Automated Error Correction of Business Process Models

M. Gambini[1], M. La Rosa[2], S. Migliorini[1], and A.H.M. ter Hofstede[2,3]

[1] University of Verona, Italy
{mauro.gambini|sara.migliorini}@univr.it
[2] Queensland University of Technology, Australia
{m.larosa|a.terhofstede}@qut.edu.au
[3] Eindhoven University of Technology, The Netherlands

Abstract. As order dependencies between process tasks can get complex, it is easy to make mistakes in process model design, especially behavioral ones such as deadlocks. Notions such as soundness formalize behavioral errors and tools exist that can identify such errors. However these tools do not provide assistance with the correction of the process models. Error correction can be very challenging as the intentions of the process modeler are not known and there may be many ways in which an error can be corrected. We present a novel technique for automatic error correction in process models based on *simulated annealing*. Via this technique a number of process model alternatives are identified that resolve one or more errors in the original model. The technique is implemented and validated on a sample of industrial process models. The tests show that at least one sound solution can be found for each input model and that the response times are short.

1 Introduction and Background

Business process models document organizational procedures and as such are often invaluable to both business and IT stakeholders. They are used to communicate and agree on requirements among business analysts, or used by solution architects and developers as a blueprint for process automation [15]. In all cases, it is of utmost importance that these models are correct. Incorrect process models can lead to ambiguities and misinterpretations, and may not be directly automated [16]. There are different types of errors. A process model can violate simple syntactical requirements (e.g. some nodes are disconnected or used improperly), or suffer from behavioral anomalies (e.g. deadlocks or incorrect completion). Behavioral errors only arise when process models are executed and thus they are typically much more difficult to spot. For these reasons, they are quite common in practice. For example, a recent analysis of more than 1,350 industrial process models reports that 54% of these models are unsound [6].

Formal correctness notions such as *soundness* [1] define behavioral anomalies for process models, and advanced process modeling tools implement verification methods based on these notions to automatically detect these anomalies in process models [6, 16]. Process modelers can take advantage of the information provided by these verification features to fix design flaws. However, generally it is up to the user to understand the (often very technical) output produced by these tools and, even worse, to figure out how to fix these errors. Correcting behavioral errors in process models is not trivial.

Apparently independent errors can have a common cause and correcting one error may introduce new errors in other parts of the model. This problem is amplified by the inherent complexity of process models, which tends to grow as organizations reach higher levels of Business Process Management (BPM) maturity [11].

This paper presents a novel technique called *Petri Nets Simulated Annealing* (PNSA) for automatically fixing unsound process models. The core of this technique is a heuristic optimization algorithm based on dominance-based Multi-Objective Simulated Annealing [14, 13]. Given an unsound process model and the output of its soundness check, at each run, the algorithm generates a small set of alternative models (i.e. "solutions") similar to the original model but containing fewer or no behavioral errors, until a maximum number of desired solutions is found or a given timeframe elapses. These solutions are produced by applying a number of controlled changes (i.e. "perturbations") on the current solution, which in turn is derived from the original model. The similarity of a solution to the original model is determined by its structural similarity and (to remain efficient) by an approximation of its behavioral similarity to the original model. Since the intentions of the process modeler are not known and there are usually many ways in which an error can be corrected, the algorithm returns several non-redundant final solutions (i.e. no solution is worse than any of the others). The differences between these solutions and the original model can then be presented to a process modeler as suggestions to rectify the behavioral errors in the original model. This technique is implemented in a prototype tool and validated on a sample of industrial process models. The results indicate that multiple errors can be fixed in a short time and at least one sound solution was found for each model.

The problem of automatically correcting errors has already been explored for software bug fixing (see e.g. [2]). Given a program, a set of positive tests and at least one failed test proving evidence for a bug, these algorithms produce a patch that fixes the error in question, provided that this error is localized. In the BPM field, [3] describes a technique for automatically fixing certain types of data anomalies that can occur in process models, while [9] presents an approach to compute the edit operations required to correct a faulty service in order to interact in a choreography without deadlocks. However, to the best of our knowledge, the problem of automatically fixing unsound process models has not been addressed yet.

In this paper, we represent process models as Workflow nets—a class of Petri nets that has been extensively applied to the formal verification of business process models [16]. In addition, mappings exist between process modeling languages used in practice (e.g. EPCs, BPMN, BPEL) and Petri nets [10]. This provides a basis to extend the results of this paper to concrete process modeling notations.

Based on the above, the rest of this paper is organized as follows: Sec. 2 provides the basic definitions of Workflow nets and soundness. Sec. 3 defines the problem in question, while Sec. 4 describes the PNSA technique in detail. The evaluation of the proposed technique is treated in Sec. 5 before Sec. 6 concludes the paper.

2 Preliminaries

Petri nets are graphs composed of two types of nodes, namely transitions and places, connected by directed arcs. Transitions represent tasks while places represent the status of the system before or after the execution of a transition. Labels are assigned to transitions to indicate the business action they perform, i.e. the observable behavior. A special label τ is used to represent invisible actions, i.e. actions that are only used for routing purposes and do not represent any task from a business perspective.

Definition 1 (Labeled Petri net). *A labeled Petri net is a tuple $N = (P, T, F, L, \ell)$ where P and T ($P \cap T = \varnothing$) are finite sets of places, resp., transitions, $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation, L is a finite set of labels representing business actions, $\tau \notin L$ is a label representing an invisible action, and $\ell : T \to L \cup \{\tau\}$ is a labeling function which assigns a label to each transition. For $n \in P \cup T$, we use $\bullet n$ and $n \bullet$ to denote the set of inputs to n (preset) and the set of outputs of n (postset).*

We are interested in Petri nets with a unique source place and a unique sink place, and such that all other nodes are on a directed path between the input and the output places. A Petri net satisfying these conditions represents a *process model* and is known as a Workflow net [1].

Definition 2 (Workflow net). *Let $N = (P, T, F, L, \ell)$ be a labeled Petri net and F^* be the reflexive transitive closure of F. N is a Workflow net (WF-net) if and only if (iff):*

- *there exists exactly one input place, i.e. $\exists!_{p_I \in P} \ \bullet p_I = \varnothing$, and*
- *there exists exactly one output place, i.e. $\exists!_{p_O \in P} \ p_O \bullet = \varnothing$, and*
- *each node is on a directed path from the input to the output place, i.e. $\forall_{n \in P \cup T} \ ((p_I, n) \in F^* \ \wedge \ (n, p_O) \in F^*)$.*

We define the universal set $\mathcal{N}$ as the set of all WF-nets. Fig. 1 shows four example Petri nets, where actions are depicted within transitions, e.g. $\ell(t_1) = a$. All these nets have single start and end places, and any transition lies on a path from the start to the end place. Hence all these nets are WF-nets.

