A Fast Bisimulation Algorithm

Agostino Dovier[1], Carla Piazza[2], and Alberto Policriti[2]

[1] Dip. di Informatica, Univ. di Verona
Strada Le Grazie 15, 37134 Verona, Italy
dovier@sci.univr.it
[2] Dip. di Matematica e Informatica, Univ. di Udine
Via Le Scienze 206, 33100 Udine, Italy
{piazza,policriti}@dimi.uniud.it

Abstract. In this paper we propose an efficient algorithmic solution to the problem of determining a *Bisimulation* Relation on a finite structure. Starting from a set-theoretic point of view we propose an algorithm that optimizes the solution to the *Relational coarsest Partition problem* given by Paige and Tarjan in 1987 and its use in model-checking packages is briefly discussed and tested. Our algorithm reaches, in particular cases, a linear solution.
Keywords: Bisimulation, non well-founded sets, automata, verification.

1 Introduction

It is difficult to accurately list all the fields in which, in one form or another, the notion of *bisimulation* was introduced and now plays a central rôle. Among the most important ones are: Modal Logic, Concurrency Theory, Formal Verification, and Set Theory.

Several existing verification tools make use of bisimulation in order to minimize the state spaces of systems description. The reduction of the number of states is important both in compositional and in non-compositional model checking. Bisimulation serves also as a means of checking equivalence between transition systems. The verification environment XEVE [5] provides bisimulation tools which can be used for both minimization and equivalence test. In general, in the case of explicit-state representation, the underlying algorithm used is the one proposed by Kanellakis and Smolka [17], while Bouali and de Simone algorithm [6] is used in the case of symbolic representation. The Concurrency Factory project [8] tests bisimulation using techniques based on the Kanellakis and Smolka algorithm. As for the criticism on the use of bisimulation algorithms, Fisler and Vardi observe in [10] that "bisimulation minimization does not appear to be viable in the context of invariance verification", but in the context of compositional verification it "makes certain problems tractable that would not be so without minimization" [2,21].

The first significant result related to the algorithmic solution of the bisimulation problem is in [16], where Hopcroft presents an algorithm for the minimization of the number of states in a given finite state automaton. The problem is equivalent to that of determining the coarsest partition of a set *stable* with

G. Berry, H. Comon, and A. Finkel (Eds.): CAV 2001, LNCS 2102, pp. 79–90, 2001.

respect to a finite set of functions. A variant of this problem is studied in [20], where it is shown how to solve it in linear time in case of a single function. Finally, in [19] Paige and Tarjan solved the problem for the general case (i.e., bisimulation) in which the stability requirement is relative to a relation E (on a set N) with an algorithm whose complexity is $O(|E| \log |N|)$.

The main feature of the linear solution to the single function coarsest partition problem (cf. [20]), is the use of a *positive* strategy in the search for the coarsest partition: the starting partition is the partition with singleton classes and the output is built via a sequence of steps in which two or more classes are merged. Instead, Hopcroft's solution to the (more difficult) many functions coarsest partition problem is based on a (somehow more natural) *negative* strategy: the starting partition is the input partition and each step consists of the split of all those classes for which the stability constraint is not satisfied. The interesting feature of Hopcroft's algorithm lies in its use of a clever ordering (the so-called "process the smallest half" ordering) for processing classes that must be used in a split step. Starting from an adaptation of Hopcroft's idea to the relational coarsest partition problem, Paige and Tarjan succeeded in obtaining their fast solution [19]. The algorithm presented in [4] is based on the naïve negative strategy, but on each iteration it stabilizes only reachable blocks with respect to all blocks. This is improved in [18], where only reachable blocks are stabilized with respect to reachable blocks only.

In this paper we present a procedure that integrates positive and negative strategies to obtain the algorithmic solution to the bisimulation problem and hence to the relational coarsest partition problem. The strategy we develop is driven by the set-theoretic notion of *rank* of a set. The algorithm we propose uses [20] and [19] as subroutines and terminates in linear time in many cases, for example when the input problem corresponds to a bisimulation problem on acyclic graphs (well-founded sets). It operates in linear time in other cases as well and, in any case, it runs at a complexity less than or equal to that of the algorithm by Paige and Tarjan [19]. Moreover, the partition imposed by the rank allows to process the input without storing the entire structure in memory at the same time.

