Dynamic Escape Game

Demonstration

Antonio Di Stasio
Università degli Studi di Napoli Federico II

Paolo Domenico Lambiase
Università degli Studi di Napoli Federico II

Vadim Malvone
Université d'Évry

Aniello Murano
Università degli Studi di Napoli Federico II

ABSTRACT

We introduce Dynamic Escape Game (DEG), a tool that provides emergency evacuation plans in situations where some of the escape paths may become unavailable at runtime. We formalize the setting as a reachability two-player turn-based game where the universal player has the power of inhibiting at runtime some moves to the existential player. Thus, the universal player can change the structure of the game arena along a play. DEG uses a graphical interface to depict the game and displays a winning play whenever it exists.

ACM Reference Format:
Antonio Di Stasio, Paolo Domenico Lambiase, Vadim Malvone, and Aniello Murano. 2018. Dynamic Escape Game. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, July 10–15, 2018,* IFAAMAS, 3 pages.

1 INTRODUCTION

Game theory is a well-developed branch of mathematics, largely applied in computer science to reason about the strategic behavior of *reactive systems* [1, 13]. These are characterized by an ongoing interaction between two or more entities, modeled as players, and the behavior of the whole system deeply relies on this interaction [8]. To reason about resource constraints, quantitative games have been considered [4, 5]. Accordingly, games are played on weighted graphs, where edges are equipped with integer values modeling rewards or costs. Well-stated quantitative games are Mean-Payoff [6] and Energy Games [5]. In this paper, we consider a variant of weighted two-player turn-based reachability games. These are games with weighted transitions in which $Player_1$ (P_1) tries to reach a target state while $Player_2$ (P_2) tries to prevent it. Along a play, P_1 plays as usual, i.e. from his current position, he chooses one of the *available* successors and moves to it. Conversely, P_2, sitting on a set of states S, chooses some of its successors that become irrevocably unavailable to P_1 and added to S. The game starts with S being the initial state for P_2 (different from the one for P_1). P_2 chooses successors under the constraint that the sum of the weights along the involved edges is lower to a given bound. Note that P_2 can dynamically change the structure of the game. The introduced game framework takes inspiration from [20] and it can be usefully applied to solve questions in planning, rescue, and traffic control. Other applications and similar reasoning can be found in [2, 3, 9–12, 14–19]. To solve the above game, we introduce the tool Dynamic Escape Game (DEG). The solution of the game is based on the computation of two priority functions, one for each player, representing their best performing strategies respectively. Our empirical evaluation shows that the priority functions computed by DEG use good heuristic and have excellent runtime execution.

2 GAME STRUCTURE

Our tool works over two-player turn-based games, on weighted graphs under the reachability condition (2TGW, for short).

Definition 2.1. A 2TGW is a tuple $G = (\mathrm{P}, I_1, I_2, \mathrm{St}, \mathrm{E}, \mathrm{T}, w, b)$, where $\mathrm{P} = \{P_1, P_2\}$ is the set of players, I_j is the initial state for P_j, St is the set of states, $\mathrm{E} \subseteq \mathrm{St} \times \mathrm{St}$ is the set of edges, $\mathrm{T} \subseteq \mathrm{St}$ is the set of the target states, $w : \mathrm{E} \rightarrow \mathbb{N}$ is a function that given an edge returns its weight, and b is a bound.

