Deterministic Graphical Games Revisited[⋆]

Daniel Andersson, Kristoffer Arnsfelt Hansen,
Peter Bro Miltersen, and Troels Bjerre Sørensen

University of Aarhus, Datalogisk Institut, DK-8200 Aarhus N, Denmark
{koda,arnsfelt,bromille,trold}@daimi.au.dk

Abstract. We revisit the *deterministic graphical games* of Washburn. A deterministic graphical game can be described as a simple stochastic game (a notion due to Anne Condon), except that we allow arbitrary real payoffs but disallow moves of chance. We study the complexity of solving deterministic graphical games and obtain an almost-linear time comparison-based algorithm for computing an equilibrium of such a game. The existence of a linear time comparison-based algorithm remains an open problem.

1 Introduction

Understanding rational behavior in *infinite duration games* has been an important theme in pure as well as computational game theory for several decades. A number of central problems remain unsolved. In pure game theory, the existence of near-equilibria in general-sum two-player stochastic games were established in a celebrated result by Vieille [14, 15], but the existence of near-equilibria for the three-player case remains an important and elusive open problem [2]. In computational game theory, Condon [4] delineated the efficient computation of positional equilibria in *simple stochastic games* as an important task. While Condon showed this task to be doable in $\mathbf{NP} \cap \mathbf{coNP}$, to this day, the best deterministic algorithms are not known to be of subexponential complexity. To the computer science community, the problem of computing positional equilibria in simple stochastic games is motivated by its hardness for finding equilibria in many other natural classes of games [17], which again implies hardness for tasks such as model checking the μ-calculus [5], which is relevant for the *formal verification* of computerized systems.

1.1 Simple Stochastic Games

A simple stochastic game [4] is given by a graph $G = (V, E)$. The vertices in V are the *positions* of the game. Each vertex belongs either to player *Max*, to player *Min*, or to *Chance*. There is a distinguished *starting position* v_0. Furthermore, there are a number of distinguished *terminal positions* or just *terminals*, each

⋆ Research supported by *Center for Algorithmic Game Theory*, funded by The Carlsberg Foundation.

labeled with a *payoff* from Min to Max.[1] All positions except the terminal ones have outgoing arcs. The game is played by initially placing a token on v_0, letting the token move along a uniformly randomly chosen outgoing arc when it is in a position belonging to Chance and letting each of the players decide along which outgoing arc to move the token when it is in a position belonging to him. If a terminal is reached, then Min pays Max its payoff and the game ends. Infinite play yields payoff 0. A *positional strategy* for a player is a selection of one outgoing arc for each of his positions. He plays according to the strategy if he moves along these arcs whenever he is to move. It is known (see [4]) that each position p in a simple stochastic game can be assigned a *value* $\mathrm{Val}(p)$ so that:

1. Max has a positional strategy that, regardless of what strategy Min adopts, ensures an expected payoff of at least $\mathrm{Val}(p)$ if the game starts in p.
2. Min has a positional strategy that, regardless of what strategy Max adopts, ensures that the expected payoff is at most $\mathrm{Val}(p)$ if the game starts in p.

The value of the game itself is the value of v_0. Condon considered the complexity of computing this value. It is still open if this can be done in polynomial time. In the present paper, we shall look at some easier problems. For those, we want to make some distinctions which are inconsequential when considering whether the problems are polynomial time solvable or not, but important for the more precise (almost linear) time bounds that we will be interested in in this paper.

– A *weak solution* is $\mathrm{Val}(v_0)$ *and* a positional strategy for each player satisfying the conditions in items 1 and 2 above for $p = v_0$.
– A *strong solution* is the list of values of *all* positions in the game *and* a positional strategy for each player that for *all* positions p, ensures an expected payoff of at least/most $\mathrm{Val}(p)$ if the game starts in p.

In game theory terminology, weak solutions to a game are *Nash equilibria* of the game while strong solutions are *subgame perfect equilibria*. Figure 1 illustrates that the distinction is not inconsequential. In the weak solution to the left, Max

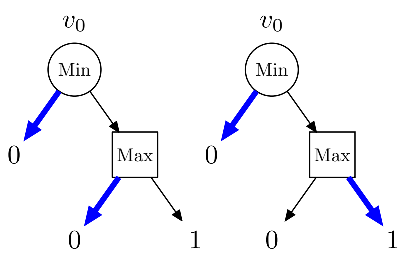

Fig. 1. The left solution is weak, the right is strong.