The paper is organized as follows: in the next section we introduce the set-theoretic formulation of the bisimulation problem. The subsequent Section 3 contains the algorithm for the well-founded case. Section 4 presents the basic idea of our proposed algorithm and its optimizations are explained in the following section. In Section 6 we show how our results and methods can be adapted to the *multi-relational* coarsest partition problem (i.e., bisimualtion on labeled graphs) and in Section 7 we discuss some testing results. Some conclusions are drawn in Section 8. Detailed proofs of all the statements in this paper can be found in [9].

2 The Problem: A Set-Theoretic Perspective

One of the main features of intuitive (naïve) Set Theory is the well-foundedness of membership. As a consequence, standard axiomatic set theories include the *foundation* axiom that forces the membership relation to form no cycles or infinite descending chains. In the 80's the necessity to consider theories that do not

assume this strong constraint (re-)emerged in many communities; hence various proposals for (axiomatic) non well-founded set theories (and universes) were developed (see [11,1,3]).

Sets can be seen as nothing but *accessible pointed graphs* (cf. Definition 1). Edges represent membership, $m \to n$ means that m has n as an element, and the nodes in the graph denote all the sets which contribute in the construction of the represented set.

Definition 1. *An* accessible pointed graph (apg) $\langle G, n\rangle$ *is a directed graph* $G = \langle N, E\rangle$ *together with a distinguished node* $n \in N$ *such that all the nodes in* N *are reachable from* n.

The resulting set-theoretic semantics for apg's, introduced and developed in [1], is based on the natural notion of *picture* of an apg. The extensionality axiom—saying that two objects are equal if and only if they contain exactly the same elements—is the standard criterion for establishing equality between sets. If extensionality is assumed it is immediate to see that, for example, different acyclic graphs can represent the same set. However, extensionality leads to a cyclic argument (no wonder!) whenever one tries to apply it as a test to establish whether two cyclic graphs represent the same non well-founded set (*hyperset*). To this end a condition (*bisimulation*) on apg's can be stated in accordance with extensionality: two apg's are bisimilar if and only if they are representations of the same set.

Definition 2. *Given two graphs* $G_1 = \langle N_1, E_1\rangle$ *and* $G_2 = \langle N_2, E_2\rangle$, *a* bisimulation *between* G_1 *and* G_2 *is a relation* $b \subseteq N_1 \times N_2$ *such that:*

1. $u_1\, b\, u_2 \wedge \langle u_1, v_1\rangle \in E_1 \Rightarrow \exists v_2 \in N_2 (v_1\, b\, v_2 \wedge \langle u_2, v_2\rangle \in E_2)$
2. $u_1\, b\, u_2 \wedge \langle u_2, v_2\rangle \in E_2 \Rightarrow \exists v_1 \in N_1 (v_1\, b\, v_2 \wedge \langle u_1, v_1\rangle \in E_1)$.

Two apg*'s* $\langle G_1, n_1\rangle$ *and* $\langle G_2, n_2\rangle$ *are bisimilar if and only if there exists a bisimulation* b *between* G_1 *and* G_2 *such that* $n_1\, b\, n_2$.

We can now say that two hypersets are equal if their representations are bisimilar. For example the apg $\langle\langle\{n\}, \emptyset, \rangle, n\rangle$ represents the empty set $\emptyset$. The hyperset Ω, i.e. the unique hyperset which satisfies the equation $x = \{x\}$ (see [1]), can be represented using the apg $\langle\langle\{n\}, \{\langle n, n\rangle\}\rangle, n\rangle$. Any graph such that each node has at least one outgoing edge can be shown to be a representation of Ω. It is clear that for each set there exists a collection of apg's which are all its representations. It is always the notion of bisimulation which allows us to find a *minimum* representation (there are no two nodes representing the same hyperset). Given an apg $\langle G, n\rangle$ that represents a set S, to find the minimum representation for S it is sufficient to consider the maximum bisimulation $\equiv$ between G and G. Such a bisimulation $\equiv$ always exists and is an equivalence relation over the set of nodes of G. The minimum representation of S is the apg $\langle G/\equiv, [n]\rangle$ (see [1]) which is usually called *bisimulation contraction of* G.

An equivalent way to present the problem is to define the concept of bisimulation as follows.

