Metrics-Based Incremental Determinization of Finite Automata

Sergiu I. Balan, Gianfranco Lamperti, Michele Scandale

Metrics-Based Incremental Determinization of Finite Automata

Sergiu I. Balan[1], Gianfranco Lamperti[1], and Michele Scandale[2]

[1] Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Brescia (I)
[2] Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano (I)

Abstract. Some application domains, including monitoring of active systems in artificial intelligence and model-based mutation testing in software engineering, require determinization of finite automata to be performed incrementally. To this end, an algorithm called *Incremental Subset Construction* (*ISC*) was proposed a few years ago. However, this algorithm was recently discovered to be incorrect is some instance problems. The incorrect behavior of *ISC* originates when the redirection of a transition causes a portion of the automaton to be disconnected from the initial state. This misbehavior is disturbing in two ways: portions of the resulting automaton are disconnected and, as such, useless; moreover, a considerable amount of computation is possibly wasted for processing these disconnected parts. To make *ISC* sound, a metrics-based technique is proposed in this paper, where the distance between states is exploited in order to guarantee the connection of the automaton, thereby allowing *ISC* to achieve soundness. Experimental results show that, besides being effective, the proposed technique is efficient too.

Keywords: Finite Automata; Incremental Determinization; Incremental Subset Construction; Model-Based Reasoning.

1 Introduction

For efficiency reasons, determinization of finite automata is essential to a wide range of applications, from pattern matching based on regular expressions [8] to analysis of protein sequences [4]. The determinization of a nondeterministic finite automaton (NFA) into an equivalent deterministic finite automaton (DFA) is commonly performed by *Subset Construction* (*SC*), an algorithm introduced several decades ago [18].

However, some application domains, including monitoring and diagnosis of active systems [12, 17, 15] in artificial intelligence, and model-based mutation testing [1–3, 9] in software engineering, require determinization to be carried out incrementally, where the NFA expands over time and determinization is required at each expansion.

Specifically, in [12], principles and techniques of diagnosis of active systems are presented. A technique for incremental processing of temporal observations in model-based reasoning is proposed in [17]. Despite being specific for automata derived from temporal observations, this technique contains the seeds of a more general-purpose algorithm for automata determinization. In [15], the notion of monotonic monitoring of discrete-event systems is introduced, which is supported by specific constraints on the fragmentation of the temporal observation, leading to the notion of stratification.

In model-based mutation testing, a test model is mutated for test case generation, thereby becoming a *mutant*. The resulting test cases are able to detect whether the faults in the mutated models have been implemented in the system under test. For this purpose, a conformance check between the original and the mutated model is required. An approach is proposed for conformance checking of action systems in [1], which relies on constraint solving techniques. This approach is extended in [2, 3] by two techniques: a strategy to efficiently handle a large number of mutants and incremental solving. An extensive approach to model-based mutation testing can be found in [9], where input-output conformance check [19] is shown to benefit from incremental determinization.

The rest of the paper is organized as follows. In Section 2, the classical technique for NFA determinization is recalled, and the problem of incremental determinization is defined. Section 3 provides some hints on the application domain in which incremental determinization originated. Section 4 introduces the basic notions of the metrics-based incremental determinization technique, while a detailed specification of the algorithm is outlined in Section 5. A discussion on why avoiding disconnection is provided in Section 6. Experimental results are shown in Section 7. Section 8 concludes the paper.

2 Determinization of Finite Automata

According to the *SC* algorithm, each state in the DFA is identified by a subset of the states of the NFA. *SC* yields the DFA starting from the ε-closure of the initial state of the NFA, which becomes the initial state of the DFA, and by progressively generating the successor states of each state $\mathbb{N}$ (subset of NFA states) as the ε-closure of the set of NFA states reached by a label ℓ from the NFA states in $\mathbb{N}$, called the ℓ-*closure* of $\mathbb{N}$.

Example 1 Traced in Figure 1 is the determinization of the NFA outlined in the left side. Gray states indicate that further processing is required. Next to the NFA is the sequence of intermediate DFAs leading to the equivalent DFA outlined in the right side.

1. The initial state of the DFA is the ε-closure of the initial state of the NFA, $\{0, 1\}$.
2. Considering $\{0, 1\}$, two transitions are created, $\{0, 1\} \xrightarrow{a} \{1, 2\}$ and $\{0, 1\} \xrightarrow{b} \{2, 3\}$, which are obtained by considering each symbol of the alphabet marking a transition exiting either 0 or 1. With a, two transitions are applicable in the NFA, $0 \xrightarrow{a} 1$ and $0 \xrightarrow{a} 2$. Thus, the target state for the transition exiting $\{0, 1\}$ and marked by a in the

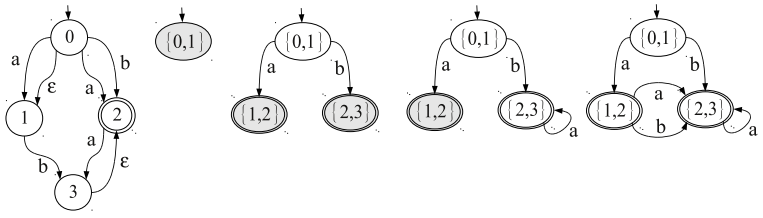

Fig. 1. Determinization by *SC*: the NFA on the left side is determinized into the equivalent DFA on the right side, starting from the initial state $\{0, 1\}$.

DFA is the ε-closure of $\{1, 2\}$, namely $\{1, 2\}$. With b, we come up with transition $\{0, 1\} \xrightarrow{b} \{2, 3\}$.

3. Considering $\{2, 3\}$, since we have $2 \xrightarrow{a} 3$ in the NFA (while no transition marked by a symbol in the alphabet exits state 3) and the ε-closure of $\{3\}$ is $\{2, 3\}$, an auto-transition $\{2, 3\} \xrightarrow{a} \{2, 3\}$ is created. Notice how this transition does not cause the creation of a new state.
4. Considering $\{1, 2\}$, since $1 \xrightarrow{b} 3$ and $2 \xrightarrow{a} 3$ are in the NFA and the ε-closure of $\{3\}$ is $\{2, 3\}$, two transitions are created in the DFA, $\{1, 2\} \xrightarrow{a} \{2, 3\}$ and $\{1, 2\} \xrightarrow{b} \{2, 3\}$, without the generation of any new state.

The final states of the DFA are those including a state which is final in the NFA, in our example, $\{1, 2\}$ and $\{2, 3\}$.