[1] In Condon's original paper, there were only two terminals, with the payoffs 0 and 1. The relaxation to arbitrary payoffs that we adopt here is fairly standard.

(if he gets to move) is content to achieve payoff 0, the value of the game, even though he could achieve payoff 1. Note that the game in Figure 1 is *acyclic*. In contrast to the general case, it is of course well known that a strong solution to an acyclic game can be found in linear time by straightforward dynamic programming (known as *backwards induction* in the game theory community). We shall say that we weakly (resp. strongly) solve a given game when we provide a weak (resp. strong) solution to the game. Note that when talking about strong solutions, the starting position is irrelevant and does not have to be specified.

1.2 Deterministic Graphical Games

Condon observed that for the case of a simple stochastic game with *no Chance positions* and only 0/1 payoffs, the game can be strongly solved in *linear time*. Interestingly, Condon's algorithm has been discovered and described independently by the artificial intelligence community where it is known under the name of *retrograde analysis* [13]. It is used routinely in practice for finding optimal strategies for combinatorial games that are small enough for the game graph to be represented in (internal or external) memory and where dealing with the possibility of cycling is a non-trivial aspect of the optimal strategies. The best known example is the construction of tables for *chess endgames* [8]. Condon's algorithm (and retrograde analysis) being linear time depends crucially on the fact that the games considered are win/lose games (or, as is usually the case in the AI literature, win/lose/draw games), i.e., that terminal payoffs are either 0 or 1 (or possibly also -1, or in some AI examples even a small range of integers, e.g., [12]). In this paper we consider the algorithmic problem arising when *arbitrary real payoffs* are allowed. That is, we consider a class of games similar to but incomparable to Condon's simple stochastic games: We disallow chance vertices, but allow arbitrary real payoffs. The resulting class were named *deterministic graphical*[2] *games* by Washburn [16].

Some simple examples of deterministic graphical games are given in Figure 2. In (a), the unique strong solution is for Min to choose right and for Max to choose left. Thus, the outcome is infinite play. In (b), the unique strong solution is for

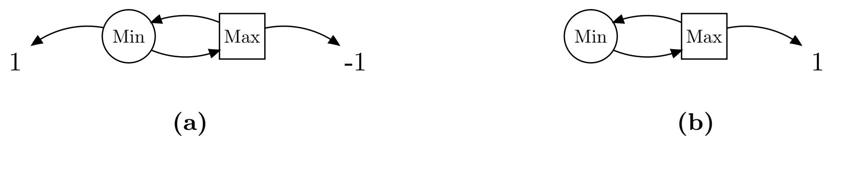

Fig. 2. (a) Infinite play equilibrium. **(b)** All values are 1, but one choice is suboptimal.

Min to choose right and for Max to choose right. The values of both vertices are

[2] There is no relation to the more recent concept of "graphical games" — a succinct representation for multi-player games [9].

1, but we observe that it is *not* a sufficient criterion for correct play to choose a vertex with at least as good a value as your current vertex. In particular, according to this criterion, Max could choose left, but this would lead to infinite play and a payoff of 0, which is a suboptimal outcome for Max.

Washburn [16] gives an algorithm for computing a strong solution to a deterministic graphical game, but its running time is cubic in the size of the game. We observe below that if a *sorted* list of the payoffs (with pointers to the corresponding terminals of the game) is given in advance, optimal strategies can again be found in linear time without further processing of the payoffs. From this it follows that a deterministic graphical game with n payoffs and m arcs in the graph can be strongly solved in time $O(n \log n + m)$ by a *comparison-based* algorithm. The main question we attempt to approach in this paper is the following:

Main Question. *Can a deterministic graphical game be (weakly or strongly) solved in linear time by a comparison-based algorithm?*

We believe this to be an interesting question, both in the context of game solving (deterministic graphical games being a very simple yet non-trivial natural variant of the general problem) and in the context of the study of comparison-based algorithms and comparison complexity. This paper provides neither a positive nor a negative answer to the question, but we obtain a number of partial results, described in the next subsection.

