Using Bisimulations in Analysis of Stochastic Games

Roland Glück

Institut für Informatik, Universität Augsburg,
D-86135 Augsburg, Germany
glueck@informatik.uni-augsburg.de

Abstract. Stochastic Games are known to be both in the complexity classes P and NP, but no provably efficietn algorithm is known. We present an approach using bisimulations which can lead to a speed-up under certain circumstances.

1 Introduction

1.1 General Ideas

In practice one is often confronted with systems containing a large, even infinite number of states and/or transitions, e.g. in control theory, model checking, internet routing and similar cases. If the task is to ensure a certain property (optimality, safety, liveness) by refining, i.e. removing (in practice preventing) transitions, this task can appear to be difficult to solve for the large system. One possible strategy is to reduce the original system into a smaller one using a suitable bisimulation, then to apply a known algorithm to that system, such that a refined subsystem of it fulfils the demanded property, and in a last step to expand that system into a subsystem of the original one. Of course this strategy will not work in all cases. To make sense, the reduction by bisimulation has to decrease the number of states/transitions in a significant way, an algorithm for computing a refined system with the required property has to be known, and the desired property has to be invariant in a certain sense with respect to the chosen bisimulation. As new material in this paper we show that this approach is correct for a class of stochastic games. In contrast to model checking, where bisimulations are commonly used to *check* properties of a system (cf. e.g. [1]) we will use them to *construct* systems with desired properties.

1.2 Recent Work

In [10] it was shown how a control policy ensuring a certain optimality property in infinite transition systems can be obtained; that approach worked without the use of bisimulations. However, the iteratively constructed sets (called strata) in that method actually correspond to the equivalence classes of a suitable bisimulation. The successor paper [4] gives an algebraic formulation of bisimulation in

general and shows the correctness of the approach for a certain liveness property. The generic algorithm was described in [5]. The most recent paper [3] applies the sketched idea to optimality problems.

1.3 Overview

The paper consists of three parts: In the first two parts we give a short overview over stochastic games and bisimulations, resp. The next section section shows how to use bisimulations in the analysis of stochastic games and discusses the effiency of this approach and further work.

2 Stochastic Games

2.1 Definitions and Basic Properties

Stochastic games were first introduced in [9]. We will consider here a special version of stochastic games, which do not mean a real restriction, according to [2].

Definition 2.1. *A* simple stochastic game (SSG) *is a finite directed graph* $G = (V, E)$ *with* $V = V_{max} \dot\cup V_{min} \dot\cup V_{average} \dot\cup \{sink_0\} \dot\cup \{sink_1\}$. *Every node except the two sink-nodes has an outdegree in* $\{1, 2\}$, *and both* $sink_0$ *and* $sink_1$ *have outdegree zero.*

The sets V_{max}, V_{min} and $V_{average}$ are called *max*, *min* and *average* nodes, resp. For uniformity we assume from now on that $V = \{1, 2, \ldots, n\}$ and $sink_0 = (n-1)$ and $sink_1 = n$. Often we refer to an SSG only as a graph $G = (V, E)$ and assume the partition to be given silently.

An SSG is played by two players, the min-player and the max-player. For a SSG a min-strategy τ is a subset of $V_{min} \times V \cap E$ such that every node in τ has exactly one outgoing edge. Analogously, a max-strategy σ is a subset of $V_{max} \times V \cap E$ with the same property as above. Corresponding to a pair of min- and max-strategies (σ, τ) there is a graph $G_{\sigma,\tau}$ which arises from G by removing all edges not chosen by σ respective τ. From mow on we will assume that every SSG under consideration is *stopping*, i.e. for every pair of min- and maxstrategies (σ, τ) some sink-node is reachable in $G_{\sigma,\tau}$ from every non-sink node. Usually, σ refers to a max- and τ to a min-strategy.

On such a graph $G_{\sigma,\tau}$ a token is placed on a node v and moved along the edges chosen by the min- and max-player according to their strategy. On an average node with exactly one outgoing edge the token is moved along this edge. If the token is on an average node with two outgoing edges one of these edges is chosen randomly with probability $\frac{1}{2}$. The game ends when the token reaches a sink node. In the case of $sink_0$ the min-player wins, in the case of $sink_1$ the max-player wins.