Definition 1 (Incremental Determinization Problem). *Let $\mathcal{N}$ be an NFA and $\mathcal{D}$ the DFA equivalent to $\mathcal{N}$ (as generated by* SC*). Let $\Delta\mathcal{N}$ be an expansion of $\mathcal{N}$ yielding $\mathcal{N}'$, a new NFA. Generate the DFA $\mathcal{D}'$ equivalent to $\mathcal{N}'$ based on $\mathcal{N}$, $\mathcal{D}$, and $\Delta\mathcal{N}$.*

Definition 1 refers to a single determinization step, following a single expansion of the NFA. When the NFA expands over time, incremental determinization is required several (possibly many) times, after each expansion. In principle, the incremental determinization problem can be solved by means of *SC* by determinizing $\mathcal{N}'$ while neglecting $\mathcal{N}$, $\mathcal{D}$, and $\Delta\mathcal{N}$. However, this naive approach is bound to poor performances, especially when $\mathcal{N}'$ becomes increasingly large, as the incremental nature of $\mathcal{N}'$ is not exploited. To solve the incremental determinization problem efficiently, an *Incremental Subset Construction* algorithm (*ISC*) was proposed [16]. *ISC* was recently discovered to be incorrect in some instances, as it generates spurious states which are disconnected from the initial state, as shown in [10]. What may be problematic in the processing is not that the final DFA is disconnected (which is however unsound) but, rather, the possibly large amount of irrelevant processing uselessly wasted on the disconnected part. To cope with this problem, a revisitation of *ISC* is presented in [5], called *RISC*, where three variants of the algorithm are proposed. In this paper, we provide details on the most elegant variant, which exploits a specific metrics in the DFA.

3 Originating Application Domain

The need for incremental determinization stems from the domain of model-based diagnosis (MBD) of active systems [12], specifically, monitoring-based diagnosis [13, 14, 17]. MBD aims to diagnose a physical system based on the model of the system and relevant observations. The discrepancy between the normal behavior of the system and the observation allows the diagnostic engine to generate candidate diagnoses, where each candidate is a set of faulty components. MBD can be applied to discrete-event systems (DESs) [7], whose behavior is modeled as networks of components, with each component being a communicating automaton [6]. Active systems are a special class of asynchronous DESs, where components may exchange events to one another

by means of links. During operation, the active system reacts to external events by performing system transitions, which possibly trigger new transitions by generating events toward neighboring components through links. The active system evolves according to its model, which incorporates both normal and faulty behavior, by performing a sequence of component transitions within its behavioral space. The evolution of the system is a sequence of transitions, called the trajectory of the system.

The problem lies in the ambiguity of the mode in which the system is evolving, because only a subset of component transitions are visible by the diagnosis engine. If the transition is visible, it generates an observable label. Consequently, the trajectory is perceived by the engine as a sequence of observable labels, called the trace.

The diagnosis engine performs consistent reasoning and eventually provides the candidate diagnoses, where each candidate is a set of faulty transitions, and corresponds to one or several candidate trajectories, each one equally possible. In large, distributed systems, the problem is complicated by the way observable labels are conveyed to the observer, which may involve multiple (possibly noisy) channels. This causes a distortion of the trace, where each label is perceived as a set of candidate labels, while the total temporal ordering among labels is relaxed to partial temporal ordering. The result is an uncertain temporal observation [11], which is represented by a directed acyclic graph, where nodes are marked by candidate labels, while edges define partial temporal ordering among nodes.

However, the observation graph, namely $\mathcal{O}$, is inconvenient for processing as is. A surrogate of it, namely $Isp(\mathcal{O})$, the index space of $\mathcal{O}$, is used instead. The index space is a DFA whose regular language is the whole set of candidate traces of the relevant observation. The point is, $Isp(\mathcal{O})$ is derived via subset-construction by an NFA called prefix space, denoted $Psp(\mathcal{O})$, which is directly derived from $\mathcal{O}$. Thus, three transformations occur for a trace $\mathcal{T}$: $\mathcal{T} \rightsquigarrow \mathcal{O} \Rightarrow Psp(\mathcal{O}) \Rightarrow Isp(\mathcal{O})$, where the former ($\rightsquigarrow$) depends on the nature of both the communication channels and the observer, and, as such, is beyond the scope of the diagnostic engine, while the others ($\Rightarrow$) are artificially performed by the diagnostic engine for computational purposes.

In monitoring-based diagnosis, candidate diagnoses must be generated each time a piece of observation is received. Typically, the observation graph is received as a sequence of fragments, with each fragment carrying information on one node and the arcs coming from its parents. These are called fragmented observations. At the reception of each fragment, the index space is to be updated based on the extension of the prefix space. Since generating the sequence of index spaces via *SC* may become computationally prohibitive in real applications, as each index space is generated from scratch at each new fragment, a better solution is to make *SC* incremental, so that each index space in the sequence is generated as an update of the previous one, thereby pursuing computational reuse.

4 Metrics-Based Incremental Subset Construction

According to the incremental determinization problem (Definition 1), based on an NFA $\mathcal{N}$, the equivalent DFA $\mathcal{D}$, and an expansion $\Delta\mathcal{N}$ of $\mathcal{N}$, the determinization $\mathcal{D}'$ of the expanded NFA $\mathcal{N}' = \mathcal{N} \cup \Delta\mathcal{N}$ is required. What makes intriguing the problem is

the possible exploitation of $\mathcal{D}$ instead of starting from scratch the determinization of $\mathcal{N}'$. Rather than applying *SC* to $\mathcal{N}'$ (disregarding altogether $\mathcal{N}$, $\mathcal{D}$, and $\Delta\mathcal{N}$), $\mathcal{D}'$ is determined by updating $\mathcal{D}$ based on $\Delta\mathcal{N}$. This idea is substantiated by algorithm *ISC*.

Let d be the identifier of a state of the automaton $\mathcal{D}$ being processed by *ISC*, ℓ a symbol of the alphabet, and $\mathbb{N}$ the ℓ-closure of the NFA states incorporated in d. The triple $(d, \ell, \mathbb{N})$ is a *bud* for $\mathcal{D}$. A bud indicates that further processing needs to be performed to update the transition exiting d and marked by ℓ in $\mathcal{D}$.

ISC produces the same results as *SC* by exploiting a stack of buds. Roughly, the bud-stack parallels the stack of DFA states in *SC*. Just as new DFA states are inserted into the stack by *SC* and thereafter processed, so are the new buds accumulated into the bud-stack of *ISC* and processed one by one. In *SC*, the first state pushed onto the stack is the initial state of the DFA. In *ISC*, the bud-stack is initialized by a number of buds relevant to the states exited by the new transitions in the NFA.

The algorithm loops, by popping a bud at each iteration, until the bud-stack becomes empty. While processing each bud, new buds are possibly inserted into the bud-stack. The processing of each bud depends on both the bud and the current DFA.

In order to make the algorithm sound by avoiding the disconnection of the DFA, a metrics is introduced, as formalized in Definition 2, where $\mathcal{D}$ is the DFA being processed by *ISC*.

Definition 2 (Distance). *Let d be a state in $\mathcal{D}$ and d_0 the initial state of $\mathcal{D}$. The* distance *of d, written $\delta(d)$, is the minimal number of transitions connecting d_0 with d.*

Example 2 Considering the DFA displayed in the right side of Figure 1, we have $\delta(\{0, 1\}) = 0$ (for the initial state), and $\delta(\{1, 2\}) = \delta(\{2, 3\}) = 1$.