1.3 Our Results

Throughout this section we consider deterministic graphical games with n denoting the number of terminals (i.e., number of payoffs) and m denoting the total size (i.e., number of arcs) of the graph defining the game. We can assume $m \geq n$, as terminals without incoming arcs are irrelevant.

Strategy Recovery in Linear Time. The example of Figure 2 (b) shows that it is not completely trivial to obtain a strong solution from a list of values of the vertices. We show that this task can be done in linear time, i.e. time $O(m)$. Thus, when constructing algorithms for obtaining a strong solution, one can concentrate on the task of computing the values $\mathrm{Val}(p)$ for all p. Similarly, we show that given the value of just the starting position, a weak solution to the game can be computed in linear time.

The Number of Comparisons. When considering comparison-based algorithms, it is natural to study the number of comparisons used separately from the running time of the algorithm (assuming a standard random access machine). By an easy reduction from sorting, we show that there is no comparison-based algorithm that *strongly* solves a given game using only $O(n)$ comparisons. In fact, $\Omega(n \log n)$ comparisons are necessary. In contrast, Mike Paterson (personal communication) has observed that a deterministic graphical game can be *weakly* solved using $O(n)$ comparisons and $O(m \log n)$ time. With his kind permission, his algorithm is included in this paper. This also means that for the case of weak solutions, our main open problem cannot be solved in the negative using current lower-bound techniques, as it is not the number of comparisons that is the bottleneck. Our lower bound uses a game with $m = \Theta(n \log n)$ arcs. Thus, the following interesting open question concerning only the comparison complexity remains: *Can a deterministic graphical game be* strongly *solved using $O(m)$ comparisons?* If resolved in the negative, it will resolve our main open problem for the case of strong solutions.

Almost-Linear Time Algorithm for Weak Solutions. As stated above, Mike Paterson has observed that a deterministic graphical game can be weakly solved using $O(n)$ comparisons and $O(m \log n)$ time. We refine his algorithm and obtain an algorithm that weakly solves a game using $O(n)$ comparisons and only $O(m \log \log n)$ time. Also, we obtain an algorithm that weakly solves a game in time $O(m + m(\log^* m - \log^* \frac{m}{n}))$ but uses a superlinear number of comparisons. For the case of *strongly* solving a game, we have no better bounds than those derived from the simple algorithm described in Section 1.2, i.e., $O(m + n \log n)$ time and $O(n \log n)$ comparisons. Note that the bound $O(m+m(\log^* m - \log^* \frac{m}{n}))$ is linear in m whenever $m \geq n \log \log \ldots \log n$ for a constant number of 'log's. Hence it is at least as good a bound as $O(m + n \log n)$, for any setting of the parameters m, n.

2 Preliminaries

Definition 1. *A* deterministic graphical game (DGG) *is a digraph with vertices partitioned into sets of non-terminals V_{Min} and V_{Max}, which are game positions where player* Min *and* Max*, respectively, chooses the next move (arc), and terminals T, where the game ends and* Min *pays* Max *the amount specified by $p : T \to \mathbf{R}$.* □

For simplicity, we will assume that terminals have distinct payoffs, i.e., that p is injective. We can easily simulate this by artificially distinguishing terminals with equal payoffs in some arbitrary (but consistent) fashion. We will also assume that $m \geq n$, since terminals without incoming arcs are irrelevant.

Definition 2. *We denote by $\text{Val}_G(v)$ the value of the game G when the vertex v is used as the initial position and infinite play is interpreted as a zero payoff. This will also be called "the value of v (in G)".* □

Remark 1. That such a value indeed exists will follow from Proposition 1. We shall later see how to construct optimal *strategies* from vertex values.

Definition 3. *To* merge *a non-terminal v with a terminal t is to remove all outgoing arcs of v, reconnect all its incoming arcs to t, and then remove v.* □

The definitions of a *strong* and a *weak* solution are as stated in the introduction. The following algorithm is a generalization of Condon's linear time algorithm [4] for solving deterministic graphical games with payoffs in $\{0, 1\}$. That algorithm is known as *retrograde analysis* in the AI community [13], and we shall adopt this name also for this more general version.