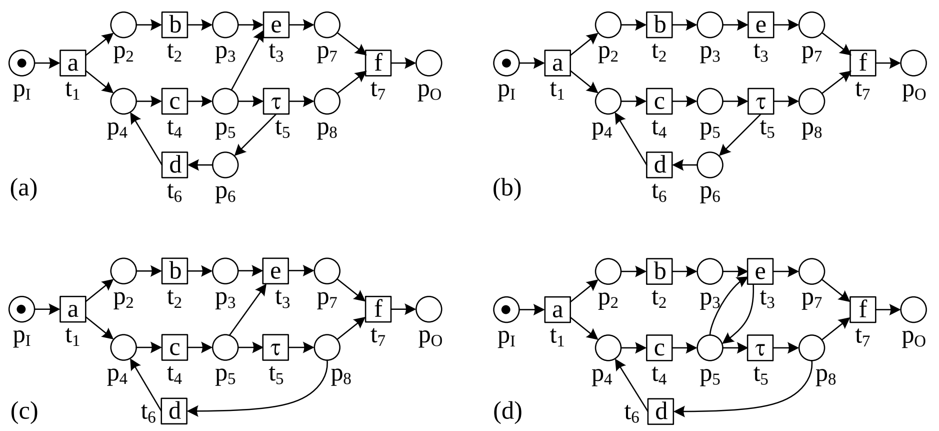

Fig. 1. Four example WF-nets.

Behavioral correctness of a WF-net is defined w.r.t. the states that a process instance can be in during its execution. A state of a WF-net is captured by the marking of its places with tokens. In a given state, each place is either empty or it contains one or more tokens (i.e. it is marked). A transition is enabled in a given marking if all the places in the transition's preset are marked. Once enabled, the transition can fire (i.e. can be executed) by removing a token from each place in the preset and putting a token into each subsequent place of the transition's postset. This leads to a new state.

Definition 3 (Marking notation). *Let $N = (P, T, F, L, \ell)$ be a WF-net. Then $M : P \rightarrow \mathbb{N}$ is a marking and $\mathcal{Q}$ is the set of all markings. Moreover:*

- *for any two markings $M, M' \in \mathcal{Q}$, $M \geq M'$ iff $\forall_{p \in P}\ M(p) \geq M'(p)$,*
- *for any two markings $M, M' \in \mathcal{Q}$, $M > M'$ iff $M \geq M'$ and $M \neq M'$.*
- *$\mathbb{M}(N)$ as the set of all markings of N,*
- *M_I^N as the* initial marking *of N with one token in place p_I, i.e. $M_I^N = [p_I]$,*
- *M_O^N as the* final marking *of N with one token in place p_O, i.e. $M_O^N = [p_O]$,*
- *for any transition $t \in T$ and any marking $M \in \mathbb{M}(N)$, t is* enabled *at M, denoted as $M[t\rangle$, iff $\forall_{p \in \bullet t}\ M(p) \geq 1$. Marking M' is* reached *from M by firing t and $M' = M - \bullet t + t\bullet$,*
- *for any two markings $M, M' \in \mathbb{M}(N)$, M' is reachable from M in N, denoted as $M' \in N[M\rangle$, iff there exists a firing sequence $\sigma = t_1.t_2 \ldots t_n$ $(n \geq 0)$ leading from M to M', and we write $M \xrightarrow{\sigma}_N M'$.*
- *$\mathsf{Tr}(N) = \{\sigma \in T^* \mid M_I \xrightarrow{\sigma}_N M\}$ is its set of* traces*, i.e. firing sequences that start from the initial marking,*
- *$\mathsf{CTr}(N) = \{\sigma \in T^* \mid M_I \xrightarrow{\sigma}_N M_O\}$ is its set of* correct traces*, i.e. traces that lead to the final marking,*
- *$\hat{\ell} : T^* \rightarrow L^*$ returns the* string of its visible actions*, where*
  $$\hat{\ell}(\sigma) = \begin{cases} \epsilon \text{ if } \sigma = \epsilon \text{ (the empty string)}, \\ \ell(t).\hat{\ell}(\sigma'), \text{ if } \sigma = t.\sigma' \text{ and } \ell(t) \neq \tau, \\ \hat{\ell}(\sigma'), \text{ if } \sigma = t.\sigma' \text{ and } \ell(t) = \tau. \end{cases}$$

In the reminder, we omit N as superscript or subscript when it is clear from the context. All the nets in Fig. 1 are marked with their initial marking, i.e. they have one token in their input place depicted as a dot inside the place.

The execution of a process instance starts with the initial marking and should then progress through transition firings until a proper completion state. This intuition is captured by three requirements [1]. First, every process instance should always have an *option to complete*. If a WF-net satisfies this requirement, it will never run into deadlocks or livelocks. Second, every process instance should eventually reach the final marking, i.e. the state in which there is one token in the output place, and no tokens are left behind in any other place, since this would signal that there is still work to be done. Third, for every transition, there should be at least one correct trace that includes at least one firing of this transition. A WF-net fulfilling these requirements is *sound* [1].

Definition 4 (Sound WF-net). *Let $N = (P, T, F)$ be a WF-net and M_I, M_O be the initial and end markings. N is sound iff:*

- *option to complete: for every marking M reachable from M_I, there exists a firing sequence leading from M to $M' \geq M_O$, i.e. $\forall_{M \in N[M_I\rangle} \exists_{M' \in N[M\rangle}\ M' \geq M_O$, and*
- *proper completion: the marking M_O is the only marking reachable from M_I with one token in place p_O, i.e. $\forall_{M \in N[M_I\rangle}\ M \geq M_O \Rightarrow M = M_O$, and*
- *no dead transitions: every transition can be reached by the initial marking, i.e. $\forall_{t \in T}\ \exists_{M \in N[M_I\rangle}\ M[t\rangle$.*

The first WF-net in Fig. 1 is not sound. This net can only complete successfully if transition t_5 fires only once before t_3 fires. In fact, if t_3 fires before t_5, t_7 will deadlock in marking $[p_7]$ waiting for a token in p_8 which will never arrive (no option to complete). Also, if t_5 fires more than once before t_3, it will put more than one token in p_8 and when t_3 and t_7 fire, the net will complete with the marking $[p_O + kp_8]$ with $k > 0$ (no proper completion). Nets (b) and (c) also suffer from behavioral problems. (b) has no proper completion in markings $[p_O + p_5 + p_8]$ and $[p_O + p_6 + kp_8]$ with $k \geq 0$, while (c) has a deadlock in $[p_7]$ and a dead transition t_7. Net (d) is sound. In the next section we focus on the problem of automatically fixing unsound process models.

3 Automatic Process Model Correction

We define the problem of automatically fixing behavioral errors in process models as the *Automatic Process Model Correction* (APMC) problem. Intuitively, given an unsound WF-net N and the output of its verification method, we want to find a set of remedial suggestions, each one presented as a minimal set of changes, that transforms N into a *sufficiently similar* WF-net N' with fewer or no behavioral errors. Since there are usually different ways to solve a behavioral error in an unsound model, we should return different alternative solutions N', such that none of these solutions is *strictly worse* than the others. However since the set of solutions can be potentially large, we should limit it to a maximum number of solutions and a maximum timeframe that the user is willing to wait to find such solutions. The user can then evaluate the solutions found to see if some of these are consistent with their initial intentions, and apply the changes accordingly.