Definition 3. *Given a graph $G = \langle N, E\rangle$, a* bisimulation *on G is a relation $b \subseteq N \times N$ such that:*

1. $u_1\, b\, u_2 \wedge \langle u_1, v_1\rangle \in E \Rightarrow \exists v_2(v_1\, b\, v_2 \wedge \langle u_2, v_2\rangle \in E)$
2. $u_1\, b\, u_2 \wedge \langle u_2, v_2\rangle \in E \Rightarrow \exists v_1(v_1\, b\, v_2 \wedge \langle u_1, v_1\rangle \in E)$.

A bisimulation on G is nothing but a bisimulation between G and G. The problem of recognizing if two graphs are bisimilar and the problem of determining the maximum bisimulation on a graph are equivalent. Two disjoint apg's $\langle\langle N_1, E_1\rangle, \nu_1\rangle$ and $\langle\langle N_2, E_2\rangle, \nu_2\rangle$ are bisimilar if and only if $\nu_1 \equiv \nu_2$, where $\equiv$ is the maximal bisimulation on $\langle\langle N_1 \cup N_2 \cup \{\mu\}, E_1 \cup E_2 \cup \{\langle\mu, \nu_1\rangle, \langle\mu, \nu_2\rangle\}\rangle, \mu\rangle$, with μ a new node. We consider the problem of finding the minimum graph bisimilar to a given graph, that is, the *bisimulation contraction* of a graph.

The notion of bisimulation can be connected to the notion of *stability*:

Definition 4. *Let E be a relation on the set N, E^{-1} its inverse relation, and P a partition of N. P is said to be* stable *with respect to E iff for each pair B_1, B_2 of blocks of P, either $B_1 \subseteq E^{-1}(B_2)$ or $B_1 \cap E^{-1}(B_2) = \emptyset$.*

Given a set N, k relations $E_1, \ldots, E_k$ on N, and a partition P of N, the *multi-relational coarsest partition* problem consists of finding the coarsest refinement of P which is stable with respect to $E_1, \ldots, E_k$. As noted in [17], the algorithm of [19] that determines the coarsest partition of a set N stable with respect to k relations solves exactly the problem of *testing if two states of an observable Finite States Process (FSP)* are strongly equivalent. Our bisimulation problem is a particular case of *observable FSPs strong equivalence* problem ($k = 1$). In Section 6 we show how the case of bisimulation over a labeled graph (*multi-relational* case) can be linearly reduced to our bisimulation problem. This means that the problem of finding the bisimulation contraction of a graph is equivalent to the multi-relational coarsest partition problem.

3 The Well-Founded Case

We start by considering the case of acyclic graphs (well-founded sets). Similarly to what is done in the minimization of Deterministic Finite Automata, it is possible to to determine the coarsest partition P stable w.r.t. E through the computation of a greatest fixpoint. A "negative" (and blind with respect to the relation) strategy is applicable: start with the coarsest partition $P = \{N\}$, choose a class B (the splitter) and split all the classes using B whenever P is *not* stable. The complexity of the algorithm, based on a negative strategy, presented in [19] for this problem is $O(|E| \log |N|)$.

We will take advantage of the set-theoretic point of view of the problem in order to develop a selection strategy for the splitters depending on the relation E. Making use of the ordering induced by the notion of *rank* we will start from a partition which is a refinement of the coarsest one; then we will choose the splitters using the ordering induced by the rank. These two ingredients allow to obtain a linear-time algorithm.

Definition 5. *Let $G = \langle N, E\rangle$ be a directed acyclic graph. The* rank *of a node n is recursively defined as follows:*

$$\begin{cases} rank(n) = 0 & \text{if } n \text{ is a leaf} \\ rank(n) = 1 + \max\{rank(m) : \langle n, m\rangle \in E\} & \text{otherwise} \end{cases}$$

The notion of *rank* determines a partition which is coarser than the maximum bisimulation.

Proposition 1. *Let m and n be nodes of an acyclic graph G. If $m \equiv n$, then $rank(m) = rank(n)$.*

The converse, of course, is not true. Let P be a partition of N such that for each block B in P it holds that $m, n \in B$ implies $rank(m) = rank(n)$; then every refinement of P fulfills the same property. Hence, we can assign to a block B the rank of its elements.

Algorithm 1 (Well-Founded Case).