Proposition 1. *Given a DGG and a permutation that orders its terminals, we can find a strong solution to the game in linear time and using no further comparisons of payoffs.*

Proof. If all payoffs are 0, then all values are 0 and every strategy is optimal.

Suppose that the minimum payoff $p(t)$ is negative. Any incoming arc to t from a Max-vertex that is not the only outgoing arc from that vertex is clearly suboptimal and can be discarded. Each other incoming arc is an *optimal choice* for its source vertex, which can therefore be merged with t. Symmetric reasoning applies when the maximum payoff is positive. $\square$

This immediately yields the *sorting method* for strongly solving DGGs: First sort the payoffs, and then apply Proposition 1.

Corollary 1. *A DGG with m arcs and n terminals can be strongly solved in $O(m + n \log n)$ time.* $\square$

Definition 4. *To* merge *a terminal s with another terminal t is to reconnect all incoming arcs of s to t and then remove s. Two terminals are* adjacent *if their payoffs have the same sign and no other terminal has a payoff in between.* $\square$

The following lemma states the intuitive fact that when we merge two adjacent terminals, the only non-terminals affected are those with the corresponding values, and they acquire the same (merged) value.

Lemma 1. *If G' is obtained from the DGG G by merging a terminal s with an adjacent terminal t, then for each non-terminal v, we have*

$$\mathrm{Val}_{G'}(v) = \begin{cases} \mathrm{Val}_G(v) & \textit{if } \mathrm{Val}_G(v) \neq \mathrm{Val}_G(s), \\ \mathrm{Val}_{G'}(t) & \textit{if } \mathrm{Val}_G(v) = \mathrm{Val}_G(s). \end{cases} \tag{1}$$

$\square$

Proof. Consider all DGGs with a fixed structure (i.e., underlying graph) but with varying payoffs. Since a strong solution can be computed by a comparison-based algorithm (Proposition 1), the value of any particular position v can be described by a min/max formula over the payoffs. The claim of the lemma can be seen to be true by a simple induction in the size of the relevant formula.

By repeatedly merging adjacent terminals, we "coarsen" the game. Figure 3 shows an example of this. The partitioning method we shall use to construct coarse games in this paper also yields sorted lists of their payoffs. Hence, we shall be able to apply retrograde analysis to solve them in linear time.

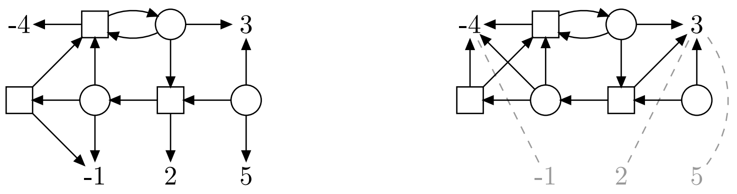

Fig. 3. Coarsening by merging $\{-4, -1\}$ and $\{2, 3, 5\}$.

Corollary 2. *The signs of the values of all vertices in a given DGG can be determined in linear time.*

Proof. Merge all terminals with negative payoffs into one, do likewise for those with positive payoffs, and then solve the resulting coarse "win/lose/draw" game by retrograde analysis. □

Clearly, arcs between vertices with different values cannot be part of a strong solution to a game. From this, the following lemma is immediate.

Lemma 2. *In a DGG, removing an arc between two vertices with different values does not affect the value of any vertex.* □

Remark 2. Corollary 2, Lemma 2, and symmetry together allow us to restrict our attention to games where all vertices have positive values, as will be done in subsequent sections.

Proposition 2. *Given the value of the initial position of a DGG, a weak solution can be found in linear time. If the values of* all *positions are known, a strong solution can be found in linear time.*

Proof. In the first case, let y be the value of initial position v_0. We partition payoffs in at most five intervals: $(-\infty, \min(y, 0))$, $\{\min(y, 0)\}$, $(\min(y, 0), \max(y, 0))$, $\{\max(y, 0)\}$ and $(\max(y, 0), \infty)$. We merge all terminals in each of the intervals, obtaining a game with at most five terminals. A strong solution for the resulting coarse game is found in linear time by retrograde analysis. The pair of strategies obtained is then a weak solution to the original game, by Lemma 1.