A solution should be sufficiently similar to the original model in order to preserve the modeler's intentions. Otherwise we could obtain a sound variant simply by reducing the original model to a trivial sequence of nodes. The notion of process model similarity can be approached from a structural and from a behavioral perspective [5]. From a structural perspective, we should create models whose structure is similar to that of the original model, i.e. we should minimize the changes that we apply to the original model in terms of insertion and deletion of nodes and arcs, and control the type of these changes. From a behavioral perspective, we should preserve the observable behavior of a process model as far as possible, and meantime, try to reduce the number of behavioral errors. Also, it is preferable that a solution does not introduce new errors. Finally, none of the identified solutions should be inferior to the others in terms of number and type of errors being fixed. For example, the final set of solutions should not contain a model that fixes one error if there also exists another model in the same set that fixes that error and a second error. In the following section, we formalize our solution to this problem.

4 Petri Nets Simulated Annealing

A well-established measure for structural similarity of process models is based on *graph-edit distance* [4]. A challenge in this technique is finding the best mapping between the nodes of the two graphs to compare, which can be computationally expensive. However we avoid this problem because we measure the graph edit distance by computing the costs of the operations that we perform to change the original model into a given solution. Unfortunately, behavioral similarity cannot be computed efficiently. As an example, the *Transition Adjacency Relation* technique [17] requires the exploration of the entire state space of the two models to determine their behavioral similarity. This would be unfeasible in our case, because we would need to check the state space of each proposed solution against that of the original model, and we typically build a high number of solutions. Thus, we opt for an approximation of behavioral similarity based on two components. First, we introduce a notion of *behavioral distance* to measure the ability of a solution to simulate a finite set of correct traces of the original model. Second, we introduce a notion of *badness* to measure how many errors of the original model are still present in the solution, and how many new errors have been introduced.

In the light of this, the APMC problem becomes a multi-objective optimization problem which can be solved by trying to simultaneously minimize three *objective functions*: the structural distance, the behavioral distance and the badness of a solution model w.r.t. the original model. At each iteration of our algorithm, we search candidate solutions with lower structural distance, behavioral distance and badness w.r.t. previous solutions. We compute the behavioral distance and the badness over a sample set of traces of the original model, which we select at each iteration. If we find a solution that increases these objective functions, we discard it. Otherwise we maintain the solution so that it can be tested with further sample traces in subsequent iterations. The more tests a solution passes, the more the *confidence* increases that this is a good solution. The procedure concludes when a maximum number of *comparable* solutions with high confidence is reached (i.e. each solution is not worse than any of the others), or a given timeframe elapses. In the following subsections, we formalize the ingredients of this technique and describe its algorithm in detail.

4.1 Structural Distance

Each candidate solution is obtained by applying a minimal sequence of edit operations, i.e. an *edit sequence*, in order to *insert* or *remove* a single arc or node. Inserting or removing one arc is a single atomic operation and does not violate the properties of a WF-net, i.e. the solution will always be a WF-net. Inserting a node implies adding the node itself and one incoming and one outgoing arc to connect this new node to the rest of the net (a total of three edit operations are needed). Deleting a node implies removing the node itself and all its incoming and outgoing arcs, and can only be done if the elements in the node's preset and postset remain on a path from p_I to p_O after removing the node. This requires at least three edit operations.

Given that each edit sequence is minimal, i.e. it corresponds to the insertion or removal of one arc or node only, more complex perturbations on the original model can be obtained by applying multiple edit sequences through a number of intermediate solutions. Coming back to the example in Fig. 1, nets (b)-(f) are all solutions of (a) which can be obtained via one or more edit sequences. For example, net (b) can be obtained directly from (a) with one edit sequence consisting of one edit operation (removal of arc $p_5 - t_3$), while net (c) can be obtained with two edit sequences consisting of four edit operations in total (addition of arc $p_8 - t_6$ and removal of place p_6 with its two arcs).

While it is safe to add or remove places and arcs, we need special considerations for transitions. Since the modeler's intention is not known, we assume that a business action was introduced with a specific business purpose, so its associated transition should not be removed from the model. On the other hand, an invisible action (i.e. a τ transition) is only used for routing purposes, so we assume it can be safely removed or inserted. Thus, we only allow insertion and removal of τ transitions.

The cost of each type of edit operation can be controlled by the user. For example, one may rate removal operations as more expensive than addition operations, based on the assumption that it is less likely that modelers would introduce something erroneous than that they forgot to introduce something essential. Similarly, one may rate operations on transitions as more expensive than operations on places, and the latter as more expensive than operations on arcs.

The *structural distance* between two WF-nets is obtained by the total cost of the edit sequence between the two nets.

Definition 5 (Structural distance). *Let $N, N' \in \mathcal{N}$ be two WF-nets, E the set of all edit operations, $e(N, N') = \langle e_i \in E \rangle_{i=1}^{k}$ the edit sequence between N and N', and $c : E \rightarrow \mathbb{R}$ a function that assigns a cost to each edit operation. The structural distance $\lambda : \mathcal{N} \times \mathcal{N} \rightarrow \mathbb{R}$ between N and N' is $\lambda(N, N') = \sum_{i=1}^{k} c(e_i)$.*

Let us assume a cost of 1 for each type of edit operation. Then the structural distance between nets (b) and (a) is 1, between (c) and (a) is 4 and between (d) and (a) is 5.

4.2 Behavioral Distance

Given a sample set of traces R from the original model N, we compute the *behavioral distance* of a solution N' from N as its ability to simulate the correct traces in R, i.e. those traces that start from marking M_I and complete in M_O. A model N' can *simulate* a trace σ if its string of visible actions $\hat{\ell}(\sigma)$ can be replayed in N'. For example, given the trace $\sigma_1 = t_1.t_2.t_4.t_5.t_6.t_4.t_3.t_7$ of net (a) in Fig. 1, we want to check whether $\hat{\ell}(\sigma_1) = a.b.c.d.c.e.f$ can be replayed in a solution of (a). Precisely, we want to know to what extent $\hat{\ell}(\sigma_1)$ can be simulated by its *best simulation* trace, i.e. by the longest prefix of $\hat{\ell}(\sigma_1)$ that can be replayed in N'. For example, net (b) can fully simulate σ_1 while (c) cannot simulate action f because t_7 is dead in (c). The extent of a simulation is called *simulation ratio* and is given by $\frac{|\hat{\ell}(\sigma')|}{|\hat{\ell}(\sigma)|}$ where σ' is the best simulation of σ in a given solution. The simulation ratio of σ_1 in net (b) is 1 while in (c) it is $\frac{6}{7} \approx 0.86$.

We limit the number of silent actions that can be used to simulate a trace via a parameter m for efficiency reasons. In fact, the higher m is, the more the solution model needs to be explored to see if a given trace can be simulated (and there could be entire paths made of τ transitions that would need to be explored).