1. **for** $n \in N$ **do compute** $rank(n)$; — compute the ranks
2. $\rho := \max\{rank(n) : n \in N\}$;
3. **for** $i = 0, \ldots, \rho$ **do** $B_i := \{n \in N : rank(n) = i\}$;
4. $P := \{B_i : i = 0, \ldots, \rho\}$; — P is the partition to be refined initialized with the B_i's
5. **for** $i = 0, \ldots, \rho$ **do**
   (a) $D_i := \{X \in P : X \subseteq B_i\}$; — determine the blocks currently at rank i
   (b) **for** $X \in D_i$ **do**
   $G :=$ **collapse**(G, X); — collapse nodes at rank i
   (c) **for** $n \in N \cap B_i$ **do** — refine blocks at higher ranks
   for $C \in P$ **and** $C \subseteq B_{i+1} \cup \ldots \cup B_\rho$ **do**
   $P := (P \setminus \{C\}) \cup \{\{m \in C : \langle m, n\rangle \in E\}, \{m \in C : \langle m, n\rangle \notin E\}\}$;

Step 1 can be performed in time $O(|N| + |E|)$ by a depth-first visit of the graph. Collapsing nodes $a_1, \ldots, a_k$, as in step $5(b)$, consists in eliminating all nodes but a_1 and replacing all edges incident to $a_2, \ldots, a_k$ by edges incident to a_1. Despite the nesting of **for**-loops the following holds.

Proposition 2. *The algorithm for the well-founded case correctly computes the bisimulation contraction of its input acyclic graph $G = \langle N, E\rangle$ and can be implemented so as to run in linear time $O(|N| + |E|)$.*

An example of computation of the above algorithm can be seen in Figure 1. In

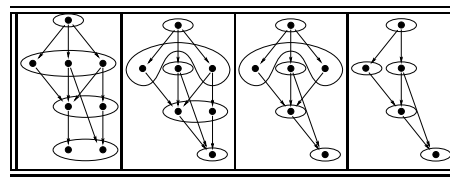

Fig. 1. Minimization Process.

all the examples we present, the computation steps proceed from left to right.

Those who are familiar with OBDDs ([7]) or with k-layered DFA's ([15]) can read our algorithm for the well-founded case as a generalization of the minimization algorithm for k-layered DFA. In the well-founded case we admit that a node at the i-th layer may reach a node at the j-th layer with $j > i$.

4 Basic Idea for the General Case

The presence of cycles causes the usual notion of rank (cf. Definition 5) to be not adequate: an extension of such a notion must be defined.

Definition 6. *Given a graph $G = \langle N, E\rangle$, let $G^{scc} = \langle N^{scc}, E^{scc}\rangle$ be the graph obtained as follows:*

$$N^{scc} = \{c : c \text{ is a strongly connected component in } G\}$$
$$E^{scc} = \{\langle c_1, c_2\rangle : c_1 \neq c_2 \text{ and } \exists n_1 \in c_1, n_2 \in c_2(\langle n_1, n_2\rangle \in E)\}$$

Given a node $n \in N$, we refer to the node of G^{scc} associated to the strongly connected component of n as $c(n)$.

Observe that G^{scc} is acyclic and if G is acyclic then G^{scc} is G itself.

We need to distinguish between the well-founded part and the non-well-founded part of a graph G.

Definition 7. *Let $G = \langle N, E\rangle$ and $n \in N$. $G(n) = \langle N(n), E \restriction N(n)\rangle$ is the subgraph of G of the nodes reachable from n. $WF(G)$, the well-founded part of G, is $WF(G) = \{n \in N : G(n) \text{ is acyclic}\}$.*

Observe that $\langle G(n), n\rangle$ is an apg; if $n \in WF(G)$ then it denotes a well-founded set.

Definition 8. *Let $G = \langle N, E\rangle$. The* rank *of a node n of G is defined as:*

$$\begin{cases} rank(n) = 0 & \text{if } n \text{ is a leaf in } G \\ rank(n) = -\infty & \text{if } c(n) \text{ is a leaf in } G^{scc} \text{ and } n \text{ is not a leaf in } G \\ rank(n) = \max(\{1 + rank(m) : \langle c(n), c(m)\rangle \in E^{scc}, m \in WF(G)\} \cup & \\ \qquad \{rank(m) : \langle c(n), c(m)\rangle \in E^{scc}, m \notin WF(G)\}) & \text{otherwise} \end{cases}$$

Since G^{scc} is always acyclic, the definition is correctly given. If G is acyclic then $G = G^{scc}$ and the above definition reduces to the one given in the well-founded case (Def. 5). Nodes that are mapped into leaves of G^{scc} are either bisimilar to $\emptyset$ or to the hyperset Ω. For a non-well-founded node different from Ω the rank is 1 plus the maximum rank of a well-founded node reachable from it (i.e., a well-founded set in its transitive closure).

