**ABSTRACT**

We present *NWin*, a tool that allows to count all different winning strategies in two-player turn-based games under the reachability condition. *NWin* uses a graphical interface to build the game model and collect all acyclic and cyclic winning strategies. By means of benchmarks over random games we show that *NWin* has a good performance in practice.

**Categories and Subject Descriptors**

I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence — Multiagent Systems

**General Terms**

Algorithms, Performance

**Keywords**

Game theory; Reachability condition; Strategic reasoning

1. INTRODUCTION

*Game theory* is a powerful framework with several applications in MAS [14]. It allows reasoning efficiently about the strategic behavior of reactive systems [2,9,13]. In this paper we consider two-players turn-based games played under the *reachability* condition, i.e. some nodes are declared *target*. Solving such a game amounts to check whether one of the players has a *winning strategy*, i.e., a sequence of moves that lead him to a target node no matter how the opponent acts.

In many cases, knowing how many winning solutions exist is crucial [4,6–8,10,11]. For example, in planning a rescue, it would give to a rescuer some backup plan in case something goes wrong. In solution concepts, this would cope with the *uniqueness* problem in Nash equilibrium (see [1,5,12]).

In this paper we introduce *NWin*, a tool that addresses the quantitative question of checking how many different strategies a player has to win a two-player reachability game. The tool uses a graphical interface to build the game model and shows all acyclic and cyclic winning strategies. By means of benchmarks over random games we show that *NWin* has a good performance in practice: it can process $10^8$ acyclic strategies in less than 10 seconds by a pc.

2. THE TOOL

In this section we describe the Graphical User Interface (GUI) of *NWin*, in the current release (1.0). Our tool works over two-player turn-based games, played by Player 1 and Player 2, under the reachability condition (2TG, for short). These are directed graphs in which the set of nodes ($V$) is partitioned in two subsets $V_1$ and $V_2$. Each $V_i$ contains all nodes owned by Player $i$. Nodes in $V_1$ ($V_2$) are graphically represented by a circle (square). The unique initial node is colored in blue and all target nodes ($T$) in green. Take a node $v$, the set of outcoming edges represent all possible moves of the player that owns $v$. A play is a path starting from the initial node and represents a sequence of performed moves from the players. A strategy for a player is the set of moves he can perform along every prefix of a play. A strategy is *winning* for Player 1 if it induces a play that reaches a target node, no matter which strategy Player 2 uses.

The GUI is depicted in Figure 1. It consists of two parts: the Output Areas (OA) and the Control Panel (CP). The OA is made of three frames. The one on the left side (Graph) is used to depict the game graph. The frame on the right upper side (Path to Target) shows all reachable paths that start at the initial node and end in a target node. Finally, the frame of right lower side (Strategy Path) reports all winning strategy plays for Player 1. The CP at the right down corner is composed by four buttons: Draw Graph (DG), Random Graph (RG), Base Strategies (BS), and All Strategies (AS). The DG button is used to build a graph manually. By pressing it, a window pops up in which the user sets the number of nodes and edges of the game graph. Then, another window comes out in which the user decides how the nodes are connected. Finally, he sets in a new window the initial number of nodes and edges of the game graph. Then, another window comes out in which the user decides how the nodes are connected. Finally, he sets in a new window the initial number of nodes and edges of the game graph.
and the set of target nodes. The RG button works similarly to the DG button but after the parameters in the first window are set the game is generated randomly. The BS button works as follows. The first time it is pressed, all reachable paths are shown. At the second press it gives all possible acyclic winning strategy plays in which the Player 1 wins. The AS button works as the BS button, but it also reports the cyclic plays. Finally, if we select a winning strategy play in the frame Strategy Path and push the button View (V), in the game graph the path appears in red.

A video demonstration of NWin is on YouTube.

3. HOW TO COMPUTE THE STRATEGIES

Given a 2TG, NWin computes all winning strategies through three main steps. The first two are preliminary and used to partition all nodes in “good” and “bad”.