In the second case, by Lemma 2, we can first discard all arcs between vertices of different values. This disintegrates the game into smaller games where all vertices have the same value. We find a strong solution to each of these games in linear time using retrograde analysis. Combining these solutions in the obvious way yields a strong solution to the original game, by Lemma 2. □

3 Solving Deterministic Graphical Games

3.1 Strongly

For solving DGGs in the strong sense, we currently know no asymptotically faster method than completely sorting the payoffs. Also, the number of *comparisons* this method performs is, when we consider bounds *only depending on the number of terminals n*, optimal. Any sorting network [10] can be implemented by an acyclic DGG, by simulating each comparator by a Max-vertex and a Min-vertex. Figure 4 shows an example of this. Thus, we have the following tight bound.

Proposition 3. *Strongly solving a DGG with n terminals requires $\Theta(n \log n)$ comparisons in the worst case.* □

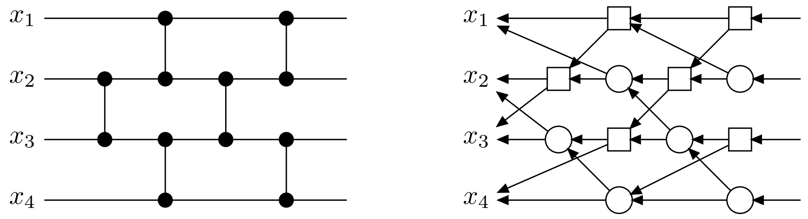

Fig. 4. Implementing a sorting network by a deterministic graphical game.

Implementing the asymptotically optimal AKS-network [1] results in a game with $\Theta(n \log n)$ vertices and arcs. Thus, it is still consistent with our current knowledge that a game can be strongly solved using $O(m)$ comparisons.

3.2 Weakly

The algorithms we propose for weakly solving DGGs all combine coarsening of the set of payoffs with retrograde analysis. By splitting the work between these two operations in different ways, we get different time/comparison trade-offs. At one extreme is the sorting method. At the other, we partition the payoffs around their median (which can be done in linear time by Blum *et al.* [3]), use retrograde analysis to solve the coarse game obtained by merging the terminals in each half, and then discard the irrelevant half of the terminals (the one *not* containing the value of the starting vertex) and all vertices with the corresponding values. This method, which is due to Mike Paterson, uses the optimal $O(n)$ comparisons, but requires $\Theta(\log n)$ iterations, each with a worst case running time of $\Theta(m)$.

$O(n)$ Comparisons and $O(m \log \log n)$ Time. To improve the running time of Paterson's algorithm, we stop and sort the remaining terminals as soon as this can be done in $O(n)$ time. The number of comparisons is still $O(n)$. As noted in Section 2, we may assume that all vertices have positive values.

Algorithm. Given a DGG G with m arcs, n terminals, and starting position v_0, do the following for $i = 0, 1, 2, \ldots$

1. Partition the current set of n_i terminals around their median payoff.

2. Solve the coarse game obtained by merging the terminals in each half.
3. Remove all vertices that do not have values in the half containing $\mathrm{Val}_G(v_0)$.
4. Undo step 1 for the half of v_0.

When $n_i \log n_i \le n$, stop and solve the remaining game by the sorting method.

Analysis. Steps 1–4 can be performed in $O(m)$ time and $O(n_i)$ comparisons. The number of iterations is $O(\log n - \log f(n))$, where $f(n)$ is the inverse of $n \mapsto n \log n$, and since this equals $O(\log \log n)$ we have the following.

Theorem 1. *A DGG with m arcs and n terminals can be weakly solved in $O(m \log \log n)$ time and $O(n)$ comparisons.* □

Almost-Linear Time. We can balance the partitioning and retrograde analysis to achieve an almost linear running time, by a technique similar to the one used in [7] and later generalized in [11].[3] Again, we assume that all vertices have positive values.