We have explicitly used the graph G^{scc} to provide a formal definition of the notion of rank. However, the rank can be computed directly on G by two visits of the graph, avoiding the explicit construction of G^{scc}.

Proposition 3. *Let m and n be nodes of a graph G:*

1. *$m \equiv \Omega$ if and only if $rank(m) = -\infty$;*
2. *$m \equiv n$ implies $rank(m) = rank(n)$.*

The converse of Proposition 3.2 is not true. Moreover, the rank of $c(n)$ in G^{scc} (that can be computed using Def. 5) is not necessarily equal to the rank of n in G.

Given a graph $G = \langle N, E\rangle$ with $\rho = \max\{rank(n) : n \in N\}$, we call the sets of nodes $B_{-\infty}, B_0, \ldots, B_\rho$, where $B_i = \{n \in N : rank(n) = i\}$, the *rank components* of G.

Since we proved in the previous section that the bisimulation contraction can be computed in linear time on well-founded graphs, it is easy to see that we can use the algorithm for the well-founded case in order to process the nodes in $WF(G)$ for the general case. Hence, we can assume that the input graph for the general case does not contain two different bisimilar well-founded nodes.

Algorithm 2 (General Case).

1. **for** $n \in N$ **do compute** $rank(n)$; — compute the ranks
2. $\rho := \max\{rank(n) : n \in N\}$;
3. **for** $i = -\infty, 0, \ldots, \rho$ **do** $B_i := \{n \in N : rank(n) = i\}$;
4. $P := \{B_i : i = -\infty, 0, \ldots, \rho\}$; — P partition to be refined initialized with the B_i's
5. $G :=$ **collapse**$(G, B_{-\infty})$; — collapse all the nodes of rank $-\infty$
6. **for** $n \in N \cap B_{-\infty}$ **do** — refine blocks at higher ranks
   for $C \in P$ **and** $C \neq B_{-\infty}$ **do**
   $P := (P \setminus \{C\}) \cup \{\{m \in C : \langle m, n\rangle \in E\}, \{m \in C : \langle m, n\rangle \notin E\}\}$;
7. **for** $i = 0, \ldots, \rho$ **do**
   (a) $D_i := \{X \in P : X \subseteq B_i\}$; — determine the blocks currently at rank i
   $G_i := \langle B_i, E \restriction B_i\rangle$; — isolate the subgraph of rank i
   $D_i :=$ **Paige-Tarjan**(G_i, D_i); — process rank i
   (b) **for** $X \in D_i$ **do**
   $G :=$ **collapse**(G, X); — collapse nodes at rank i
   (c) **for** $n \in N \cap B_i$ **do** — refine blocks at higher ranks
   for $C \in P$ **and** $C \subseteq B_{i+1} \cup \ldots \cup B_\rho$ **do**
   $P := (P \setminus \{C\}) \cup \{\{m \in C : \langle m, n\rangle \in E\}, \{m \in C : \langle m, n\rangle \notin E\}\}$;

In steps 1–4 we determine the ranks and we initialize a variable P representing the computed partition using the ranks. The collapse operation (steps 5 and 7(b)) is as in the well-founded case. Splits of higher rank blocks is instead done in steps 6 and 7(c). Step 7 is the core of the algorithm, where optimizations will take place. For each rank i we call the procedure of [19] on $G_i = \langle B_i, E \restriction B_i\rangle$, with a cost $O(|E \restriction B_i| \log |B_i|)$ and we update the partition P on nodes of rank greater than i. From these observations:

Proposition 4. *If $G = \langle N, E\rangle$ is a graph, then the worst case complexity of the above algorithm is $O(|E| \log |N|)$. The algorithm for the general case on input G correctly computes the bisimulation contraction of G.*

Proof. (Sketch) The global cost is no worse than (for some $c_1, c_2 \in \mathbb{N}$):

$$c_1(|N| + |E|) + \sum_{i=1}^{\rho} c_2(|E \restriction B_i| \log |B_i|) = O(|E| \log |N|). \quad (1)$$

The complexity of the method sketched above is asymptotically equivalent to that of Paige and Tarjan. However, as for the well-founded Algorithm 1, we take advantage of a refined initial partition and of a selection strategy of the blocks to be setected for splitting blocks of higher ranks. In a single rank, the negative strategy of the Paige-Tarjan algorithm is applied to the rank components which, in general, are much smaller than the global graph. In particular, for families of graphs such that ρ is $\Theta(|N|)$ and the size of the each rank component is bounded by a constant c the global cost becomes linear (cf. formula (1)).