The *value* $val_{\sigma,\tau}(v)$ of a node v of G with respect to the strategy pair (σ, τ) is the probality that the max-player wins if the strategies σ and τ are chosen. The

optimal value $val(v)$ of a node v is defined by $val(v) = \max_{\sigma} \min_{\tau} val_{\sigma,\tau}$. In [9] the equality $\max_{\sigma} \min_{\tau} val_{\sigma,\tau} = \min_{\tau} \max_{\sigma} val_{\sigma,\tau}$ is shown. As a decision problem the *SSG problem* is if the optimal value of a node for given SSG is at least $\frac{1}{2}$. This problem is known to be in $NP \cap coNP$. However, given a min-strategy σ the computation of an optimal associated max-strategy can be done in polynomial time using linear programming.

An example for an SSG is given in Figure 1 where $V_{max} = \{max_1, max_2\}$, $V_{min} = \{min_1, min_2\}$ and $V_{average} = \{avg\}$; the sinks are labelled canonically.

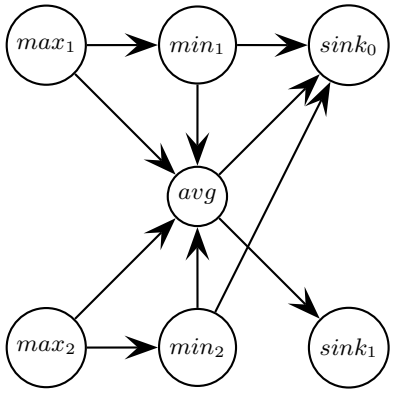

Fig. 1. A Simple Stochastic Game

A possible pair of strategies (σ, τ) can given by $\sigma = \{(max_1, min_1), (max_2, avg)\}$ and $\tau = \{(min_1, sink_0), (min_2, sink_0)\}$. For this pair of strategies we have $val_{\sigma,\tau}(max_1) = 0$, $val_{\sigma,\tau}(max_2) = \frac{1}{2}$, $val_{\sigma,\tau}(min_1) = val_{\sigma,\tau}(min_2) = 0$, $val_{\sigma,\tau}(avg) = \frac{1}{2}$, $val_{\sigma,\tau}(sink_0) = 0$ and $val_{\sigma,\tau}(sink_1) = 1$. Obviously, this is not the optimal strategy for the max player, who should switch his choice at max_1. The resulting strategy $\{(max_1, avg), (max_2, avg)\}$ is optimal (which is easy to verify, moreover, the min player can not do better), and the optimal value function is given by $val(max_1) = val(max_2) = \frac{1}{2}$, $val(min_1) = val(min_2) = 0$, $val(avg) = \frac{1}{2}$, $val_{\sigma,\tau}(sink_0) = 0$ and $val_{\sigma,\tau}(sink_1) = 1$.

2.2 Analysis of SSG's

For a given pair of strategies (σ, τ) the values $val_{\sigma,\tau}(v)$ can be easily computed using facts from the theory of Markov chains. Therefore, let $\overline{val}_{\sigma,\tau}$ be the vector $\overline{val}_{\sigma,\tau} = (val_{\sigma,\tau}(1), \ldots, val_{\sigma,\tau}(n-2))$. The matrix $Q \in \{0, \frac{1}{2}, 1\}^{(n-2)\times(n-2)}$ with entries q_{ij} has at position (i, j) the probability of reaching node j from node i in one single step in G_{sigtau}. In particular, for a max node i the entry q_{ij} is one iff there is the edge (i, j) in $G_{\sigma,\tau}$, and zero otherwise (analogously for min nodes). If an averages node i has exactly one outgoing edge (i, j) than we have $q_{ij=1}$ and $q_{ij'} = 0$ for all $j \neq j'$. Similarly, for an average node i with two outgoing edges (i, j_1) and (i, j_2) we have $q_{ij_1} = q_{ij_2} = \frac{1}{2}$ and $q_{ij} = 0$ for all

$j \neq j_1, j_2$. Furthermore, let $\overline{b}$ be a vector with $n-2$ entries, whose i-th entry is the probability of reaching $sink_1$ from node i in exactly one step. Then we have the following lemma:

Lemma 2.2. *$\overline{val}_{\sigma,\tau}$ is the unique solution of the fixpoint equation $\overline{val}_{\sigma,\tau} = Q\,\overline{val}_{\sigma,\tau} + \overline{b}$.*

Obviously, the computation of $\overline{val}_{\sigma,\tau}$ can be done in polynomial time.