Note that, even if the DFA is cyclic, the distance of each state d is always less than the (finite) number of states of the DFA, as the minimal path of transitions connecting the initial state to d cannot include cycles.

5 Detailed Specification of Metrics-Based *ISC*

A pseudo-coded formalization of metrics-based *ISC* is outlined below (lines 1–90). *ISC* takes as input an NFA $\mathcal{N}$, the equivalent DFA $\mathcal{D}$ (as being generated by *SC*), and an extension $\Delta\mathcal{N}$ of $\mathcal{N}$. $\mathcal{D}$ is updated based on $\Delta\mathcal{N}$ so as to make it equivalent to the extended NFA $\mathcal{N}' = \mathcal{N} \cup \Delta\mathcal{N}$ (as if it were generated by *SC*). We assume that each state d in $\mathcal{D}$ is qualified by the relevant distance $\delta(d)$, as specified in Definition 2.

ISC is supported by bud-stack $\mathcal{B}$. Buds in $\mathcal{B}$ are implicitly grouped by the first two fields: if a new bud $B = (d, \ell, \mathbb{N})$ is pushed onto $\mathcal{B}$ and a bud $B' = (d, \ell, \mathbb{N}')$ is already in $\mathcal{B}$, then B' will be absorbed by B, thereby becoming $B = (d, \ell, \mathbb{N} \cup \mathbb{N}')$.

Throughout the pseudo-code, we keep a distinction between the identifier of a state in $\mathcal{D}$ and its content, where the former is a symbol (e.g. d), while the latter is a set of nodes in $\mathcal{N}'$ (e.g. $\mathbb{N}$). The content of a node d is written $\|d\|$. During execution, the content may change, while the identifier cannot.

The algorithm makes use of two auxiliary procedures, *Relocate* and *Expand*. Procedure *Relocate* (lines 8–26) takes as input a state d and a distance $\bar{\delta}$, and possibly updates

the distance of d and the distance of a finite set of states reached by d. Specifically, $\bar{\delta}$ represents the upper bound for the distance of d as a consequence of an update in the topology of $\mathcal{D}$, typically by the creation of a new transition entering d.

Example 3 Outlined in the left side of Figure 2 is a fragment of $\mathcal{D}$ where states are associated with relevant distances before the insertion of transition from d_1 to d (while transition labels are omitted). Since $\delta(d_1) = 2$ and $\delta(d) = 5$ (we assume other transitions entering d but not displayed in the figure), the distance of d cannot be larger than $\bar{\delta} = \delta(d_1) + 1 = 3$. Since $\delta(d) > \bar{\delta}$, the distance $\delta(d)$ becomes $\bar{\delta}$, while the distances of successive states d_2, d_3, and d_4 change as shown in the right side of the figure.

After the possible change of $\delta(d)$ (lines 12–13), relocation of distances of successive nodes of d is required. This is accomplished by means of breadth-first distance-propagation. First a list $\mathbb{D}$ of candidate states is initialized by the singleton $[d]$ (line 14). Then, a loop is iterated until $\mathbb{D}$ becomes empty (lines 15–24). At each iteration, the first candidate h (head) is removed from $\mathbb{D}$ and the children $\mathbb{D}_c$ of h are considered. For each child $d_c \in \mathbb{D}_c$, the relevant distance is possibly updated (lines 19–20) and, if so, d_c is appended to $\mathbb{D}$ (line 21), in order to propagate this change to its successors.

Example 4 In Example 3, the update of the distance of d is propagated as follows:

1. Initial configuration: $\mathbb{D} = [d]$, $\delta(d) = 3$, $\delta(d_2) = 6$, $\delta(d_3) = 6$, and $\delta(d_4) = 7$.
2. First iteration: d is removed from $\mathbb{D}$ and the distance of the children of d are updated, thereby $\delta(d_2) = \delta(d_3) = 4$, while $\mathbb{D} = [d_2, d_3]$.
3. Second iteration: d_2 is removed from $\mathbb{D}$, however the distance of d_3 (the only child of d_2) is not updated, as $\delta(d_3) \not> \delta(d_2) + 1$, in fact $\delta(d_2) = \delta(d_3) = 4$.
4. Third iteration: d_3 is removed from $\mathbb{D}$ and the distance of child d_4 is updated, $\delta(d_4) = 5$. Since d_4 has no child, $\mathbb{D}$ remains empty and the propagation terminates.

Procedure *Expand* (lines 30–50) takes as input a state d in $\mathcal{D}$ and adds to its content the subset $\mathbb{N}$ of states in $\mathcal{N}'$. The bud-stack $\mathcal{B}$ is extended by the buds relevant to d and the labels exiting nodes in $(\mathbb{N} - \|d\|)$ in $\mathcal{N}'$. If the content of the extended node d equals the content of a node d' already in $\mathcal{D}$, then the two nodes are merged into a single node (lines 36–49). Before the merging, if the distances of d and d' differ, the distance of the state with maximal distance is updated with the distance of the other state by means of *Relocate* (which also propagates the distance change to successor nodes). Then, all transitions entering (exiting) d are redirected to (from) d'. Finally, after the removal of d, all buds relevant to d are renamed to d'. The redirection of transitions exiting d may cause nondeterminism exiting d' (two transitions exiting d' that are marked by the same label ℓ). However, such nondeterminism is transient and disappears at the end of the processing.

Fig. 2. Relocation of distances after the insertion of the new transition entering d.

Considering the body of *ISC* (lines 51–90), after determining the subset $\bar{N}$ of states in $\mathcal{N}$ that are exited by transitions in $\Delta\mathcal{N}$, $\mathcal{N}$ is extended by $\Delta\mathcal{N}$. Bud-stack $\mathcal{B}$ is initialized with buds (d, ℓ, N), where N is the ℓ-closure of $\|d\| \cap \bar{N}$. A loop is then iterated until $\mathcal{B}$ becomes empty (lines 55–89). At each iteration, a bud (d, ℓ, N) is popped from $\mathcal{B}$. Depending on the content of the bud, one of seven *action rules*, $\mathcal{R}_1 \cdots \mathcal{R}_7$, is applied, in the form of [*Condition*] $\Rightarrow$ *Action*, as specified below.

($\mathcal{R}_1$) [$\ell = \varepsilon$] $\Rightarrow$ d is expanded by N (line 58).

($\mathcal{R}_2$) [$\ell \neq \varepsilon$, no ℓ-transition exits d, $\exists d'$ such that $\|d'\| = N$] $\Rightarrow$ A transition $d \xrightarrow{\ell} d'$ is inserted into $\mathcal{D}$ and distance relocation is applied to d' (lines 61–62).

($\mathcal{R}_3$) [$\ell \neq \varepsilon$, no ℓ-transition exits d, $\nexists d'$ such that $\|d'\| = N$] $\Rightarrow$ An empty state d' and a transition $d \xrightarrow{\ell} d'$ are created; then, after assigning the distance of d', the latter is expanded by N (lines 64–66).