5 Optimizations in the General Case

We present here two situations in which we are able to optimize our algorithm. In some cases, a linear running time is reached. Other possible optimizations are presented in [9].

First Optimization. This optimization makes use of the **Paige-Tarjan-Bonic** procedure [20]. Such a procedure can be used in some cases to solve the coarsest partition problem in linear time adopting a "positive" strategy. Its integration in our algorithm produces a global strategy that can therefore be considered as a mixing of positive and negative strategies.

Definition 9. *A node n belonging to a rank component $B_i \subseteq N$ is said to be a* multiple node *if $|\{m \in B_i : \langle n, m\rangle \in E\}| > 1$.*

Whenever B_i has no multiple nodes, we can replace the call to **Paige-Tarjan** in step $7(a)$ with a call to **Paige-Tarjan-Bonic**. This allows us to obtain a *linear time* performance at rank i (in the formula (1) the term $c_2(|E \restriction B_i| \log |B_i|)$ can be replaced by $c_3(|E \restriction B_i| + |B_i|)$ for some $c_3 \in \mathbb{N}$).

Proposition 5. *The optimized algorithm for the general case on input G correctly computes the bisimulation contraction of G. If $G = \langle N, E\rangle$ is a graph with no multiple nodes, then its worst case complexity is $O(|N| + |E|)$.*

In Figure 2 we show an example of a graph on which the above optimization can be performed and the overall algorithm turns out to be linear.

Second Optimization. The crucial consideration behind the second optimization we propose is the following: the outgoing edges of a node u allow one to establish to which other nodes of the same rank component it is bisimilar. If we have some means to know that u is not bisimilar to any other nodes of its rank component, we can simply delete all edges outgoing from u. The deletion of a set of edges splits a rank component (i.e., we can recalculate the rank) and makes it possible

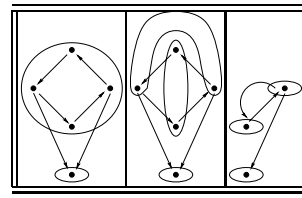

Fig. 2. Example of the First Kind of Optimization.

to recursively apply our algorithm on a simpler case. The typical case in which the above idea can be applied occurs when, at a given iteration i, there exists a block X in the set D_i of the blocks of rank i which is a singleton set $\{n\}$: then all the outgoing edges from the node n can be safely deleted. In next section we show the usefulness of this optimization in cases coming from formal verification.

6 Labeled Graphs

In several applications (e.g., Concurrency, Databases, Verification) graphs to be tested for bisimilarity have labels on edges (typically, denoting *actions*) and, sometimes, labels on nodes (typically, stating a *property* that must hold in a state). If only edges are labeled, we are in the context of the multi-relation coarsest partition problem. The definition of bisimulation has to be refined in order to take into consideration the labels on nodes and the labels on edges.

Definition 10. *Let L be a finite set of labels and A be a finite set of actions. Given a labeled graph $G = \langle N, E, \ell \rangle$, with $E \subseteq N \times A \times N$ (we use $u \xrightarrow{a} v \in E$ for $\langle u, a, v \rangle \in E$) and $\ell : N \longrightarrow L$, a* labeled bisimulation *on G is a symmetric relation $b \subseteq N \times N$ such that:*

- *if $u_1 \, b \, u_2$, then $\ell(u_1) = \ell(u_2)$;*
- *if $u_1 \, b \, u_2$ and $u_1 \xrightarrow{a} v_1 \in E$, then there is an edge $u_2 \xrightarrow{a} v_2 \in E$ and $v_1 \, b \, v_2$.*

Let us analyze how our algorithm can solve the extended problem. To start, assume that only nodes are labeled. The only change is in the initialization phase: the partition suggested by the *rank* function must be refined so as to leave in the same block only nodes with the same label. Then the algorithm can be employed without further changes. Assume now that edges can be labeled.

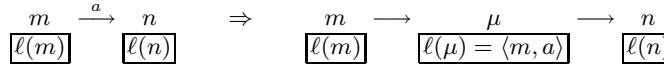

Fig. 3. Removing Edges Labels.