The first step calculates all acyclic reachability paths, i.e., the acyclic paths, from the initial node to a target node, that are present in the monolithic version of the game graph. This is done by applying a classic DFS algorithm. Precisely, the algorithm uses two sets of nodes, namely white ($W$) and black ($B$). The set $W$ ($B$) represents all nodes that can (cannot) reach a target node in $T$. This step starts with $B = \emptyset$ and $W = T$. For each node $v$, if it is adjacent to a node in $W$, then $W = W \cup \{v\}$, otherwise $B = B \cup \{v\}$. The algorithm ends when a fixed point is reached.

The second step works on the nodes in $\{W \cap V_2\} \setminus T$, i.e., the nodes of Player 2 that are white but not target nodes. Along this step we remove a node $v$ from $W$ and add it in $B$ iff at least one of the following conditions is satisfied: (i) $v$ has a loop; (ii) $v$ is adjacent to a node in $B$; (iii) $v$ belongs to a trap cycle $C$ for Player 1 (from which he can not go out). Note that, each $v' \in C \cap V_1$ has a single outcome edge.

The third step considers the entire game and in particular the strategies of the players. Given a node $v \in W \cap V_1 \setminus T$, Player 1 has a degree of preference in the mining that, if there exists a node $v'$ adjacent to $v$ and $v' \in W$, then he chooses and moves to $v'$ (even if this node has been already visited). Similarly, given a node $v \in W \cap V_2$, if there exists a node $v'$ adjacent to $v$ and $v' \in B$, then Player 2 chooses $v'$, otherwise he prefers to choose a node that has been already visited, if possible. This step, and thus the algorithm, ends when a target node or a node in $B$ is entered.

It is important to note that, after the first step the tool prints all possible reachability paths. All winning strategy plays are printed only after the third step is completed.

4. CRITICAL ISSUE

To compute all winning strategy plays, we have faced with two main critical issues: all the edges need to be visited in order to discover cycles and nodes require to be visited more than ones whenever they belong to a winning strategy play. The classical DFS is too weak to handle these two points. Indeed, the DFS analyzes all the nodes of the graph, but it does not analyze all edges and so all possible moves of both players. For this reason, we have introduced opportune extensions of the DFS. The first we consider is DFSR, in which it allows the algorithm to revisit a node in order to consider all possible edges and all possible acyclic winning strategy plays. In order to retrieve the winning strategy plays for Player 1 that contain also cycles, we have built MDFSR.

5. PERFORMANCE

In this section, we report on some benchmarks. For the lack of space we only describe the ones over DSFR. We have run NWin on an i5 processor with 4GB of RAM. We have considered random games evaluated under the DG button by setting as number of nodes $n \in \{2, \ldots, 16\}$ and number of edge $e \in \{n + 1, \ldots, n^2\}$. In particular, we have executed one hundred cases for each $n$ and thus one thousand and five hundred games in total. For each game, we have collected the execution time and the number of paths processed.

The benchmarks are depicted in Figure 2 in which we report the number of paths ($\#\text{Path}$) processed per second ($s$). The graph gives a proportional relationship between the number of paths processed and the seconds that NWin takes to process such paths. In particular, it shows that under 10 seconds NWin can process more than $10^5$ paths. It is worth observing that we analyze the number of paths rather than nodes as the former is the fulcrum of NWin.

6. CONCLUSION

In this paper, we have presented NWin a tool that allows to retrieve all possible winning strategy plays in a two-player game under the reachability condition. NWin makes use of a simple but efficient GUI that allows to build the game under two options: random and manual. The tool calculates and reports, upon request, all acyclic and cyclic winning strategy plays. We have reported on some benchmarks on the former case and showed that NWin can calculate more than $10^8$ paths in less than 10 seconds. We believe that our tool can be used as a core engine to solve several problems in MAS. As future work we plan to extend this tool in the setting of solution concepts, e.g. Nash Equilibrium (NE). In particular, by means of this extension, we aim to solve the uniqueness of NE in practice.
REFERENCES