patrol strategies to maximize pristine forest area. In: AAAI (2012)
6. Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 282–293. Springer, Heidelberg (2006)
7. Mańdziuk, J., Świechowski, M.: UCT application to Dynamic Vehicle Routing Problem with Heavy Traffic Jams (submitted) (2015)
8. Pita, J., John, R., Maheswaran, R., Tambe, M., Yang, R., Kraus, S.: A robust approach to addressing human adversaries in security games. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, vol. 3, pp. 1297–1298. International Foundation for Autonomous Agents and Multiagent Systems (2012)
9. Rimmel, A., Teytaud, F., Cazenave, T.: Optimization of the nested monte-carlo algorithm on the traveling salesman problem with time windows. In: Di Chio, C., et al. (eds.) EvoApplications 2011, Part II. LNCS, vol. 6625, pp. 501–510. Springer, Heidelberg (2011)
10. Sabharwal, A., Samulowitz, H., Reddy, C.: Guiding combinatorial optimization with UCT. In: Beldiceanu, N., Jussien, N., Pinson, É. (eds.) CPAIOR 2012. LNCS, vol. 7298, pp. 356–361. Springer, Heidelberg (2012)
11. Shieh, E., An, B., Yang, R., Tambe, M., Baldwin, C., DiRenzo, J., Maule, B., Meyer, G.: Protect: A deployed game theoretic system to protect the ports of the united states. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, vol. 1, pp. 13–20. International Foundation for Autonomous Agents and Multiagent Systems (2012)
12. Stackelberg, H.V.: Marktform und gleichgewicht. Springer, Vienna (1934)
13. Świechowski, M., Mańdziuk, J.: Self-adaptation of playing strategies in general game playing. IEEE Transactions on Computational Intelligence and AI in Games 6(4), 367–381 (2014)
14. Walędzik, K., Mańdziuk, J.: Multigame playing by means of UCT enhanced with automatically generated evaluation functions. In: Schmidhuber, J., Thórisson, K.R., Looks, M. (eds.) AGI 2011. LNCS(LNAI), vol. 6830, pp. 327–332. Springer, Heidelberg (2011)
15. Walędzik, K., Mańdziuk, J.: An automatically-generated evaluation function in general game playing. IEEE Transactions on Computational Intelligence and AI in Games 6(3), 258–270 (2014)
16. Walędzik, K., Mańdziuk, J., Zadrożny, S.: Proactive and reactive risk-aware project scheduling. In: 2nd IEEE Symposium on Computational Intelligence for Human-like Intelligence (CIHLI 2014), pp. 94–101. IEEE Press (2014)
17. Wang, Y., Gelly, S.: Modifications of uct and sequence-like simulations for monte-carlo go. CIG 7, 175–182 (2007)