We suggest the following encoding: for each pair of nodes m, n and for each label a such that there is an edge $m \xrightarrow{a} n \in E$ (see also Fig. 3):

- remove the edge $m \xrightarrow{a} n$;
- add a new node μ, labeled by the pair $\langle m, a \rangle$;
- add the two (unlabeled) new edges $m \rightarrow \mu, \mu \rightarrow n$.

Starting from $G = \langle N, E, \ell \rangle$ we obtain a new graph $G' = \langle N', E', \ell \rangle$, with $E' \subseteq N \times N$, where $|N'| = |N| + |E| = O(|N|^2)$ and $|E'| = 2|E|$. Thus, our algorithm can run in $O(|E'| \log |N'|) = O(|E| \log |N|)$.

Proposition 6. *Let $G = \langle N, E, \ell \rangle$ be a graph with labeled edges and nodes, $\equiv$ be its maximum labeled bisimulation, and G' the graph with labeled nodes obtained from G. Then, $m \equiv n$ if and only if m and n are in the same class at the end of the execution of Algorithm 2 on G' with the initial partition (Step 4) further split using node labels.*

7 Testing

To the best of our knowledge there is no "official" set of benchmarks for testing an algorithm such as the one we propose in our paper. We decided to test our implementation in the context of formal verification using model checkers and considering the transition graphs they generate from a given program. In particular, we have considered the transition systems generated by the examples in the SPIN package [14]: built using ideas from [13], their aim is to check that the implementation of a protocol verifies a formal specification. Usually, the graphs generated consist of a unique strongly connected component and the set of possible labels is huge. When we rewrite them into unlabeled graphs, we usually obtain graphs on which we can perform the second optimization proposed in Section 5. Such an optimization allows us to delete edges in the graphs, obtaining graphs on which the algorithm runs in linear time. In Figure 4 we show the graph obtained for the process Cp0 of the Snooping Cache protocol. From left to right are depicted: the labeled graph generated, its corresponding unlabeled graph, the graph after our optimization, and, finally, its bisimulation contraction that can be computed in linear time.

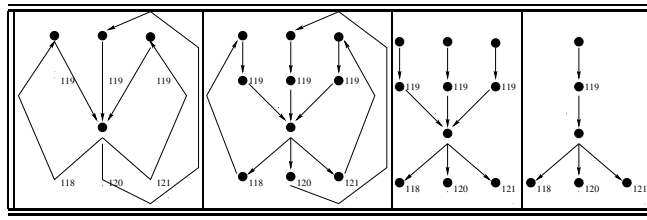

Fig. 4. Bisimulation Contraction of Cp0 from Snoopy.

These considerations about the "topology" of verification graphs suggested us some examples on which compare the performances of our algorithm with that of Paige and Tarjan. Details about the implementation (both in C and in Pascal), the machine used for the tests together with the code and the results of further tests are available at `http://www.sci.univr.it/~dovier/BISIM`. The graphs for Test 1 (cf. Figure 5) we present here are transitive closures of binary trees. The graphs for Test 2 are obtained by linking with cycles nodes at the

	Test1				Test2			
Nodes	8191	16383	32767	65535	8204	16397	32782	65551
Edges	90114	196610	425896	917506	102411	221196	475149	1015822
PT	.22	.49	1.09	2.91	.49	1.10	2.47	5.77
Alg	.06	.12	.33	.71	.23	.55	1.25	2.86

Fig. 5. Two Tests (Time in Seconds).

same level of the graphs of the first test. Then the "even" nodes of these cycles are connected by an edge to a node of an acyclic linear graph.

8 Conclusion and Further Developments

We proposed algorithms to determine the minimum, bisimulation equivalent, representation of a directed graph or, equivalently, to test bisimilarity between two directed graphs. The algorithms are built making use of algorithmic solution to the relational and single function coarsest partition problem as subroutines. In the acyclic case the performance of the sketched algorithm is linear while, in the cyclic case turns out to be linear when there are no multiple nodes. In general its performance is no worse than that of the best known solution for the relational coarsest partition problem.