($\mathcal{R}_4$) [$\ell \neq \varepsilon$, an ℓ-transition t exits d, the state d' entered by t is not the initial state, no other transition enters d'] $\Rightarrow$ d' is expanded by N (line 71).

($\mathcal{R}_5$) [$\ell \neq \varepsilon$, an ℓ-transition t exits d, either the state d' entered by t is the initial state or another transition different from t enters d' from a state d_p such that $\delta(d_\mathrm{p}) \leq \delta(d)$, $\exists d''$ such that $\|d''\| = \|d'\| \cup N$] $\Rightarrow$ t is redirected toward d'' and distance propagation is applied to d'' (lines 74–75).

($\mathcal{R}_6$) [$\ell \neq \varepsilon$, an ℓ-transition t exits d, either the state d' entered by d is the initial state or another transition different from t enters d' from a state d_p such that $\delta(d_\mathrm{p}) \leq \delta(d)$, $\nexists d''$ such that $\|d''\| = \|d'\| \cup N$] $\Rightarrow$ d' is duplicated into d'' (along with exiting transitions and buds), t is redirected toward d'', distance propagation is applied to d'', the latter is expanded by N (lines 77–80).

($\mathcal{R}_7$) [$\ell \neq \varepsilon$, an ℓ-transition t exits d, either the state d' entered by t is the initial state or another transition different from t enters d' and the distances of the states exited by these other transitions entering d'' are all greater than $\delta(d)$] $\Rightarrow$ all transitions entering d' other than t are removed and surrogated by newly created buds, d' is expanded by N (lines 83–85).

Rules $\mathcal{R}_4$, $\mathcal{R}_5$, $\mathcal{R}_6$, and $\mathcal{R}_7$ correspond to a single bud (line 56), but may be applied several times, depending on the number of ℓ-transitions exiting d (as stated above, a temporary nondeterminism in $\mathcal{D}$ may be caused after merging two states by *Expand*).

Rules $\mathcal{R}_5$, $\mathcal{R}_6$, and $\mathcal{R}_7$ requires some additional explanation as far as the connectivity of $\mathcal{D}$ is concerned, as they all remove at least one transition entering d', which in principle may cause a disconnection. Considering $\mathcal{R}_5$ and $\mathcal{R}_6$, since there exists a transition, other than t, entering d' from d_p such that $\delta(d_\mathrm{p}) \leq \delta(d)$, if the removal of t from d' causes a disconnection, also d_p will be disconnected, but in this case, being a successor of d, we will have $\delta(d_\mathrm{p}) > \delta(d)$, a contradiction. Hence, no disconnection occurs. Moreover, the distance of d' cannot increase, as it is at most $\delta(d) + 1$.

Considering $\mathcal{R}_7$, since all other transitions entering d' are such that the state they exit has distance greater than $\delta(d)$, the removal of t from d' is not safe because all parent states of d' other than d might be connected to the initial state by means of t. On the other hand, based on the same reasoning adopted for $\mathcal{R}_5$ and $\mathcal{R}_6$, all these other transitions entering d' can be safely removed because they are not essential to the connection of d (which has shorter distance). Hence, the removal of all other entering

transitions leaves d' still connected to the initial state. Moreover, the distance of d' cannot increase, as it is $\delta(d)+1$.

```
1.  algorithm ISC (𝒩, 𝒟, Δ𝒩)
2.    𝒩 = (N, Σ, T_n, n_0, F_n): an NFA,
3.    𝒟 = (D, Σ, T_d, d_0, F_d): the DFA equivalent to 𝒩 (as generated by SC),
4.    Δ𝒩 = (ΔN, ΔT_n, ΔF_n): an extension of 𝒩;
5.    side effects
6.      𝒩 is extended by Δ𝒩,
7.      𝒟 is updated, becoming the DFA equivalent to (𝒩' = 𝒩 ∪ Δ𝒩) (as by SC);
8.    auxiliary procedure Relocate(d, δ̄)
9.      d: a state in 𝒟,
10.     δ̄: the upper-bound distance for d;
11.   begin ⟨Relocate⟩
12.     if δ(d) > δ̄ then
13.       δ(d) := δ̄;
14.       𝔻 := [d];
15.       repeat
16.         Remove the first element h (head) from 𝔻;
17.         Let 𝔻_c be the set of child states of h in 𝒟;
18.         foreach d_c ∈ 𝔻_c do
19.           if δ(d_c) > δ(h) + 1 then
20.             δ(d_c) := δ(h) + 1;
21.             Append d_c to 𝔻
22.           endif
23.         endfor
24.       until Empty(𝔻)
25.     endif
26.   end ⟨Relocate⟩;
27.   auxiliary procedure Expand (d, ℕ)
28.     d: a state in 𝒟,
29.     ℕ: a subset of states in 𝒩';
30.   begin ⟨Expand⟩
31.     if ℕ ⊈ ‖d‖ then
32.       𝔹' := {(d, ℓ, ℕ') | ℓ ∈ Σ, ℕ' = ℓ-closure(ℕ − ‖d‖), ℕ' ≠ ∅};
33.       Push buds 𝔹' onto 𝔅;
34.       Enlarge ‖d‖ by ℕ;
35.       if d ∉ F_d, ℕ ∩ F_n ≠ ∅ then Insert d into F_d endif;
36.       if 𝒟 includes a state d' such that ‖d'‖ = ‖d‖ then
37.         if δ(d) > δ(d') then
38.           Relocate(d, δ(d'))
39.         elsif δ(d) < δ(d') then
40.           Relocate(d', δ(d))
41.         endif;
42.         Redirect to d' all transitions entering d and remove duplicates;
43.         Redirect from d' all transitions exiting d and remove duplicates;
```

```
44.              if d is the initial state d_0 then d_0 := d' endif;
45.              if d ∈ F_d then Remove d from F_d endif;
46.              Remove d from 𝒟;
47.              Convert to d' the buds in ℬ relevant to d
49.            endif
49.          endif
50.      end ⟨Expand⟩;
51.      begin ⟨ISC⟩
52.        N̄ := the set of states in 𝒩 exited by transitions in ΔT_n;
53.        Extend 𝒩 by Δ𝒩;
54.        ℬ := {(d, ℓ, ℕ) | d ∈ D, n ∈ ‖d‖ ∩ N̄, n -ℓ-> n' ∈ ΔT_n, ℕ = ℓ-closure(‖d‖ ∩ N̄)};
55.        repeat
56.          Pop bud (d, ℓ, ℕ) from the top of bud-stack ℬ;
57.  (ℛ1)   if ℓ = ε then
58.            Expand (d, ℕ)
59.          elsif no ℓ-transition exits d then
60.  (ℛ2)     if 𝒟 includes a state d' such that ‖d'‖ = ℕ then
61.              Insert a new transition d -ℓ-> d' into 𝒟;
62.              Relocate(d', δ(d) + 1)
63.  (ℛ3)     else
64.              Create a new state d' and insert d -ℓ-> d' into 𝒟;
65.              δ(d') := δ(d) + 1;
66.              Expand (d', ℕ)
67.            endif
68.          else
69.            foreach transition t = d -ℓ-> d' such that ℕ ⊈ ‖d'‖ do
70.  (ℛ4)       if d' ≠ d_0 and no other transition enters d' then
71.                Expand (d', ℕ)
72.              elsif 𝒟 includes a transition d_p -ℓ'-> d' ≠ t such that δ(d_p) ≤ δ(d) then
73.  (ℛ5)         if 𝒟 includes a state d'' such that ‖d''‖ = ‖d'‖ ∪ ℕ then
74.                  Redirect t toward d'';
75.                  Relocate(d'', δ(d) + 1)
76.  (ℛ6)         else
77.                  Create a copy d'' of d' along with the buds relevant to d'';
78.                  Redirect t toward d'';
79.                  δ(d'') := δ(d) + 1;
80.                  Expand (d'', ℕ)
81.                endif
82.  (ℛ7)         else
83.                  Remove all transitions entering d' other than t;
84.                  Update ℬ with the buds for the starting state of the removed transitions;
```