Given a game G, the players move in turn, starting from their starting states, with P_1 moving first. The game makes use of a set S containing states that are unavailable to P_1 along a play. The game starts with $S = \{I_2\}$ and only P_2 can operate on it by possibly adding states. If the game is at P_1's round, given the current state s, P_1 can move in any successor state of $s \in St$, but those in S. If the game is at P_2's round then he can add to S any set $S' \in St \setminus S$ of states reachable from S whose sum of the weights of the traversed edges is not greater than b. A *configuration* is a couple $(s, \mathrm{S}) \in \mathrm{St} \times 2^{\mathrm{St}}$ where $s \notin S$. By C we denote the set of all configurations. A *play* is a finite sequence of configurations $\pi = (s_0, \mathrm{S}_0), \ldots, (s_n, \mathrm{S}_n)$, such that $s_0 = I_1$, $\mathrm{S}_0 = \{I_2\}$, $S_i \subseteq \mathrm{S}_{i+1}$, $(s_i, s_{i+1}) \in \mathrm{E}$, and for all $s' \in \mathrm{S}_{i+1} \setminus \mathrm{S}_i$ there exists $s \in \mathrm{S}_i$ such that $(s, s') \in \mathrm{E}$, where $0 \leq i < n$. Note that the maximum length of a play is $n = |\mathrm{St}| - 1$, since in the worst case $|S_n| = |\mathrm{St}| - 1$. A P_1 *strategy* is a function $\sigma_1 : \mathrm{C} \rightarrow \mathrm{St}$ such that for all $(s, \mathrm{S}) \in \mathrm{C}$ it holds that $\sigma_1(s, \mathrm{S}) \notin \mathrm{S}$ and $(s, \sigma_1(s, \mathrm{S})) \in \mathrm{E}$. A P_2 *strategy* is a function $\sigma_2 : \mathrm{C} \rightarrow 2^{\mathrm{St}}$ such that: *i)* for all $(s, \mathrm{S}) \in \mathrm{C}$ and for all $s' \in \sigma_2(s, \mathrm{S}) \setminus \mathrm{S}$ there exists $s \in \mathrm{S}$ such that $(s, s') \in \mathrm{E}$; *ii)* it holds that $\Sigma_{(s,s') \in \mathrm{E}} w(s, s') < b$, where $s \in \mathrm{S}$ and $s' \in \sigma_2(s, \mathrm{S})$. Given P_1 strategy σ_1 and P_2 strategies σ_2 they induce a play $\pi = (s_0, \mathrm{S}_0), \ldots, (s_n, \mathrm{S}_n)$ such that for all $0 \leq i \leq n$, $s_{i+1} = \sigma_1((s_i, S_i))$ and $S_{i+1} = \sigma_2((s_i, S_i))$. Finally, P_1 (*resp.*, P_2) *wins* the game G if there exists a P_1 (*resp.*, P_2) strategy such that for all P_2 (*resp.*, P_1) strategies, the induced play $\pi = (s_0, \mathrm{S}_0), \ldots, (s_n, \mathrm{S}_n)$ allows P_1 (*resp.*, prevents P_1) to visit a target state.

3 HOW TO COMPUTE THE STRATEGIES

In this section we describe the two functions we implemented in our tool which are used by the players to select the strategies to win the game. Let $dist(x, y)$ be the smallest distance *w.r.t.* the number of edges from x to y, v the current state of P_1, and S the set of

---

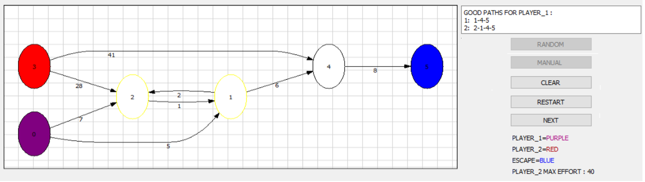
Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden.

states of P_2. The priority function δ_1 for P_1 works as follows: *(1)* The distances between S and target states are calculated. For each $t \in T$, such a distance is set as $\min_{s \in S}\{dist(s,t)\}$ and calculated by applying a BFS algorithm on the transpose graph for each $t \in T$ and by choosing the smallest value, which we call $distP2(t)$. *(2)* The distances between each state u such that $(v,u) \in E$ and each state $t \in T$ are calculated by using the BFS algorithm. So, if the state u cannot reach any target state, u is colored with *red*. Instead, if a target state $t \in T$ exists such that $dist(u,t) \leq distP2(t)$, then u is colored with *green* since P_2 cannot block P_1 to reach t. Finally, u is colored with *yellow* if $dist(u,t) > distP2$. Using the priority function δ_1, P_1 chooses a green state, if such a state exists, otherwise he chooses a *yellow* state with the smallest distance between u and a state $t \in T$. The priority function for P_2 works as follows: *(1)* The distances between v and each state $t \in T$ are calculated by applying the BFS algorithm. *(2)* For each $s \in S$ and for all u such that $(s,u) \in E$, the distance between u and the target states is calculated by using the BFS algorithm. Moreover, a priority, i.e. an integer value between 0 and 3, is associated to u in this way: *(i)* if $u \in T$ and $(v,u) \in E$, then the priority of u is 3. Instead, if $(v,u) \notin E$ the priority of u is 1 because P_2 is in advantage on this state; *(ii)* if there is a $t \in T$ such that $dist(u,t) < dist(v,t)$ and the constraint on the edges holds then the priority of u is 2. *(iii)* if there is no $t \in T$ such that $dist(u,t) < dist(v,t)$, then the priority of u is 0 because P_1 could escape anyway in t. Using the priority function described above, P_2 chooses a state starting from those with higher priority up to a smallest priority, as long as the sum of the weights of the edges is not over the bound.