Definition 6 (Trace simulation, Best simulation). *Let N and N' be two WF-nets and $\sigma \in \mathsf{Tr}(N)$ be a trace of N. Then $Sim(N, N', \sigma, m) = \{\sigma' \in \mathsf{Tr}(N') : |\{t \in \alpha(\sigma) : l(t) = \tau\}| \leq m \wedge \hat{\ell}(\sigma') \in \mathsf{Pref}(\hat{\ell}(\sigma))\}$ is the* set of traces that can simulate *σ in N' with at most m silent actions, where $\alpha(\sigma)$ returns the alphabet of σ and $\mathsf{Pref}(s)$ returns the set of all prefixes of string s. We say that σ' is a* best simulation *of σ in N' with at most m silent actions, denoted as $sim_b(N, N', \sigma, m)$, iff $\sigma' \in Sim(N, N', \sigma, m)$ and $|\hat{\ell}(\sigma')| = \max_{\theta \in Sim(N,N',\sigma,m)} |\hat{\ell}(\theta)|$. Let $\sigma' = sim_b(N, N', \sigma, m)$ such that $M_I \xrightarrow{\sigma'}_{N'} M$. Then $M_e^{\sigma'} = M$ is its final marking and $\mathbb{M}^{\sigma'} = \{M \in \mathbb{M}(N') : \exists_{\theta \in \mathsf{Pref}(\sigma')} M_I \xrightarrow{\theta}_{N'} M\}$ is the set of all markings traversed by σ'.*

The behavioral distance of a solution w.r.t. a set of traces R is the inverse of the cumulative simulation ratio of all the best simulation traces, normalized to the number of correct traces in R.

Definition 7 (Behavioral distance to N w.r.t. R and m). *The behavioral distance $\delta : \mathcal{N} \times \mathcal{N} \times 2^{\mathsf{Tr}(N)} \times \mathbb{N} \rightarrow [0, 1]$ of N' to N w.r.t. a representative set of traces R of N, and a simulation parameter m is $\delta(N, N', R, m) = 1 - \dfrac{\sum_{\sigma \in R \cap \mathsf{CTr}(N)} \psi(N, N', \sigma, m)}{|R \cap \mathsf{CTr}(N)|}$, where the simulation ratio $\psi(N, N', \sigma, m) = \begin{cases} \dfrac{|\hat{\ell}(\sigma')|}{|\hat{\ell}(\sigma)|} \text{ such that } \sigma' = sim_b(N, N', \sigma, m) \\ 0 \text{ otherwise} \end{cases}$*

With reference to Fig. 1, let us consider $R_1 = \{(t_1.t_2.t_4.t_5.t_6.t_4.t_3.t_7), (t_1.t_4.t_5.t_6.t_4.t_2.t_3.t_7), (t_1.t_2.t_4.t_3), (t_1.t_2.t_4.t_5.t_6.t_4.t_5.t_6.t_4.t_3.t_7)\}$ where the first two traces are correct. The behavioral distance of nets (b) and (d) to (a) over the correct traces in R_1 is 0 since all such traces in R_1 can be fully simulated in (b) and (d), while the behavioral distance of (c) to (a) is $1 - \frac{0.86+0.86}{2} = 0.14$ (using at most $m = 5$ silent actions).

4.3 Badness

Broadly speaking a model is *behaviorally better* than another if it contains less behavioral errors, but not all errors are equal and errors of the same type can have a different severity. For example, a no option to complete may prevent an entire process fragment from being executed, which is far worse than having a single dead transition in a model. Thus, we need a function to rank errors based on their gravity. More precisely, given a sample set of traces R, we want to find out if and to what extent each of these traces leads to an error. In the WF-net context, a trace is proof of no option to complete if it produces a marking M such that $M \neq M_O$ and no transition can be enabled in M. A trace is proof of improper completion if it produces a marking M such that $M > M_O$, i.e. there are other marked places besides p_O. These conditions can be efficiently checked on a trace. Unfortunately it is not possible to provide evidence for a dead transition unless the whole state space is explored, which we cannot afford to do when generating solutions (exploring the entire state space is notoriously an exponential problem). However, given a trace, we can still provide a "warning" for a *potentially-dead* transition, i.e. a transition that is partially enabled in the last marking of that trace, due to some places in its preset not being marked (where the maximum number of admissible missing places is a parameter d).

Definition 8 (Potentially-dead transition). *Given a WF-net N, a marking $M \in \mathbb{M}(N)$ and a transition $t \in T$, t is* potentially-dead *in marking M, i.e. pdt(N, M, t, d) iff it holds that $0 < |\{p \in \bullet t \mid M(p) = 0\}| \leq d$ and $\exists_{p \in \bullet t}\ M(p) > 0$, where $0 < d < |\bullet t|$ denotes the* maximum number of admissible missing places.

We can now provide the classification of erroneous traces.

Definition 9 (Erroneous trace classification). *Let N be a WF-net, $\sigma \in \mathsf{Tr}(N)$ be one of its traces ($M_I \xrightarrow{\sigma} M$), and d be a parameter indicating the maximum number of admissible missing places. Then σ has:*
1. *a no option to complete error iff $M \neq M_O$ and $\nexists_{t \in T}\ M[t\rangle$*
2. *an improper completion error iff $M > M_O$*
3. *a potentially-dead transition, iff $\exists_{t \in T}$ pdt(N, M, t, d).*

We denote the set of all erroneous traces for a net N as $\mathsf{ETr}(N)$. While no option to complete and improper completion are mutually exclusive errors in a trace, a trace that suffers from either of these problems can also have a potentially-dead transition.

For each σ in R of N, the *badness* is a function measuring the severity of each error for a best simulation σ' of σ in N' (if $N \equiv N'$, the best simulation of σ is the trace itself). For example, if σ has an improper completion error while its best simulation σ' in N' does not have this error, the badness of σ' will be lower than that of σ (0 if no errors are found for that trace). Thus, while for behavioral similarity we are only interested in simulating correct traces, when computing badness we need to make sure

R contains both correct and erroneous traces of N. Correct traces in N will have a badness of 0 and can be tested against their simulations in N' to see if a new error has been introduced (the badness of the simulation trace will be greater than 0); erroneous traces in N will have a badness > 0 and can be tested against their simulations to see if the errors have diminished or disappeared (the badness of the simulation trace will be lower than that of the original trace).

The badness, denoted as β, consists of three components each measuring the severity of one error type: β_n for no option to complete, β_i for improper completion and β_d for potentially-dead transitions. Given a trace σ of N, β_n measures the probability of ending up in a deadlock while executing one of its best simulations σ' in N'. This is done by counting the number of enabled transitions that are not fired at each state traversed by σ': each such a transition provides an option to diverge from the route of σ, thus potentially avoiding the final deadlock. So the more such options there are while executing σ', the lower the probability is of ending up in the deadlock in question, and the less onerous this error is. β_n is also proportional to the complexity of σ', which is estimated by counting the number of different transitions in σ' over the total number of transitions in N'. The intuition is that an error appearing in a more complex trace is worse than the same error appearing in a simple trace. Thus, if the badness of a complex trace is higher than that of a simpler trace featuring the same error, we will prioritize the fixing of the complex trace as opposed to fixing the simple one, with the hope that as a side-effect of fixing the complex trace, other erroneous traces may be fixed. β_i measures the probability of ending up in an improper completion state while executing σ'. This is done by counting the number of places that are marked when σ' marks p_O, over the total number of places that are marked while executing σ'. β_i is also proportional to the complexity of σ'. Finally, β_d returns the probability of having dead transitions in the last state of σ'. This is done by counting how many places are not marked over the size of the preset of each transition that i) is potentially-dead in the last state of σ' and ii) is not fired in any best simulation of any trace in R. Indeed, even if a transition satisfies the potentially-dead condition, we know for sure that it is not dead if it can be fired in a best simulation of a trace in R. This measure is counteracted by the complexity of the trace, under the assumption that the more transitions are fired by σ', the less likely it is that the potentially-dead transitions in the last state of σ' need to be fired.