85. *Expand* $(d', \mathbb{N})$
86. **endif**
87. **endfor**
88. **endif**
89. **until** bud-stack $\mathcal{B}$ becomes empty
90. **end** $\langle ISC \rangle$.

Definition 3 (Configuration). *Let $\mathcal{D}_i$ be the automaton $\mathcal{D}$ after the processing of i buds, and $\mathcal{B}_i$ the corresponding instance of $\mathcal{B}$. The pair $\alpha_i = (\mathcal{D}_i, \mathcal{B}_i)$ is a* configuration*, where $\alpha_0 = (\mathcal{D}_0, \mathcal{B}_0)$ is the initial configuration, with $\mathcal{D}_0$ being the DFA in input and $\mathcal{B}_0$ the initial instance of $\mathcal{B}$. The* path *of* ISC *is the sequence* $[\alpha_0, \ldots, \alpha_i, \alpha_{i+1}, \ldots]$ *of configurations.*

Example 5 Outlined in plain lines in the left side of Figure 3 is an NFA $\mathcal{N}$. An expansion of $\mathcal{N}$, namely $\mathcal{N}' = \mathcal{N} \cup \Delta\mathcal{N}$, is represented in dashed lines, where $\Delta\mathcal{N}$ includes a new state and three transitions. The DFA $\mathcal{D}'$ equivalent to $\mathcal{N}'$ (generated by *SC*) is shown in the right side of Figure 3. We now show how the same result is generated by means of *ISC*, starting from $\mathcal{N}$, $\mathcal{D}$, and $\Delta\mathcal{N}$. Depicted in Figure 4 is the corresponding path of *ISC*, namely $[\alpha_0, \alpha_1, \ldots, \alpha_7]$ (the last configuration $\alpha_8 = \mathcal{D}'$, not shown in Figure 4, equals the DFA placed on the right-hand side of Figure 3). Distance of each state is indicated, while each bud $(d, \ell, \mathbb{N})$ is represented as a dashed arc exiting d, marked by ℓ, and entering a filled node marked by $\mathbb{N}$.

Initially, according to *ISC* (line 54), four buds are generated for $\alpha_0 = (\mathcal{D}_0, \mathcal{B}_0)$, where $\mathcal{D}_0 = \mathcal{D}$ and $\bar{\mathbb{N}} = \{2\}$ (line 52). Therefore, the initial buds are relevant to nodes incorporating state 2, namely $\{1, 2\}$ and $\{2, 3\}$, giving rise to bud-stack $\mathcal{B}_0 = [B_1, B_2, B_3, B_4]$, where $B_1 = (\{2, 3\}, b, \{2, 3, 4\})$, $B_2 = (\{2, 3\}, a, \{0, 1, 2, 3\})$, $B_3 = (\{1, 2\}, b, \{2, 3, 4\})$, and $B_4 = (\{1, 2\}, a, \{0, 1, 2, 3\})$. Then, the main loop (lines 55–89) is started and buds are precessed one by one. Each processed bud is indicated in Figure 4 by a dashed filled node. The path of *ISC* is described below.

(α_0) $B = (\{1, 2\}, a, \{0, 1, 2, 3\}) \Rightarrow$ rule $\mathcal{R}_6$: since transition $\{1, 2\} \xrightarrow{a} \{2, 3\}$ is not essential to the connectivity of state $\{2, 3\}$, state $\{2, 3\}$ is duplicated (along with ex-

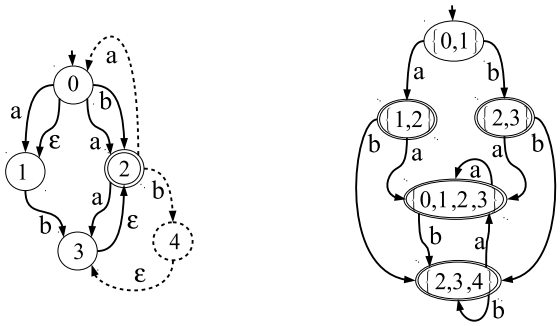

Fig. 3. Determinization of the expanded NFA $\mathcal{N}' = \mathcal{N} \cup \Delta\mathcal{N}$ by *SC* (the dashed part is $\Delta\mathcal{N}$).