Complexity result. Given a configuration $(s, S) \in C$, the above algorithm computes the priority functions δ_1 and δ_2 in $O(|V|^2 \cdot (|V| + |E|))$. Since in the worst case the number of rounds of the game is $|V|$, the overall complexity is $O(|V| \cdot [|V|^2 \cdot (|V| + |E|)])$.

4 THE TOOL

The GUI is depicted in Fig. 1. It consists of two parts: the Output Area (OA) and the Control Panel (CP). The OA is made of two frames. The one on the left side is used to depict the game graph. In particular, according to the positions of P_1, P_2, as well as the target states, the states of the graph are colored in the follow way: *purple* for P_1, *red* for P_2 and *blue* for the target states. The frame on the right upper side shows all *possible paths* that P_1 can follow starting from its adjacent states and ending to a target state. The CP at the right button corner is composed by five buttons: Manual (M), Random (RD), Clear (C), Restart (RS) and Next (N). The M button is used to build a graph manually. By pressing it, a window pops up in which the user sets the number of states of the game graph. Then, another window comes out in which the user decides how to connect the states. Finally, the user sets the initial states for P_1 and P_2 and the target states. The RD button works similarly to the M one, but generates a random game of n states, with n provided by the user. The C button allows to clean the graph area. The RS button allows to restart the current game. Finally, the N button runs automatically a move for the player's round. Note that there are two ways to move for P_1: by pressing the N button or manually. [1]



GOOD PATHS FOR PLAYER_1 :
1: 1-4-5
2: 2-1-4-5

PLAYER_1=PURPLE
PLAYER_2=RED
ESCAPE=BLUE
PLAYER_2 MAX EFFORT : 40

Figure 1: The main window of our tool.

5 BENCHMARKS

In this section we report experimental results to evaluate the performance between DEG and a brute force algorithm, named BF, that uses all possible strategies for P_1 and the priority function for P_2. Note that we used in BF the priority function for P_2 because it is optimal. In particular, we tested the tool on several instances by comparing the priority function with all possible strategies for P_2 and we observed that the output in both cases is the same for all instances. The tool[2] has been implemented in C++ and all tests have been run on an Intel Core i7-6500u with 8 GB of RAM running Microsoft Windows 10. The benchmarks show that over 500 instances, with $17 \leq |St| \leq 21$, DEG returns the correct solution, that is P_1 wins in both DEG and BF, in about 91% of the instances. Moreover, as reported in Fig. 2, DEG outperforms the BF algorithm in all the instances. In conclusion, the results show that DEG uses a good heuristic for P_1 and an excellent runtime execution.

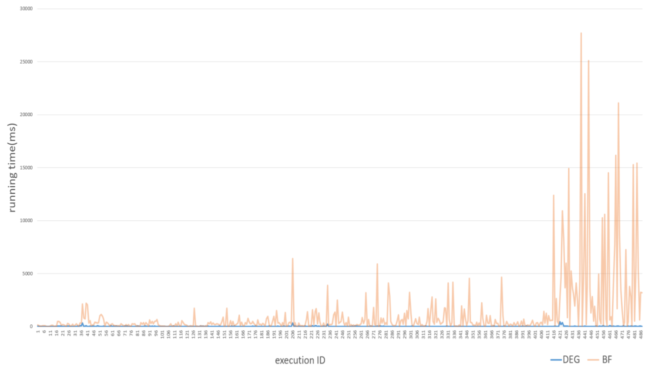


Figure 2: Comparison between DEG and BF w.r.t. running time execution.