Definition 10 (Badness w.r.t. R, m and d). *Let N, N' be two WF-nets, R a set of traces of N, m the simulation parameter and d the maximum number of admissible missing places. Let also $V = \bigcup_{\sigma \in R} \alpha(sim_b(N, N', \sigma, m))$ be the set of all transitions of N' fired in a best simulation of a trace in R. The badness of N' w.r.t. R, m and d is $\beta : \mathcal{N} \times \mathcal{N} \times 2^{\mathsf{Tr}(N)} \times \mathbb{N} \times \mathbb{N} \to \mathbb{R}$:*

$$\beta(N, N', R, m, d) = \sum_{\sigma \in R} (w_n \beta_n(N, N', \sigma, m) + w_i \beta_i(N, N', \sigma, m) + w_q \beta_d(N, N', \sigma, m, d))$$

where w_n, w_i and w_q are the weights of each error type. For each $\sigma \in R$, given $\sigma' = sim_b(N, N', \sigma, m)$ with set of traversed markings $\mathbb{M}^{\sigma'}$ and final marking $M_e^{\sigma'}$ we define:[1]

$$\beta_n(N, N', \sigma, m) = \frac{|\alpha(\sigma')|}{|T'|} \cdot \frac{[M_e^{\sigma'}(p_O) = 0 \wedge \nexists_{t \in T} M_e^{\sigma'}[t\rangle]}{1 + (\sum_{M \in \mathbb{M}^{\sigma'}, t \in T'} [M[t\rangle]) - |\sigma'|}$$

$$\beta_i(N, N', \sigma, m) = \frac{|\alpha(\sigma')|}{|T'|} \cdot \frac{[M_e^{\sigma'}(p_O) > 0] \sum_{p \in P' \setminus \{p_O\}} [M_e^{\sigma'}(p) > 0]}{\sum_{M \in \mathbb{M}^{\sigma'}, p \in P' \setminus \{p_O\}} [M(p) > 0]}$$

$$\beta_d(N, N', \sigma, m, d) = \frac{1}{|\alpha(\sigma')|} \cdot \sum_{t \in T' \setminus V, pdt(N', M_e^{\sigma'}, t, d)} \frac{\{|p \in \bullet t \ : \ M_e^{\sigma'}(p) = 0|\}}{|\bullet t|}$$

[1] $[x]$ returns 1 if the boolean formula x is true, or 0 otherwise

Let us consider again set R_1 used in Sec. 4.2 for net (a) and let us assume $m = 5$ and $d = 2$. We recall that net (a) has a deadlock in state $[p_7]$. This can be obtained by firing $\sigma_3 = t_1.t_2.t_4.t_3 \in R_1$ producing a badness of 0.29. This error is corrected in net (b). In fact the best simulation of σ_3 in (b) completes in state $[p_5 + p_7]$ which enables t_5. Its badness is thus 0 (there are no potentially-dead transitions in this state with $d = 2$). So we can infer that (b) improves (a) w.r.t. σ_3. On the other hand, since the deadlock still remains in (c), the badness for σ_3 in (c) is 0.41. This badness is higher than that of (a) since t_7 can never be executed in (c). So we can infer that (c) worsens (a) w.r.t. σ_3.

The no proper completion of (a) in $[p_O + p_8]$, obtained e.g. by firing $\sigma_4 = t_1.t_2.t_4.t_5.t_6.t_4.t_5.t_6.t_4.t_3.t_7$ with badness 0.12, is best simulated in (b) by a trace completing in $[p_O + p_5 + p_8]$. Since this state marks two places besides p_O instead of one, it induces a badness of 0.17 which is worse than that of σ_4. So (b) worsens (a) w.r.t. σ_4. For net (c), σ_4 can be best simulated with a trace completing in state $[p_7]$ with a badness of 0.28. In fact, while there is no improper completion error (the trace does not even mark p_O), there is still a deadlock in $[p_7]$ and t_7 is dead. So (c) also worsens (a) w.r.t. σ_4. Since (d) is sound, its badness for the above traces is 0.

Finally, both (b) and (c) worsen (a) w.r.t. σ_1 and σ_2 (the correct traces of R_1 shown in Sec. 4.2), as they introduce new improper completion, resp., no option to complete errors. The overall badness of (a), (b), (c) and (d) w.r.t. R_1 is 0.41, 0.53, 1.19 and 0.

4.4 Dominance-based Simulated Annealing

Given an erroneous model $N \in \mathcal{N}$, the goal of the PNSA technique is to produce a good set of solutions $\mathcal{S} \subseteq \mathcal{N}$ such that each model $N_i \in \mathcal{S}$ is similar to N but contains fewer or no errors. $\mathcal{S}$ can be considered good if i) its members are good solutions according to the three objective functions (structural distance, behavioral distance and badness); ii) they are *not redundant*, i.e. no member is better than the others; and iii) all have *high confidence*, i.e. they have been tested against a given number of sets R.

In order to find good solutions while avoiding redundancy in the final solution set, we need to be able to compare two solutions. To do so, we use the values of their objective functions. First, we need to group these functions into a *unique objective function* w.r.t. a set R. Since a solution can be tested against multiple sets R, we also need a notion of *average unique objective function* over the various R.

Definition 11 (Unique objective function). *Let N, N' be two WF-nets. Assuming N and the simulation parameter m are fixed, we define the following objective functions w.r.t. a set of traces R of N: $f_1(N', R) = \lambda(N, N')$, $f_2(N', R) = \delta(N, N', R, m)$ and $f_3(N', R) = \beta(N, N', R, m)$. These functions are grouped into a* unique objective function $\bar{f} : \mathcal{N} \times 2^{\mathsf{Tr}(N)} \rightarrow \mathbb{R}^3$ *such that for each N' and R, $\bar{f}(N', R)$ identifies the triple $(\lambda(N, N'), \delta(N, N', R, m), \beta(N, N', R, m))$. Given i sets R_k with $1 \leq k \leq i$, we compute the* average unique objective function $\bar{f}_{avg}(\{\bar{f}^k(N', R_k)\}_{k=1}^{i}) = (avg_{1 \leq k \leq i}(f_1(N', R_k)), avg_{1 \leq k \leq i}(f_2(N', R_k)), avg_{1 \leq k \leq i}(f_3(N', R_k)))$.

Considering our example set R_1, the values of the unique objective functions for nets (a-d) are: (0,0,0.41), (1,0,0.53), (4,0.14,1.19) and (5,0,0). The (average) unique objective function can be used to compare two solutions via the notion of *dominance*. A solution N_1 dominates a solution N_2 iff N_1 is better than N_2 in at least one objective function and equivalent in the remaining ones.