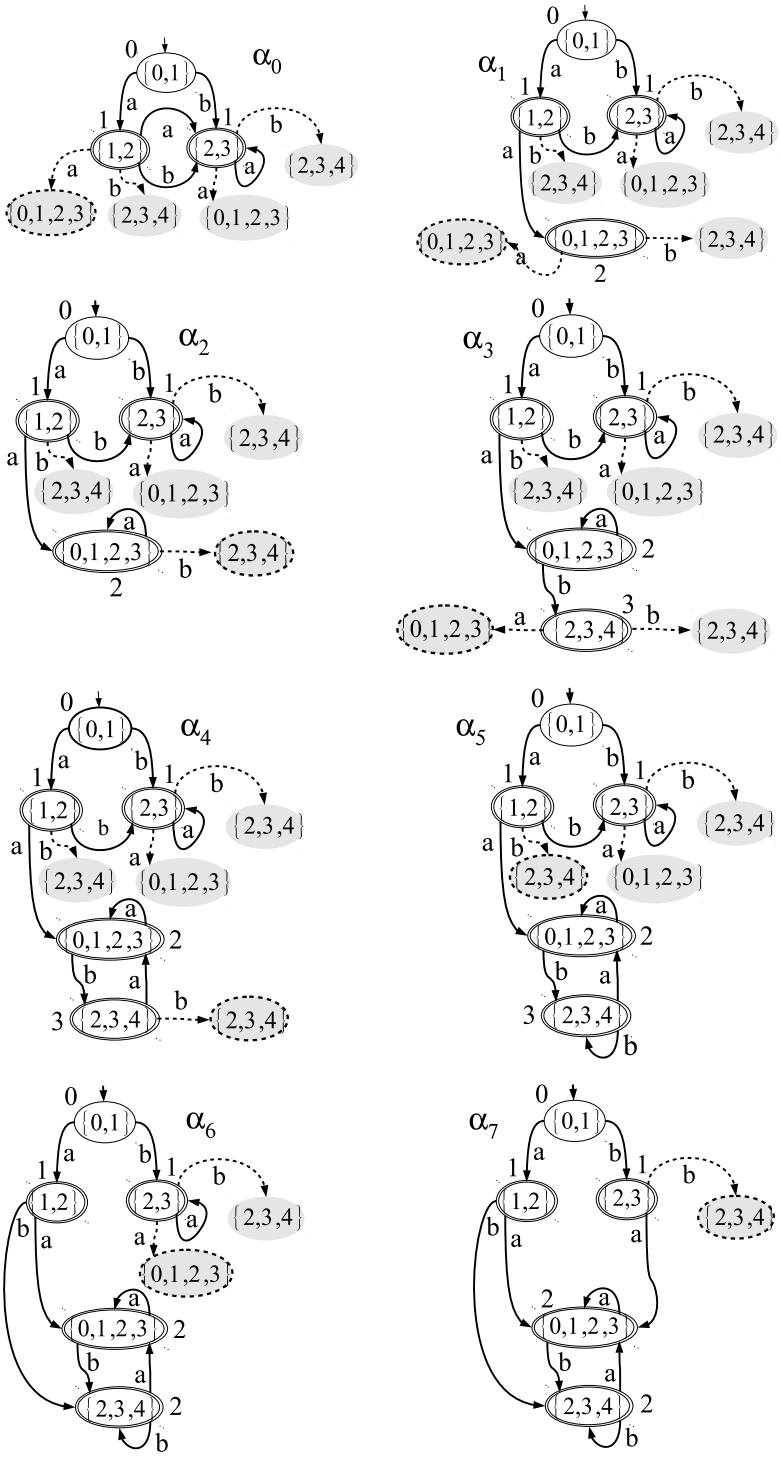

Fig. 4. Path of algorithm *ISC* for incremental determinization of the expanded NFA in Figure 3.

iting transitions and buds), transition $\{1,2\} \xrightarrow{a} \{2,3\}$ is redirected toward the new state, with the latter being expanded to $\{0,1,2,3\}$.

(α_1) $B = (\{0,1,2,3\}, a, \{0,1,2,3\}) \Rightarrow$ rule $\mathcal{R}_3$: a new empty state entered by a new transition is created; then, the expansion of the empty state by $\mathbb{N} = \{0,1,2,3\}$ causes a merging with an equivalent state, creating $\{0,1,2,3\} \xrightarrow{a} \{0,1,2,3\}$.

(α_2) $B = (\{0,1,2,3\}, b, \{2,3,4\}) \Rightarrow$ rule $\mathcal{R}_3$: a new empty state entered by a new transition is created; then, the expansion of the empty state by $\mathbb{N} = \{2,3,4\}$ causes the creation of buds $(\{2,3,4\}, a, \{0,1,2,3\})$ and $(\{2,3,4\}, b, \{2,3,4\})$.

(α_3) $B = (\{2,3,4\}, a, \{0,1,2,3\}) \Rightarrow$ rule $\mathcal{R}_3$: a new empty state entered by a new transition is created; then, the expansion of the empty state by $\mathbb{N} = \{0,1,2,3\}$ causes a merging with an equivalent state, creating $\{2,3,4\} \xrightarrow{a} \{0,1,2,3\}$.

(α_4) $B = (\{2,3,4\}, b, \{2,3,4\}) \Rightarrow$ rule $\mathcal{R}_3$: a new empty state entered by a new transition is created; then, the expansion of the empty state by $\mathbb{N} = \{2,3,4\}$ causes a merging with an equivalent state, creating $\{2,3,4\} \xrightarrow{b} \{2,3,4\}$.

(α_5) $B = (\{1,2\}, b, \{2,3,4\}) \Rightarrow$ rule $\mathcal{R}_5$: since transition $\{1,2\} \xrightarrow{b} \{2,3\}$ is not essential to the connectivity of $\{2,3\}$, it is redirected toward existing state $\{2,3,4\}$.

(α_6) $B = (\{2,3\}, a, \{0,1,2,3\}) \Rightarrow$ rule $\mathcal{R}_5$: since transition $\{2,3\} \xrightarrow{a} \{2,3\}$ is not essential to the connectivity of $\{2,3\}$, it is redirected toward state $\{0,1,2,3\}$.

(α_7) $B = (\{2,3\}, b, \{2,3,4\}) \Rightarrow$ rule $\mathcal{R}_2$: transition $\{2,3\} \xrightarrow{b} \{2,3,4\}$ is created.

The processing of the last bud in configuration α_7 makes the bud-stack empty, thereby *ISC* terminates. As required, the automaton obtained after α_7 equals the DFA generated by *SC* as a determinization of $\mathcal{N}'$ (see Figure 3).

6 Why Avoiding Disconnection?

At this point one may ask why maintaining each state of $\mathcal{D}$ connected with the initial state is so important in incremental determinization. After all, this attention is not applied to possible nondeterminism caused by the merging of two states in the *Expand* function. So, why worrying about disconnection? What differentiates nondeterminism from disconnection in $\mathcal{D}$ is that possible nondeterminism generated by *Expand* is always transient, as it invariably disappears before the end of *ISC*. By contrast, the disconnection of a state (along with a possibly large portion of DFA rooted in this state) can be permanent. The detrimental effect of disconnection is twofold:

1. The resulting DFA embodies a (possibly large) set of unreachable states;
2. Being not aware of the disconnection, *ISC* is bound to waste computational resources in processing these disconnected states.

Example 6 Drawn in plain lines in the left side of Figure 5 is an NFA. An expansion of the NFA is represented in dashed lines (four auto-transitions). The DFA equivalent to the NFA (generated by *SC*) is shown in the right side of Figure 5. We now trace the processing of incremental determinization as specified in [16], where connection of states is not checked. Depicted in Figure 6 is the path of of the algorithm, namely

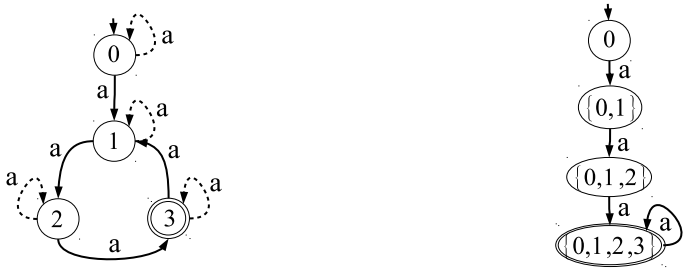

Fig. 5. Determinization of the expanded NFA by *SC*, with the dashed part being the expansion.