In contrast, for the computation of the optimal value no provably polynomial algorithm is known. The optimal values $val(i)$ for SSG's can be shown to be the unique solution of the following system of constraints:

$$
\begin{array}{ll}
val(i)=\max(val(j), val(k)), & \text{if } i \in V_{max} \text{ with two successors } j \text{ and } k \\
val(i) = val(j), & \text{if } i \in V_{max} \text{ with exactly one successor } j \\
val(i)=\min(val(j), val(k)), & \text{if } i \in V_{min} \text{ with two successors } j \text{ and } k \\
val(i) = val(j), & \text{if } i \in V_{min} \text{ with exactly one successor } j \\
val(i) = \frac{1}{2}(val(j) + val(k)), & \text{if } i \in V_{average} \text{ with two successors } j \text{ and } k \\
val(i) = val(j), & \text{if } i \in V_{average} \text{ with exactly one successor } j \\
val(sink_0) = 0 & \\
val(sink_1) = 1 &
\end{array}
$$

Given the optimal values for every node an optimal max strategy can be easily determined if only the edges leading into nodes with the greater value among the successors of a node are kept (if there are two successors with the same value one edge can be kept arbitrarily). Analogously an optimal min strategy can be computed from the optimal values.

3 Bisimulations

The SSG's from the previous section can have a large numbers of nodes. One possibility to reduce the problem is the use of bisimulations, which will be introduced in this section. Before getting formal we fix some notation.

For a relation R we denote its converse by R°. For relational composition we use the semicolon $;$. If $E \subseteq X \times X$ is an equivalence relation we write x/E for the equivalence class of an element $x \in X$. For a subset $X' \subseteq X$ we define X'/E by $X'/E = \bigcup_{x \in X'} x/E$.

3.1 Basic Definitions

Definition 3.1. *A* bisimulation *between two relations $R \subseteq X \times X$ and $R' \subseteq X' \times X'$ is a relation $B \subseteq X \times X'$ with the properties $B^\circ; R \subseteq R'; B^\circ$ and $B; R' \subseteq R; B$.*

Intuitively this means that if a step from x to y is possible under the relation R then a step from x' to y' is possible under R' where x and x' respectively y and y' are related via B. The analogous property holds for transitions in R' compared to those in R; here the elements are related by B°.

A bisimulation between a relation $R \subseteq X \times X$ and itself is called an *autobisimulation*. Since autobisimulations are closed under union, composition and conversion and the identity is an autobisimulation there is a coarsest autobisimulation for a relation $R \subseteq X \times X$, which is an equivalence. An autobisimulation, which is also an equivalence is called an *autobisimulation equivalence*.

Here we are interested in a special kind of autobisimulation equivalences, which also respects a given partition of the node set of a graph. Therefore we define:

Definition 3.2. *An autobisimulation equivalence B on a relation $R \subseteq X \times X$* respects *the partition $X = \dot{\bigcup}_{i \in I} X_i$ if for every $i \in I$ the set X_i can be written as the union of suitable equivalence classes of B.*

This means that B relates only elements of the same sets X_i to each other. Also here, the set of autobisimulation equivalences respecting a given partition is closed under composition, union and taking the converse, and since the identity is an autobisimulation equivalence respecting every partition there is also a unique coarsest autobisimulation equivalence respecting a given partition. In [8] it is shown that this coarsest autobisimulation equivalence can be computed in $O(log|R| \cdot |X|)$ (or $O(log|E| \cdot |V|)$ in terms of a graph $G = (V, E)$).

3.2 Quotient and Expansion

For our purposes bisimulation equivalences can be used to reduce the state numbers of stochastic games in a reasonable way. This is done via the *quotient graph*:

Definition 3.3. *Let $G = (V, E)$ a stochastic game with the usual partition $V = V_{max} \dot\cup V_{min} \dot\cup V_{average} \dot\cup \{sink_0\} \dot\cup \{sink_1\}$ and B an autobisimulation equivalence for G respecting this partition. The* quotient *G/B of G by B is defined as the stochastic game $G_B = (V_B, E_B)$ with*

- $V_B = \{v/B \,|\, v \in V\}$
- $E_B = \{(x/B, y/B \,|\, (x, y) \in E)\}$

The partition of V_B into max, min and average nodes is given by $V_{B_{max}} = V_{max}/B$, $V_{B_{min}} = V_{min}/B$ and $V_{B_{average}} = V_{average}/B$. The sinks in G/B are $sink_0/B$ and $sink_1/B$.