Algorithm. Given a DGG G with m arcs, n terminals, and starting position v_0, do the following for $i = 0, 1, 2, \ldots$

1. Partition the current set of n_i terminals into groups of size at most $n_i/2^{m/n_i}$.
2. Solve the coarse game obtained by merging the terminals in each group.
3. Remove all vertices having values outside the group of $\mathrm{Val}_G(v_0)$.
4. Undo step 1 for the group of v_0.

When $n_i/2^{m/n_i} < 1$, stop and solve the remaining game by the sorting method.

Analysis. All steps can be performed in $O(m)$ time. For the first step we can do a "partial perfect quicksort", where we always partition around the median and stop at level $\lceil m/n_i \rceil + 1$.

To bound the number of iterations, we note that n_i satisfies the recurrence

$$n_{i+1} \le n_i/2^{m/n_i} , \tag{2}$$

which by induction gives

$$n_i \le \frac{n}{\underbrace{b^{b^{\cdot^{\cdot^{\cdot^{b}}}}}}_{i}} \tag{3}$$

where $b = 2^{m/n}$. Thus, the number of iterations is $O(\log_b^* n)$, where $\log_b^*$ denotes the number of times we need to apply the base b logarithm function to get below 1. This is easily seen to be the same as $O(1 + \log^* m - \log^* \frac{m}{n})$. We have now established the following.

Theorem 2. *A DGG with m arcs and n terminals can be weakly solved in $O(m + m(\log^* m - \log^* \frac{m}{n}))$ time.* □

Remark 3. When $m = \Omega(n \log^{(k)} n)$ for some constant k, this bound is $O(m)$.

[3] Note, however, that while the technique is similar, the problem of solving deterministic graphical games does not seem to fit into the framework of [11].

Acknowledgements. We are indebted to Mike Paterson for his contributions to this work. We would also like to thank Uri Zwick and Gerth Brodal for helpful discussions.

References

1. M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *Proceedings of the 15th Annual ACM Symposium on the Theory of Computing*, pages 1–9, 1983.
2. Robert J. Aumann. Presidential address at the First International Congress of the Game Theory Society. *Games and Economic Behavior*, 45:2–14, 2003.
3. M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R.E. Tarjan. Linear time bounds for median computations. In *Proceedings of the 4th Annual ACM Symposium on the Theory of Computing*, pages 119–124, 1972.
4. Anne Condon. The complexity of stochastic games. *Information and Computation*, 96:203–224, 1992.
5. E. Allen Emerson, Charanjit S. Jutla, and A. Prasad Sistla. On model checking for the μ-calculus and its fragments. *Theor. Comput. Sci.*, 258(1-2):491–522, 2001.
6. H. Everett. Recursive games. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games Vol. III*, volume 39 of *Annals of Mathematical Studies*. Princeton University Press, 1957.
7. H.N. Gabow and R.E. Tarjan. Algorithms for two bottleneck optimization problems. *J. Algorithms*, 9:411–417, 1988.
8. Ernst A. Heinz. *Scalable Search in Computer Chess: Algorithmic Enhancements and Experiments at High Search Depths*. Morgan Kaufmann Publishers Inc., 1999.
9. M. Kearns, M.L. Littman, and S. Singh. Graphical models for game theory. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 253–260, 2001.
10. D.E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching.* Addison-Wesley, 1997 (3rd edition).
11. A.P. Punnen. A fast algorithm for a class of bottleneck problems. *Computing*, 56:397–401, 1996.
12. J. Romein and H. Bal. Solving the game of awari using parallel retrograde analysis. *IEEE Computer*, 36(10):26–33, 2003.
13. K. Thompson. Retrograde analysis of certain endgames. *Journal of the International Computer Chess Association*, 9(3):131–139, 1986.
14. Nicolas Vieille. Two-player stochastic games I: A reduction. *Israel Journal of Mathematics*, 119:55–91, 2000.
15. Nicolas Vieille. Two-player stochastic games II: The case of recursive games. *Israel Journal of Mathematics*, 119:93–126, 2000.
16. A.R. Washburn. Deterministic graphical games. *Journal of Mathematical Analysis and Applications*, 153:84–96, 1990.
17. Uri Zwick and Mike S. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1–2):343–359, 1996.