$[\alpha_0, \alpha_1, \ldots, \alpha_9]$. Each bud (d, ℓ, N) is represented as a dashed arc exiting d, marked by ℓ, and entering a filled node marked by N.

At the beginning, four buds are generated for $\alpha_0 = (\mathcal{D}_0, \mathcal{B}_0)$, where $\mathcal{D}_0 = \mathcal{D}$ and $\mathcal{B}_0 = [(\{0\}, a, \{0,1\}), (\{1\}, a, \{1,2\}), (\{2\}, a, \{2,3\}), (\{3\}, a, \{1,3\})]$. The next processed bud is dashed in the figure. Intermediate configurations are described below.

(α_0) $B = (\{3\}, a, \{1,3\})$: since state $\{1\}$ is also entered by transition $\{0\} \xrightarrow{a} \{1\}$, the latter is removed and surrogated by bud $(\{0\}, a, \{0,1\})$ (which is already in the bud stack), while state $\{1\}$ is expanded to $\{1,3\}$, accompanied by the update of the relevant bud, which becomes $(\{1,3\}, a, \{1,2,3\})$. Note how $\mathcal{D}$ is now disconnected.

(α_1) $B = (\{1,3\}, a, \{1,2,3\})$: state $\{2\}$ is expanded to $\{1,2,3\}$, causing the expansion of the relevant bud to $(\{1,2,3\}, a, \{1,2,3\})$.

(α_2) $B = (\{1,2,3\}, a, \{1,2,3\})$: state $\{3\}$ is expanded to $\{1,2,3\}$, causing the merging with the equivalent state and the generation of the new bud $(\{1,2,3\}, a, \{1,2,3\})$.

(α_3) $B = (\{1,2,3\}, a, \{1,2,3\})$: since two relevant transitions exit $\{1,2,3\}$, based on $\{1,2,3\} \xrightarrow{a} \{1,3\}$, state $\{1,3\}$ is extended to $\{1,2,3\}$, thereby causing a merging and the generation of bud $(\{1,2,3\}, a, \{2,3\})$; at this point, the second transition is $\{1,2,3\} \xrightarrow{a} \{1,2,3\}$, therefore no other expansion of state $\{1,2,3\}$ is generated.

(α_4) $B = (\{1,2,3\}, a, \{2,3\})$: no effect is produced by the processing of this bud.

(α_5) $B = (\{0\}, a, \{0,1\})$: transition $\{0\} \xrightarrow{a} \{0,1\}$ is created along with state $\{0,1\}$, and bud $(\{0,1\}, a, \{0,1,2\})$ is generated.

(α_6) $B = (\{0,1\}, a, \{0,1,2\})$: transition $\{0,1\} \xrightarrow{a} \{0,1,2\}$ is created along with state $\{0,1,2\}$, and bud $(\{0,1,2\}, a, \{0,1,2,3\})$ is generated.

(α_7) $B = (\{0,1,2\}, a, \{0,1,2,3\})$: transition $\{0,1,2\} \xrightarrow{a} \{0,1,2,3\}$ is created along with state $\{0,1,2,3\}$, and bud $(\{0,1,2,3\}, a, \{0,1,2,3\})$ is generated.

(α_8) $B = (\{0,1,2,3\}, a, \{0,1,2,3\})$: transition $\{0,1,2,3\} \xrightarrow{a} \{0,1,2,3\}$ is created.

(α_9) Since the bud stack is empty, this is the final configuration of $\mathcal{D}$.

Note how the resulting DFA in α_9 is still disconnected, although the part connected with the initial state equals the expected DFA generated by *SC* and displayed in the right side of Figure 5. As anticipated, what is disturbing in the resulting DFA is not only the disconnection, which may be removed by eventual garbage collection: the real disturbing point is the wasted processing on the disconnected part, which may cause considerable expenditure of computational resources.

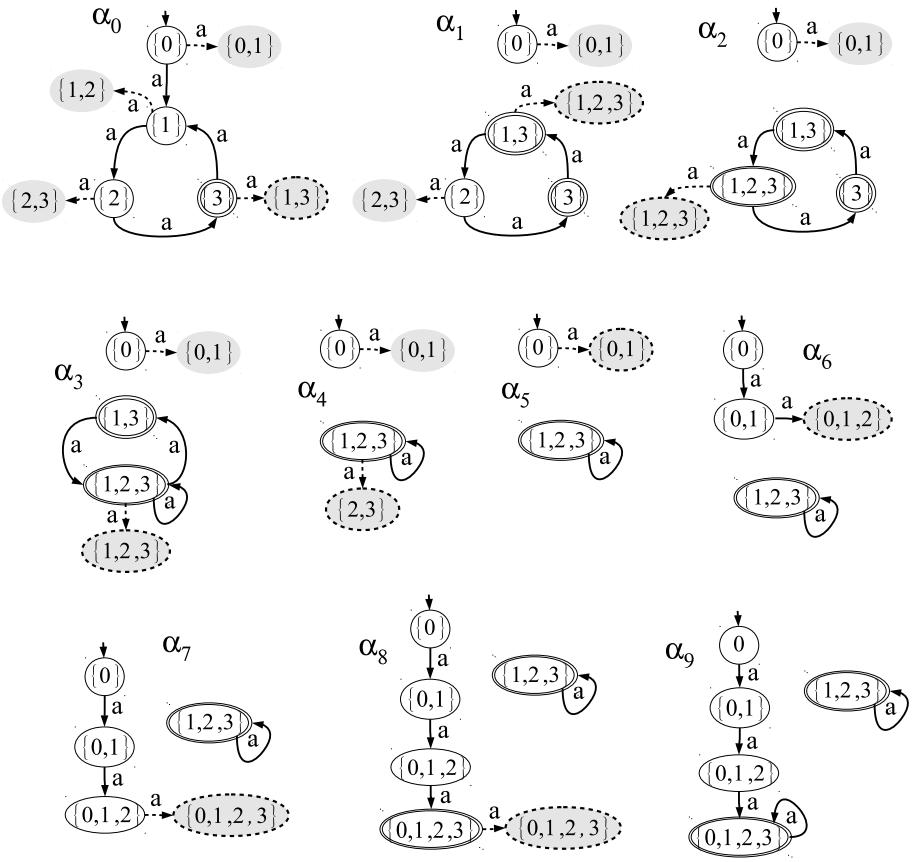

Fig. 6. Incremental determinization with disconnection.