This construction is analogous to the one of a minimal automaton in automata theory, see for example the classics [6] and [7].

Note that this construction is well defined: First, the sets $V_{B_{max}}$, $V_{B_{min}}$, $V_{B_{average}}$, $\{sink_0\}$ and $sink_1$ are disjoint, since the corresponding sets in G are disjoint and B respects this partition of V. Second, according to the construction of E_B both $sink_0/B$ and $sink_1$ have no outgoing edges in E_B. For analogous reasons every other node in V_B has at least one and at most two outgoing edges in E_B. Note that a node $v \in V$ can have two outgoing edges in E whereas the node v/B can have exactly one outgoing edge in E_B.

The coarsest bisimulation equivalence of the SSG from Figure 1 induces the equivalence classes $\{max_1, max_2\}$, $\{min_1, min_2\}$, $\{avg\}$, $\{sink_0\}$ and $\{sink_1\}$. So the coarsest quotient is given by the SSG from Figure 2, where the nodes caption is selfexplaining.

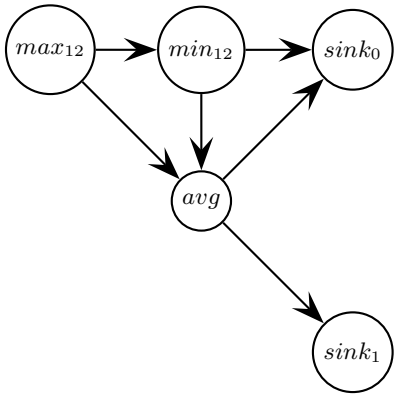

Fig. 2. A Coarsest Quotient

If we want to reduce the number of states of a stochastic game in a suitable manner we use the coarsest autobisimulation equivalence to build the quotient, because it reduces the number of nodes maximally among all autobisimulation equivalences. In this case the resulting quotient is called the *coarsest quotient*.

Generally, the coarsest quotient of a stochastic game will have a smaller number of nodes than the original one, especially it is well structured (i.e., it contains identic subgraphs, which can be merged by the coarsest quotient).

4 Putting the Pieces together

To make use of the quotient operation in SSG's we have to show how to construct an optimal strategy via the quotient. The key idea is that the optimal values in the quotient correspond to the optimal values in the original SSG. This is expressed in the following lemma:

Lemma 4.1. *Let $G = (V, E)$ be an SSG, B an autobisimulation equivalence for G and $val_B : V/B \to [0, 1]$ the function of the optimal values of G/B. Then the function $val' : V \to [0, 1]$, defined by $val'(v) = val_B(v/B)$, is the optimal value function of G.*

Proof. The idea of the proof is to show that the function val' fulfilles the constraints from 2.2 for all nodes $v \in V$. For an arbitrary node $v \in V$ we distinguish the following cases:

(a) v/B is an average node with one outgoing edge and v has two outgoing edges.

(b) v/B is an average node with one outgoing edge and v has one outgoing edge.
(c) v/B is an average node with two outgoing edges.
(d) v/B is a min (max) node with one outgoing edge and v has two outgoing edges.
(e) v/B is a min (max) node with one outgoing edge and v has one outgoing edge.
(f) v/B is a min (max) node with two outgoing edges and v has two outgoing edges.
(g) v/B is a sink node.

First, assume case (a), and let w/B be the successor of v/B. Then v has exactly two successors w_1 and w_2. According to the construction of val' we have $val'(v) = val_B(v/B)$ and $val'(w_1) = val'(w_2) = val_B(w/B)$. Because val_B is the optimal value function on G/B we have $val_B(v/B) = val_B(w/B)$. So the constraint $val'(v) = \frac{1}{2}(val'(w_1) + val'(w_2))$ is fulfilled.
Case (b) is trivial, since the values of v/B and v and their successors simply coincide.
In case (c) let w_1/B and w_2/B be the two successors of v/B. Then v has also two successors w' and w'' with $w' \in w_1/B$ and $w'' \in w_2/B$. Because of $val_B(v/B) = \frac{1}{2}(val_B(w_1/B) + val_B(w_2)/B)$ we have according to the construction of val' the equality $val'(v) = \frac{1}{2}(val'(w') + val'(w''))$.
In the cases (d)-(g) we consider only min nodes, the proof for max nodes is done analogously. So in case (d) let w/B be the successor of v/B and w' and w'' the two distinct successors of v (note that $\{w', w''\} \subseteq w/B$). Because val_B is the optimal value function we have $val_B(v/B) = val_B(w/B)$. By construction of val' we have $val'(v) = val_B(v)$ and $val'(w') = val'(w'') = val_B(w/B)$, so the constraint $val'(v) = \min\{val'(w_1), val'(w_2)\}$.
Case (e) is as trivial as case (b).
In case (f) let w_1/B and w_2/B be the two successors of v/B. Then v has also two successors w' and w'' with $w' \in w_1/B$ and $w'' \in w_2/B$. Because of $val_B(v/B) = \min\{val_B(w_1/B), val_B(w_2)/B\}$ we have according to the construction of val' the equality $val'(v) = \min\{val'(w'), val'(w'')\}$.
The sink nodes from case (g) are assigned the right values due to the requirements on B. □

