Computing the winning coalitions of a turn-based game with reachability objectives

Loïs Vanhée
ENS Cachan Bretagne
lois.vanhee@irisa.fr

Sophie Pinchinat
IRISA
sophie.pinchinat@irisa.fr

We consider the setting of multi-agent systems designed to achieve an objective e.g. a multi-node distributed system that has to perform a task in a bounded amount of time. Whatever the objective is, a way to enforce it can be to restrict the behaviour of a subset of its agents; we call this subset a *coalition*. Note that, depending on the initial configuration of the system, the set of winning coalitions may vary. This shows that the link between the initial configuration of the system and the winning coalitions is tricky, therefore interesting to determine. In this work, we propose an algorithm which computes the set of winning coalitions for any configuration.

Formally, we consider the problem of computing the set W of winning coalitions for every position in a turn-based game with a reachability objective. Incidentally, the more general class of concurrent games may have been considered, but the turn-based assumption makes the framework easier to handle. Let us define a turn-based game as a structure $G = (P, A, T, Goal)$ where P is a set of positions, each controlled by an agent in the set of agents A, T is the transition function on these positions and $Goal \subseteq P$ denotes the set of goals. In real cases, the complexity of manipulating P and T is huge because $|P|$ is exponential in the number of agents. A play is an infinite sequence of positions pl s.t. for any position $pl(i+1)$ is a successor of $pl(i)$ by T. A strategy for a set of agents $A' \subseteq A$ is a function $s : P \to P$ (since the winning condition is reachability we can restrict to memoryless strategies) defined for every p s.t. the owner of p is in A' and $s(p)$ is a successor of p in T. A position $p \in P$ *is winning for a coalition C (or C is winning from p)* if there exists a strategy s for C s.t. whatever the strategies of the agents in $A \setminus C$ are, any play pl starting from p s.t. for every i, $pl(i+1) = s(pl(i))$, if $pl(i)$ is owned by an agent in C, pl contains a position in $Goal$. Let us note that if a coalition C is winning from p then any coalition $C' \supseteq C$ is winning from p (we call this the *winning monotonicity*). The desired result $W \subseteq (P \times 2^A)$ is $W = \{(p, C)|C$ is winning from $p\}$.

Deciding if a given coalition is winning from a set of position can be reduced to deciding if an agent is winning in a 2-player game (in affecting the coalition to the first player and other agents to the second player). Using the *pre* operation in the attractor algorithm of [3, pages 34-37] solves this problem. Unfortunately, to our knowledge, no algorithm can answer in time less than $O(d * |T|)$ where d is the size of the longest path without cycles, starting from the initial position to the goal.

In this contribution, we present an algorithm to compute $W = \{(p, C)|C$ is winning from $p\}$ ordered first by positions and then by coalitions. Notice that such computation is worth: once W is computed, model-checking if a coalition C is winning from a position p can be achieved in searching $(p, C) \in W$ in $O(|A| * log(|P|))$ (a search by dichotomy), instead of model-checking it, in $O(d * |T|)$. W allows to efficiently test sophisticated relations between initial position and the winning coalitions: given any set $S \subseteq (P \times 2^A)$, $W \cap S = \emptyset$ or $S \subseteq W$ are such interesting tests. For instance, given a formula f on the positions and f' on the coalitions, $S = \{(p, C)|p$ verifies f and C verifies $f'\}$, $S \subseteq W$ answers the question "does any coalition validating f' is winning from any position validating f?".

In order to be more realistic, we may introduce a cost to form the coalition that we want to minimize (i.e. computing the cheapest winning coalition). In the distributed system example, an optimal coalition is a least set of nodes that performs the task in a bounded amount of time. Then, computing the cheapest winning coalition from any position can be easily done in $O((|A| + d * |T|) * 2^{|A|})$, in model-checking every coalition. Notice that the problem we solve is an extension of the NP-complete problem presented in [2]. Once W is computed, instead of model-checking if C is winning from p, we can test $(p, C) \in W$, cutting down the complexity. Consequently, computing the cheapest winning coalition can be achieved in $O(log(|P|) + |A| * 2^{|A|})$. Since W is sorted by positions first, then by coalitions, the set of coalitions winning from p is obtained in $O(log(|P|))$, then we look for the cheapest coalition into this set. If the initial position is in a set $P' \subseteq P$ (e.g. because of a partial observation on the system), then, computing the cheapest winning coalition that enforces victory from any position from P' can be easily achieved in searching for the minimal coalition of $\cap_{p \in P'} \{C | (p, C) \in W\}$.

Back to our contribution, our algorithm is dual to the attractor algorithm [3, pages 34-37] which computes for a given coalition C, by iterating the *pre* operation, the set of winning positions from *Goal* until stabilization. Let pre^i (resp. W^i) be the set computed by the i^{th} iteration of the *pre* (resp. our) algorithm. We prove that for every pair (p, C), $(p, C) \in W^i$ iff $p \in pre^i$ for C. This entails validity and termination of our algorithm. The worse-case complexity of our algorithm is equivalent to computing the *pre* algorithm for every coalition and for every position. This procedure has the same complexity as the search for the cheapest winning coalition, but W can be used as pre-computation for multiple cheapest coalition searches and interrogation operations.

We enhanced performances of our algorithm making it tractable: in representing the set of winning coalitions by the minimal winning ones (thanks to the winning monotonicity) using a technique similar to [4] and using OBDDs [1] to represent symbolically games and coalitions and to order W.

Empirically, tests showed that the runtimes are of the order of $2^{\alpha|A|} * |T|$ with $\alpha < 1$. Moreover, when $|T|$ increases, α seems to decrease. We believe the OBDDs variable ordering and the compression of the OBDD data-structure play some role here.

To conclude, we have developed an algorithm to solve the problem of computing the set of winning coalitions for every position in a turn-based game with a reachability objective. As explained, such computation is worth as it can be exploited to compute efficiently the cheapest winning coalition of any set of positions and to perform more elaborated analysis.

References

[1] *Systems and software verification: model-checking techniques and tools.* Springer-Verlag New York, Inc., New York, NY, USA, 1999.

[2] Thomas Brihaye, Mohamed Ghannem, Nicolas Markey, and Lionel Rieg. Good friends are hard to find! In *TIME '08: Proceedings of the 2008 15th International Symposium on Temporal Representation and Reasoning*, pages 32–40, Washington, DC, USA, 2008. IEEE Computer Society.

[3] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science.* Springer, 2002.

[4] Shin-ichi Minato and Hiroki Arimura. Frequent pattern mining and knowledge indexing based on zero-suppressed bdds. In *KDID'06: Proceedings of the 5th international conference on Knowledge discovery in inductive databases*, pages 152–169, Berlin, Heidelberg, 2007. Springer-Verlag.