7 Implementation and Results

Both algorithms *SC* and *ISC* were implemented in C++ on a laptop, under GNU/Linux. In the first stage of the project, we adopted a more naive approach based on classical search techniques. Before removing (redirecting) a transition entering state d' we test the essentiality of such a transition: the transition is essential if, after its removal (redirection), d' is no longer connected with the initial state. If essential, the transition is not removed (redirected); instead, all other transitions entering d' are removed and surrogated by buds. By contrast, if not essential, the transition can be removed (redirected). The problem with this technique lies in the complexity of the connectivity check: in the worst case, a complete traversing of the processed automaton is required. Early experimentation showed that an overwhelming percentage of the processing time was devoted to connectivity checking, in many cases with the result of making *ISC* and *SC* comparable, thereby nullifying altogether the advantage of incrementality. That is why we started searching for a more efficient alternative approach, which led us to

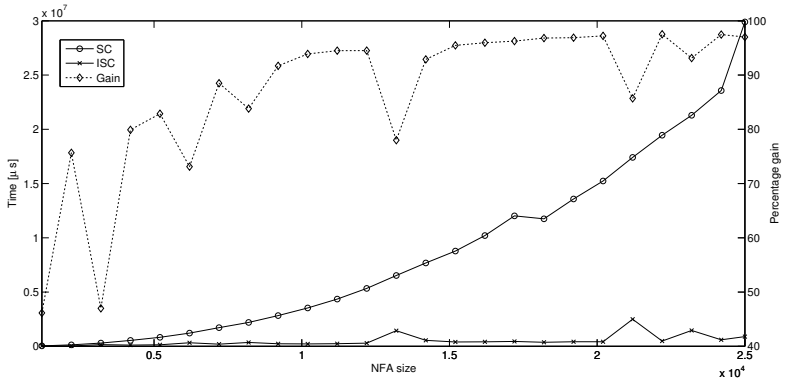

Fig. 7. Comparison between *SC* and *ISC* . Outlined in the diagram is also the gain, defined as $((time(SC) - time(ISC))/time(SC)) * 100$.

the metrics-based technique presented in this paper. Based on this new technique, we can efficiently check the inessentiality of a transition based on the distance of involved states: we removed (redirect) the transition only if it is *not* essential.

Results from subsequent experimentation based on metrics-based *ISC* show that memory allocation is equivalent in the two algorithms. Instead, in CPU time, *ISC* outperforms *SC*: the larger the NFA, the more favorable the performances of *ISC*. Hereafter we present average results, based on one reference experiment, with the following parameters: the initial NFA includes 1000 states, the alphabet includes 30 labels, and the percentage of ε-transitions is 10%. The NFA is extended up to 25000 states by 1000 states each time. Outlined in Figure 7 is the graphical representation of the comparison between *SC* and *ISC*. Besides, the *gain* is indicated (right-side y-axis), that is, the percentage of time saving when using lazy *ISC* rather than *SC*, defined as $((time(SC) - time(ISC))/time(SC)) * 100$. The gain grows with the size of the NFA: in the last determinization, the gain is 97.00% (0.89" for *ISC* vs. 29.89" for *SC*).

8 Conclusion

In contrast with the first algorithm introduced in [16], metrics-based *ISC* is sound in the sense that it generates the same DFA which is produced by *SC*, with the additional advantage of being considerably more efficient than *SC*. To do so, *ISC* exploits a metrics based on the distances of states from the initial state. This metrics allows *ISC* to efficiently check the connectivity of the processed automaton when conflicts arise in removing (or redirecting) transitions.

A goal for future research is the proof of formal correctness (termination, soundness, and completeness) of *ISC*. The extension of the incremental approach to minimization of DFAs, where the DFA equivalent to the NFA is required to include a minimal set of states at each expansion, is a further interesting research topic.

References

1. Aichernig, B., Jöbstl, E.: Efficient refinement checking for model-based mutation testing. In: 12th International Conference on Quality Software (QSIC 2012), pp. 21–30. IEEE, Xi'an, Shaanxi, China (2012)
2. Aichernig, B., Jöbstl, E., Kegele, M.: Incremental refinement checking for test case generation. In: Veanes, M., Viganò, L. (eds.) 7th International Conference on Tests and Proofs (TAP 2013), LNCS, vol. 7942, pp. 1–19. Springer, Budapest, Hungary (2013)
3. Aichernig, B., Jöbstl, E., Tiran, S.: Model-based mutation testing via symbolic refinement checking. Science of Computer Programming (2014), in Press
4. Bairoch, A., Apweiler, R.: The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000. Nucleic Acids Research 28(1), 45–48 (2000)
5. Balan, S., Lamperti, G., Scandale, M.: Incremental subset construction revisited. In: Neves-Silva, R., Tshirintzis, G., Uskov, V., Howlett, R., Jain, L. (eds.) Smart Digital Futures, Frontiers in Artificial Intelligence and Applications, vol. 262, pp. 25–37. IOS Press, Amsterdam, Netherlands (2014)
6. Brand, D., Zafiropulo, P.: On communicating finite-state machines. Journal of ACM 30(2), 323–342 (1983)
7. Cassandras, C., Lafortune, S.: Introduction to Discrete Event Systems, The Kluwer International Series in Discrete Event Dynamic Systems, vol. 11. Kluwer Academic Publishers, Boston, MA (1999)
8. Friedl, J.: Mastering Regular Expressions. O'Reilly Media, Sebastopol, CA, 3rd edn. (2006)
9. Jöbstl, E.: Model-based mutation testing with constraint and SMT solvers. Ph.D. thesis, Institute for Software Technology, Graz University of Technology, Austria (2014)
10. Lamperti, G., Scandale, M.: From diagnosis of active systems to incremental determinization of finite acyclic automata. AI Communications 26(4), 373–393 (2013)
11. Lamperti, G., Zanella, M.: Diagnosis of discrete-event systems from uncertain temporal observations. Artificial Intelligence 137(1–2), 91–163 (2002)
12. Lamperti, G., Zanella, M.: Diagnosis of Active Systems – Principles and Techniques, The Kluwer International Series in Engineering and Computer Science, vol. 741. Kluwer Academic Publishers, Dordrecht, Netherlands (2003)
13. Lamperti, G., Zanella, M.: A bridged diagnostic method for the monitoring of polymorphic discrete-event systems. IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics 34(5), 2222–2244 (2004)
14. Lamperti, G., Zanella, M.: Monitoring and diagnosis of discrete-event systems with uncertain symptoms. In: Sixteenth International Workshop on Principles of Diagnosis DX'05. pp. 145–150. Monterey, CA (2005)
15. Lamperti, G., Zanella, M.: Monitoring of active systems with stratified uncertain observations. IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans 41(2), 356–369 (2011)
16. Lamperti, G., Zanella, M., Chiodi, G., Chiodi, L.: Incremental determinization of finite automata in model-based diagnosis of active systems. In: Lovrek, I., Howlett, R., Jain, L. (eds.) Knowledge-Based Intelligent Information and Engineering Systems. LNAI, vol. 5177, pp. 362–374. Springer (2008)
17. Lamperti, G., Zanella, M., Zanni, D.: Incremental processing of temporal observations in model-based reasoning. AI Communications 20(1), 27–37 (2007)
18. Rabin, M., Scott, D.: Finite automata and their decision problems. IBM Journal of Research and Development 3(2), 114–125 (1959)
19. Tretmans, J.: Test generation with inputs, outputs and repetitive quiescence. Software – Concepts and Tools 17(3), 103–120 (1996)