So because from the optimal value function an optimal strategy can easily deduced we have the following algorithm for computing an optimal strategy for a SSG G:

1. Construct the coarsest quotient G/B
2. Compute the optimal value val_B function of G/B
3. Expand the val_B to the optimal value function val of G
4. Construct an optimal strategy from val

Here the third and fourth step can easily be done in linear time in $|V| + |E|$.

Let us demonstrate this approach on our example from Figure 1. After the first step we obtain the coarsest quotient from Figure 2. The computation

of the optimal value function val_B on the quotient yields $val_B(max_{12}) = \frac{1}{2}$, $val_B(min_{12}) = 0$, $val_B(avg) = \frac{1}{2}$, $val_B(sink_0) = 0$ and $val_B(sink_1) = 1$. So the optimal value function val of the initial SSG is given by $val(max_1) = val(max_2) = \frac{1}{2}$, $val(min_1) = val(min_2) = 0$, $val(avg) = \frac{1}{2}$, $val_B(sink_0) = 0$ and $val_B(sink_1) = 1$. Now a pair of optimal strategies can easily be determined.

5 Conclusion

Given an algorithm which computes the optimal value function of an SSG $G = (V, E)$ in time $O(f(|V|, |E|))$ our algorithm computes the optimal value function in time $O(log|E| \cdot |V| + f(|V/B|, |E/B|))$ where $(V/B, E/B)$ is the coarsest quotient of G. The runtime advantage (is there is one) heavily depends on how the coarsest simplifies the original SSG. In certain cases, for example if G has a high degree of symmetry or redundancy, our algorithm will lead to a speed up compared to the immediate application of the algorithm for computing an optimal strategy.

References

1. C. Baier and J-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
2. A. Condon. The complexity of stochastic games. In *Information and Computation*, volume 96, pages 203–224, 1992.
3. R. Glück. Using bisimulations for optimality problems in model refinement. In R. Berghammer and B. Möller, editors, *12th International Conference on Relational and Algebraic Methods in Computer Science — RAMICS 2011*, volume 6663 of *Lecture Notes in Computer Science*, pages 164–179. Springer, 2011.
4. R. Glück, B. Möller, and M. Sintzoff. A semiring approach to equivalences, bisimulations and control. In R. Berghammer, A.M. Jaoua, and B. Möller, editors, *11th International Conference on Relational Methods in Computer Science — RelMiCS 2009*, volume 5827 of *Lecture Notes in Computer Science*, pages 134–149. Springer, 2009.
5. R. Glück, B. Möller, and M. Sintzoff. Model refinement using bisimulation quotients. In M. Johnson and D. Pavlovic, editors, *13th International Conference on Algebraic Methodology And Software Technology — AMAST 2010*, volume 6486 of *Lecture Notes in Computer Science*, pages 76–91. Springer, 2011.
6. J. Myhill. Finite automata and the representation of events. *WADD TR-57-624*, pages 112–137, 1957.
7. A. Nerode. Linear automaton transformations. volume 9 of *Proceedings of the American Mathematical Society*, pages 541–544, 1958.
8. R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal for Computing*, 16(6).
9. L.S. Shapley. Stochastic games. In *Proceedings of the National Academy of Sciences*, volume 39, pages 1095–1100, 1953.
10. M. Sintzoff. Synthesis of optimal control policies for some infinite-state transition systems. In P. Audebaud and C. Paulin-Mohring, editors, *Mathematics of Program Construction — MPC 2008*, volume 5133 of *Lecture Notes in Computer Science*, pages 336–359. Springer, 2008.