Definition 12 (Dominance). *Given a set of traces R, a solution N_1* dominates *a different solution N_2 w.r.t. R , i.e. $N_1 \prec_R N_2$, iff $\bar{f}(N_1, R) \leq \bar{f}(N_2, R)$ and there exists a $1 \leq j \leq 3$ such that $f_j(N_1, R) < f_j(N_2, R)$. If N_1 does not dominate N_2, we write $N_1 \not\prec_R N_2$. When we have multiple sets R, we compute the* average dominance *by comparing the average unique objective functions of two solutions, and denote this relation with $\prec_{avg}$.*

The dominance relation establishes a partial order and two solutions N_1 and N_2 are *mutually non-dominating* iff neither dominates the other. In our example, net (a) dominates both (b) and (c), (b) dominates (c), while (d) is mutually non-dominating with all other nets: (d) has higher structural distance than (a-c) although its behavioral similarity and badness are lower than those of the other nets.

A *Pareto-set* is the set of all mutually non-dominating solutions w.r.t. the average unique objective functions of the solutions, i.e. a set of non-redundant solutions.

Definition 13 (Pareto-set). *Two solutions $N_1, N_2 \in \mathcal{N}$ are* mutually non-dominating *w.r.t. a set of traces R iff $N_1 \not\prec_R N_2$ and $N_2 \not\prec_R N_1$. Similarly, N_1 and N_2 are* mutually non-dominating on average *w.r.t. a number of sets of traces, iff $N_1 \not\prec_{avg} N_2$ and $N_2 \not\prec_{avg} N_1$. A* Pareto-set *is a set $\mathcal{S} \subseteq \mathcal{N}$ such that for all solutions $N_1, N_2 \in \mathcal{S}$, $N_1 \not\prec_{avg} N_2$ and $N_2 \not\prec_{avg} N_1$.*

A *Pareto-optimum* is a solution that is not dominated by any other solution. The set of Pareto-optima is called the *Pareto-front*.

Definition 14 (Pareto-optimum, Pareto-front). *A* Pareto-optimum *$N_1 \in \mathcal{N}$ is a solution for which no $N_2 \in \mathcal{N}$ exists such that $N_2 \prec_{avg} N_1$. A* Pareto-front *$\mathcal{G}$ is the set of all Pareto-optima.*

Our PNSA technique is inspired by dominance-based Multi-Objective Simulated Annealing (MOSA) [13, 14]: a robust technique for solving multi-objective optimization problems. At the core of the MOSA technique is an optimization procedure called *simulated annealing* [12]. The term “simulated annealing” derives from the “annealing” process used in metallurgy: the idea is to heat and then slowly cool down a metal so that its atoms reach a low-energy, crystalline state. At high temperatures atoms are free to move around. However, as the temperature lowers down, their movements are increasingly limited due to the high-energy cost of movement. By analogy with this physical process, each step of the annealing procedure replaces the current solution by a random “nearby” solution with a probability that depends both on the difference between the corresponding objective values and a global parameter *Temp* (the *temperature*), that is gradually decreased during the process. The dependency is such that the current solution changes almost randomly when *Temp* is high, but increasingly less as *Temp* goes to zero. Allowing “uphill” moves potentially saves the method from getting stuck at local optima, which is the main drawback of greedy algorithms. The goal is thus to move towards the Pareto-front while encouraging the diversification of the candidate solutions. It has been shown that simulated annealing can be more effective than exhaustive enumeration when the goal is to find an acceptably good solution in a fixed amount of time, rather than the best possible solution [12].

In order to escape from local optima, and in-line with dominance-based MOSA techniques, we do not simply compare two solutions based on their dominance relation. This in fact would exclude a candidate solution N_2 that based on the current set of sample traces R is dominated by the current solution N_1 even if globally N_2 may be better than N_1. Rather, we use a notion of *energy*. The energy of a solution measures the portion of the front that dominates that solution. Thus, the lower the energy of a solution is, the better the solution is. Unfortunately, the true Pareto-front $\mathcal{G}$ is unavailable during an optimization procedure. To obviate this problem, the energy function is computed based on a finite approximation of the Pareto-front $G \subseteq \mathcal{N}$, called *estimated Pareto-front*. G is built incrementally based on the Pareto-set $\mathcal{S}$ under construction. Thus, G is initially empty and incrementally populated with new values as long as new mutually non-dominating solutions are added to $\mathcal{S}$ during the annealing procedure.

Definition 15 (Energy). *Let R be a set of traces and $G \subseteq \mathcal{N}$ be a finite estimation of the Pareto-front $\mathcal{G}$. The* energy *of a WF-net N' w.r.t. R and G is $E(N', R, G) = |\{N'' \in G : N'' \prec_R N'\}|$.*

Having defined the notion of energy, we can use this to compare two solutions in the annealing procedure. A candidate solution N_2 is accepted in place of the current solution N_1 on the basis of their *energy difference* Δ_E w.r.t. R and an estimated Pareto-front, i.e. the difference in the number of solutions in the estimated Pareto-front that dominate N_1 and N_2 w.r.t. R.

Definition 16 (Energy difference). *Let N_1, N_2 be two WF-nets, R be a set of traces and G be the finite estimation of $\mathcal{G}$. Given the set $\tilde{G} = G \cup \{N_1, N_2\}$, the* energy difference *between N_1 and N_2 w.r.t. R and G is $\Delta_E(N_2, N_1, R, G) = \dfrac{E(N_2, R, \tilde{G}) - E(N_1, R, \tilde{G})}{|\tilde{G}|}$.*

A candidate solution N_2 that is dominated by one or more members of the current estimated Pareto-front, may still be accepted with a probability equal to $min\,(1, exp\,(-\Delta_E(N_2, N_1, R, G)/Temp(i)))$ where $Temp(i)$ is a monotonically decreasing function indicating the temperature for the iteration i of the annealing procedure. In Def. 16, the inclusion of N_1 and N_2 in set $\tilde{G}$ yields a negative Δ_E if $N_2 \prec_R N_1$. This ensures that candidate solutions that move the estimated front towards the true front are always accepted. The division by $|\tilde{G}|$ also ensures that Δ_E is always less than 1, and provides some robustness against fluctuations in the number of solutions in G. A further benefit of Δ_E is that, while fostering convergence to the front, it also fosters its wide coverage. For example, let us assume we only have two objective functions f_1 and f_2. Fig. 2 depicts the objective values of two solutions, N_1 and N_2 (represented as empty circles) in relation to the values of the Pareto-front and its estimation G. N_1 and N_2 are mutually non-dominating (N_1 is better than N_2 along f_2 but N_2 is better along f_1). However N_1 is dominated by fewer elements of G than N_2 (2 instead of 4). Thus, N_1 has lower energy and would be more likely accepted in place of N_1.

Let us assume an estimated Pareto-front of $\{(0, 0, 0.41), (5, 0.12, 0.23), (9, 0, 0), (11, 0, 0.71)\}$ for our working example. Accordingly, net (b) has a lower energy than (c) so it has a higher probability of being accepted than (c). In turn, despite (d) is mutually non-dominating with both (b) and (c), it has a lower energy than these two nets, so it has a higher probability of being accepted in place of them.