In [10], Fisler and Vardi compare three minimization algorithms with an invariance checking algorithm (which does not use minimization) and argue that the last is more efficient. The minimization algorithms they consider are those of Paige and Tarjan [19], of Bouajjani, Fernandez and Halbwachs [4], and of Lee and Yannakakis [18]. An important conclusion they draw is that even if the last two algorithms are tailored to verification contexts, while the Paige and Tarjan one is not, the latter performs better. This suggests that "minimization algorithms tailored to verification settings should pay attention to choosing splitters carefully". We have presented here an algorithm, which is not specifically tailored to verification, but whose main difference w.r.t. the Paige and Tarjan's one is that it performs better choices of the splitters and of the initial partition thanks to the use of the notion of rank. In some cases we obtain linear time runs, moreover the initial partition we use allows to process the input without storing the entire structure in memory at the same time.

Our next task will be the integration of this algorithm with the symbolic model-checking techniques. Further studies relative to the applicability of the circle of ideas presented here to the problem of determining *simulations* (cf. [12]) are also under investigation.

Acknowledgements

We thank Nadia Ugel for her C implementation and the anonymous referees for useful suggestions. The work is partially supported by MURST project: *Certificazione automatica di programmi mediante interpretazione astratta.*

References

1. P. Aczel. *Non-well-founded sets*, volume 14 of *Lecture Notes, Center for the Study of Language and Information*. Stanford, 1988.
2. A. Aziz, V. Singhal, G. Swamy, and R. Brayton. Minimizing interacting finite state machines: a compositional approach to language containment. In *Proc. Int'l Conference on Computer Design*, 1994.
3. J. Barwise and L. Moss. *Vicious Circles. On the Mathematics of non-well-founded phenomena*. Lecture Notes, Center for the Study of Language and Information. Stanford, 1996.
4. A. Bouajjani, J.C. Fernandez, and N. Halbwachs. Minimal model generation. In E. Clarke and R. Kurshan, editors, *Proc. Int'l Conference on Computer-Aided Verification CAV'90*, volume 531 of *LNCS*, pages 197–203. Springer, 1990.
5. A. Bouali. XEVE, an ESTEREL verification environment. In A. J. Hu and M. Y. Vardi, editors, *Proc. Int'l Conference on Computer-Aided Verification CAV'98*, LNCS, pages 500–504. Springer, 1998.
6. A. Bouali and R. de Simone. Symbolic bisimulation minimization. In *Proc. Int'l Conference on Computer-Aided Verification CAV'92*, volume 663 of *LNCS*, pages 96–108. Springer, 1992.
7. R.E. Bryant. Graph based algorithms for Boolean function manipulation. *IEEE Transaction on Computers*, C-35(8):677–691, 1986.
8. R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench: A semantics-based tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 15(1):36–72, 1993.
9. A. Dovier, C. Piazza, and A. Policriti. A fast bisimulation algorithm. TR UDMI/14/00/RR, Dip. di Matematica e Informatica, Univ. di Udine, 2000. `http://www.dimi.uniud.it/~piazza/bisim.ps.gz`.
10. K. Fisler and M.Y. Vardi. Bisimulation and model checking. In *Proc. Correct Hardware Design and Verification Methods*, volume 1703 of *LNCS*, pages 338–341. Springer, 1999.
11. M. Forti and F. Honsell. Set theory with free construction principles. *Annali Scuola Normale Superiore di Pisa, Cl. Sc.*, IV(10):493–522, 1983.
12. M.R. Henzinger, T.A. Henzinger, and P.W. Kopke. Computing simulations on finite and infinite graphs. In *Proc. 36th IEEE Symp. on Foundations of Computer Science, FOCS 1995*, pages 453–462, 1995.
13. G.J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.
14. G.J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5), 1997.
15. G.J. Holzmann and A. Puri. A minimized automaton representation of reachable states. *Software Tools for Technology Transfer*, 2(3):270–278, November 1999.
16. J.E. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In *Theory of Machines and Computations, Ed. by Zvi Kohavi and Azaria Paz*, pages 189–196. Academic Press, 1971.
17. P.C. Kanellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, 1990.
18. D. Lee and M. Yannakakis. Online minimization of transition systems. In *Proc. 24th ACM Symposium on Theory of Computing*, pages 264–274, May 1992.
19. R. Paige and R.E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.
20. R. Paige, R.E. Tarjan, and R. Bonic. A linear time solution to the single function coarsest partition problem. *Theoretical Computer Science*, 40(1):67–84, 1985.
21. F. Rahim. Property-dependent modular model checking application to VHDL with computational results. In *Proc. Int'l Workshop HLDVT*, 1998.