6 CONCLUSION

This paper introduces Dynamic Escape Game, a tool solving a specific weighted game under reachability condition, where the opponent can dynamically modify the game arena. To solve the game we have introduced two priority functions to be used by the players. Our benchmarks have showed that our tool exhibits an excellent running-time execution. Also that the introduced priority functions are good heuristics. We believe that our tool can be used as a core engine to practically address real escape problems in MAS.

---

[1] A video demonstration is available from https://goo.gl/71FqHF

[2] The tool is available for download from https://bitbucket.org/antonylogic/deg/

REFERENCES

[1] R. Alur, T.A. Henzinger, and O. Kupferman. 2002. Alternating-Time Temporal Logic. *JACM* 49, 5 (2002), 672–713.

[2] B. Aminof, V. Malvone, A. Murano, and S. Rubin. 2016. Graded Strategy Logic: Reasoning about Uniqueness of Nash Equilibria. In *AAMAS'16*. 698–706.

[3] B. Aminof, V. Malvone, A. Murano, and S. Rubin. 2018. Graded Modalities in Strategy Logic. *Inf. Comput.* (2018), to appear.

[4] R. Bloem, K. Chatterjee, T. A. Henzinger, and B. Jobstmann. 2009. Better Quality in Synthesis through Quantitative Objectives. In *CAV 2009*. 140–156.

[5] A. Chakrabarti, L. de Alfaro, T. A. Henzinger, and M. Stoelinga. 2003. Resource Interfaces. In *EMSOFT 2003*. 117–133.

[6] A. Ehrenfeucht and J. Mycielski. 1979. Positional strategies for mean payoff games. *International Journal of Game Theory* 8, 2 (1979), 109–113.

[7] T. Gawlitza and H. Seidl. 2009. Games through Nested Fixpoints. In *CAV 2009*. 291–305.

[8] D. Harel and A. Pnueli. 1985. *On the Development of Reactive Systems*. Springer.

[9] W. Jamroga, V. Malvone, and A. Murano. 2017. Reasoning about Natural Strategic Ability. In *AAMAS17*. 714–722.

[10] W. Jamroga and A. Murano. 2014. On Module Checking and Strategies.. In *AAMAS14*. 701–708.

[11] W. Jamroga and A. Murano. 2015. Module Checking of Strategic Ability. In *AAMAS15*. 227–235.

[12] O. Keidar and N. Agmon. 2017. Safety First: Strategic Navigation in Adversarial Environments. In *AAMAS17*. 1581–1583.

[13] O. Kupferman, M.Y. Vardi, and P. Wolper. 2001. Module Checking. *IC* 164, 2 (2001), 322–344.

[14] C. Löding and P. Rohde. 2003. Model Checking and Satisfiability for Sabotage Modal Logic. In *FST TCS 2003*. 302–313.

[15] V. Malvone, F. Mogavero, A. Murano, and L. Sorrentino. 2015. On the Counting of Strategies. In *TIME 2015*. 170–179.

[16] V. Malvone, F. Mogavero, A. Murano, and L. Sorrentino. 2018. Reasoning about graded strategy quantifiers. *Inf. Comput.* 259, 3 (2018), 390–411.

[17] V. Malvone, A. Murano, and L. Sorrentino. 2015. Games with additional winning strategies. In *CILC 2015*. 175–180.

[18] V. Malvone, A. Murano, and L. Sorrentino. 2017. Hiding Actions in Multi-Player Games. In *AAMAS17*. 1205–1213.

[19] V. Malvone, A. Murano, and L. Sorrentino. 2018. Additional Winning Strategies in Reachability Games. *Fundam. Inform.* 159, 1-2 (2018), 175–195.

[20] A. Murano, G. Perelli, and S. Rubin. 2015. Multi-agent Path Planning in Known Dynamic Environments. In *PRIMA 2015*. 218–231.