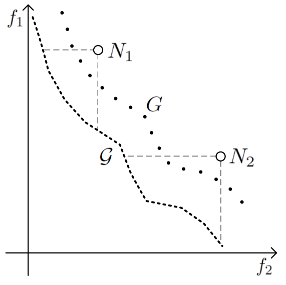

Fig. 2. Two solutions N_1 and N_2 and their energy.

At each iteration i of the annealing procedure we test the solutions for a random set of traces R_i of the original model. If at any iteration a candidate solution N_2 has the same energy as the current one N_1, N_2 has probability of

1 of being chosen ($\Delta_E = 0$). However, we prefer to keep the current solution since this has also been tested against some other sets $R_{j<i}$. This is captured by the notion of *confidence* of a solution, which indicates how many annealing iterations a solution has survived through. The more iterations a solution survives through, the more the confidence increases that this is a good solution.

We now have all ingredients to present the PNSA algorithm. The PNSA algorithm consists of multiple runs of the annealing procedure so as to incrementally construct a Pareto-set formed by mutually non-dominated solutions on average, with the same high confidence. At each run the produced solutions are exploited to feed the estimated Pareto-front, which in turn is used to compare solutions based on their energy difference. The PNSA algorithm terminates when a given timeframe *tf* elapses or a maximum number of solutions s is found, and returns the Pareto-set. The algorithm also requires as input the original model N, a finite representation of its traces $\overline{\mathsf{Tr}}(N)$ and of its erroneous traces $\overline{\mathsf{ETr}}(N)$, the desired confidence c of a solution, the maximum number of iterations o for each run of the annealing procedure, a temperature $\mathit{Temp}(i)$ decreasing at each iteration i, the maximum size k of each set R_i, and the parameters used for behavioral distance and badness (see Sec. 4.2 and 4.3).

The annealing procedure invoked at each run of the PNSA algorithm requires as input the Pareto-set $\mathcal{S}$ of the current solutions (initially empty), and uses it to create the estimated Pareto-front G. Then, at each iteration the procedure creates a perturbation of a solution randomly drawn from $\mathcal{S} \cup \{N\}$, and compares their energy difference w.r.t. a random set of traces R_i of N and G. Based on the resulting probability, one of the two solutions is added to a priority list of current solutions *LS* ordered by decreasing confidence. The procedure terminates when the maximum number of iterations o is reached or when the first member of *LS* has confidence c, and produces as output the first element of *LS* if this has confidence equal to c. This solution is added to $\mathcal{S}$ for the next run of the PNSA algorithm if it is non dominated on average by any element currently in $\mathcal{S}$. If so, the elements of $\mathcal{S}$ that are dominated on average by the solution being added are removed to ensure that all elements of $\mathcal{S}$ are always mutually non-dominated on average. The average dominance is computed by keeping for each solution N' an archive $A_{N'}$ storing the values of the unique objective functions of N' for all sets R_i used for testing N'. The steps of the PNSA algorithm are:

1. Initialize the estimated Pareto-front G with the Pareto-set $\mathcal{S}$ and empty the solution list A (in the initial run, $\mathcal{S} = \varnothing$).
2. Randomly draw a solution N_1 from $\mathcal{S} \cup \{N\}$ and set N_1 as the current solution (in the initial run, $N_1 \equiv N$).
3. Generate a random perturbation $N_2 \in \mathcal{N}$ of N_1 via a minimum sequence of edit operations.
4. Randomly draw a set $R_i \subseteq \overline{\mathsf{Tr}}(N)$ of size k such that $|\overline{\mathsf{ETr}}(N) \cap R_i| = k/2$ if $|\overline{\mathsf{ETr}}(N)| > k/2$ or $\overline{\mathsf{ETr}}(N) \subset R_i$ if $|\overline{\mathsf{ETr}}(N)| \leq k/2$.
5. Compute the unique objective functions of N_1 and N_2 w.r.t. R_i and add their values to the respective archives A_{N_1} and A_{N_2}.
6. Compute the energy difference $\Delta_E(N_2, N_1, R_i, G)$ between N_2 and N_1 using the estimated Pareto-front G.
7. If $\Delta_E(N_2, N_1, R_i, G) \neq 0$ replace the current solution N_1 with N_2 with a probability equal to $min\,(1, \mathit{exp}\,(-\Delta_E(N_2, N_1, R_i, G)/\mathit{Temp}(i)))$. Otherwise discard N_2 and increase the confidence of N_1 by 1.
8. If N_2 is accepted in place of N_1, set N_2 as the current solution with confidence 1, and add N_2 to *LS*.

9. Repeat from Step 3 while the confidence of the current solution is less than c and the maximum number of iterations o is not reached.
10. Add the current solution to *LS*. If the first element of *LS*, ls_1, has confidence at least c, compute its average unique objective function $\bar{f}_{avg}(ls_1, A_{ls_1})$ based on its archive A_{ls_1}. Add ls_1 to $\mathcal{S}$ if ls_1 is not dominated on average by any element of $\mathcal{S}$, and remove all elements of $\mathcal{S}$ that are dominated on average by ls_1.
11. Repeat from Step 1 until the timeframe *tf* elapses or $|\mathcal{S}| = s$.

The complexity of each annealing iteration is dominated by Steps 3, 5 and 6 (the rest is achieved in constant time). Step 3 computes a perturbation, which is linear on the size of the WF-net to be changed (the net is explored depth-first to check if a node can be removed; node/arc insertion and arc removal are achieved in constant time). Step 5 entails the computation of the objective functions. Computing the structural similarity is linear on the number of edit operations used in the perturbation, which in turn is bounded by the size of the WF-net. For the behavioral similarity and badness we use a depth-first search on the WF-net, which is linear on the product of i) the sum of the lengths of the traces in R to be simulated, ii) the maximum number of silent actions we can use to simulate a trace, and iii) the size of the WF-net. Step 6 entails the computation of the energy difference, which is done in logarithmic time on the size of the estimated Pareto-front [13]. So each annealing iteration can be executed efficiently.

5 Experimental Results

We implemented the PNSA algorithm in a prototype Java tool.[2] This tool imports an unsound WF-net in LoLA format,[3] builds its state-space using Karp-Miller's approximation [8] and generates a finite number of traces with at least one sample trace per error. The construction of the state-space is done in a way to explore as many distinct transitions as possible in the shortest number of states (in order to identify dead transitions). The result of the soundness check and the user parameters trigger the PNSA algorithm, which produces as output a set of solutions in LoLA format. The number of solutions and the maximum response time are limited by the input parameters.

To evaluate the feasibility of our technique, we used the tool to fix a sample of 152 unsound nets drawn from the BIT process library [6]. This library contains 1,386 models in five collections (A, B1, B2, B3, C), out of which 744 are unsound. We converted these 744 models into WF-nets and filtered out those models that after the conversion introduced new (artificial) errors. As a result, we obtained 152 models none of which from collection C. The maximum number of solutions was set to 6 and after the experiment each solution was checked for soundness. The tests were conducted on a PC with a 3GHz Intel Dual-Core x64, 3GB memory, running Microsoft Windows 7 and JVM v1.5. Each test was run 10 times by using the same random seed (to obtain deterministic results) and the execution times were averaged. The results are reported in Table 1.

The average error-reduction rate is 76.6% (avg errors in the solutions/avg errors in the input model). This indicates that the algorithm is able to fix most errors in the input models. Moreover, the structural distance of the solutions is very low (4.34 on average using a cost of 1 for each edit operation). Thus, the solutions are very similar to

[2] Available at `www.apromore.org/tools`
[3] `www.informatik.uni-rostock.de/~nl/wiki/tools/lola`

Input models				**Solutions**					
Collection	**No. models**	**Avg/Max nodes**	**Avg/Max errors**	**Avg/Max nodes**	**Avg/Max errors**	**Error reduction**	**Sound models**	**Avg/Max str. distance [cost]**	**Avg/Max Time [s]**
BIT A	48	46.54 / 129	2.21 / 5	45.85 / 129	0.74 / 20	73.6%	85.7%	4.25 / 39	16.52 / 171.43
BIT B1	22	29.55 / 87	2.55 / 6	27.18 / 87	1.07 / 29	75.6%	82.9%	5.02 / 91	8.75 / 100.45
BIT B2	35	22.86 / 117	2.54 / 9	21.79 / 118	0.69 / 9	84.9%	87.3%	3.47 / 47	12.08 / 217.91
BIT B3	47	20.38 / 73	2.77 / 9	21.42 / 118	1.26 / 24	72.4%	79.6%	4.62 / 61	10.85 / 156.93

Table 1. Experimental results.

the original model. These solutions are obtained with an average response time of 11s. This time is very short if compared to the time that would be required by a modeler to manually fix one such a process model (average size of 30 nodes). Despite the large response time in some outlier cases (218s), most solutions are behaviorally better than their input models (83.9% are sound) and at least one sound solution was found for each input model. Very few cases are worse than the input model (e.g. 29 errors instead of 6 errors in the input model). This is due to the fact that we did not fine-tune the annealing parameters based on the characteristics of each input model (e.g. if the number of annealing iterations is too low w.r.t. the number of errors in the input model, a solution could contain more errors than the input model).

6 Conclusion

This paper contributes a technique for automatically fixing behavioral errors in process models. Given an unsound WF-net and the output of its soundness check, we generate a set of alternative nets containing fewer or no behavioral errors, until a given number of desired solutions is found or a timeframe elapses. Each solution is i) sufficiently similar to the original model, ii) non-redundant in terms of fixed errors, and iii) optimal with high-confidence, i.e. it must pass a number of tests. Moreover, there is no restriction on the type of unsound WF-net that can be fixed (e.g. acyclic, non-free choice).

The core of this technique is a heuristic optimization algorithm based on dominance-based MOSA. The choice of this algorithm is dictated by the fact that the modeler's intentions are unknown, which determines the fuzziness of the result. Also, an important advantage of MOSA over greedier algorithms is that while converging to optimal solutions, it encourages the diversification of the candidate solutions. In turn, this allows the algorithm to escape from local optima, i.e. solutions that minimally improve the original model. Our adaptation of MOSA to the problem of fixing unsound process models uses three objective functions to drive the selection of candidate solutions. These functions measure the similarity of a solution to the unsound model and the severity of its errors. Moreover, we embed a notion of *confidence* to increase the reliability of a solution, given that this is tested against different sample traces of the unsound model at each iteration of the algorithm. Clearly, more sophisticated metrics could be employed in place of our objective functions, e.g. to identify dead transitions. However one has to strike a trade-off between accuracy and computational costs. Indeed, we compute our objective functions in linear time.

We prototyped our technique in a tool and validated it on a sample of industrial process models. While we cannot guarantee that the returned solutions are always sound, we found at least one sound solution for each model used in the tests, and the response times were short. This is definitely an improvement compared to manual correction.

More generally, our work can be seen as a modular framework for improving business process models. In fact one could plug in other objective functions to serve different purposes, such as fixing non-compliance issues or increasing process performances.

There are several interesting avenues for future work. First, the randomness of the perturbations could be controlled by exploiting *crossover* techniques from genetic algorithms [7]. The idea is to obtain a new perturbation by combining correct (sub-)traces from each solution in the current Pareto-set. Second, in order to further accelerate the identification of optimal solutions, the energy resolution could be increased by using *attainment surface sampling* techniques [13]. This would compensate for the small size of the estimated Pareto-front at the beginning, which yields coarse-grained comparisons of solutions. Third, the structural features of the unsound model and the result of its soundness check could be exploited to estimate the annealing parameters (e.g. the number of annealing iterations could depend on the number of errors found in the unsound model). Finally, we plan to evaluate the quality of the proposed changes with modeling experts and use the results to train the algorithm so as to discard certain perturbations.

References

1. W.M.P. van der Aalst, K.M. van Hee, A.H.M. ter Hofstede, N. Sidorova, H.M.W. Verbeek, M. Voorhoeve, and M.T. Wynn. Soundness of Workflow Nets: Classification, Decidability, and Analysis. *Formal Aspects of Computing*, 2011.
2. A. Arcuri. On the automation of fixing software bugs. In *ICSE*, 2008.
3. A. Awad, G. Decker, and N. Lohmann. Diagnosing and repairing data anomalies in process models. In *BPM Workshops*, 2009.
4. H. Bunke and G. Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245 – 253, 1983.
5. R. Dijkman, M. Dumas, B. van Dongen, R. Kaarik, and J. Mendling. Similarity of business process models: Metrics and evaluation. *Information Systems*, 36(2), 2011.
6. D. Fahland, C. Favre, B. Jobstmann, J. Koehler, N. Lohmann, H. Völzer, and K. Wolf. Instantaneous soundness checking of industrial business process models. In *BPM*, 2009.
7. J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan, 1975.
8. R.M. Karp and R.E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2), 1969.
9. N. Lohmann. Correcting deadlocking service choreographies using a simulation-based graph edit distance. In *BPM*, 2008.
10. N. Lohmann, E. Verbeek, and R.M. Dijkman. Petri net transformations for business processes – a survey. *TOPNOC*, 2:46–63, 2009.
11. H.A. Reijers, R.S. Mans, and R.A. van der Toorn. Improved Model Management with Aggregated Business Process Models. *Data Knowl. Eng.*, 68(2):221–243, 2009.
12. C. Gelatt S. Kirkpatrick and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
13. K.I. Smith, R.M. Everson, J.E. Fieldsend, C. Murphy, and R. Misra. Dominance-based multiobjective simulated annealing. *IEEE Trans. on Evolutionary Computation*, 12(3), 2008.
14. B. Suman. Study of simulated annealing based algorithms for multiobjective optimization of a constrained problem. *Computers & Chemical Engineering*, 28(9), 2004.
15. M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag, Berlin, 2007.
16. M.T. Wynn, H.M.W. Verbeek, W.M.P. van der Aalst, A.H.M. ter Hofstede, and D. Edmond. Business process verification: Finally a reality! *BPM Journal*, 15(1):74–92, 2009.
17. H. Zha, J. Wang, L. Wen, C. Wang, and J. Sun. A workflow net similarity measure based on transition adjacency relations. *Computers in Industry*, 61(5), 2010.