Acta Cybernetica **17** (2005) 225–245.

Optimal Trajectory Generation for Petri nets*

Szilvia Gyapay† and András Pataricza†

Abstract

Recently, the increasing complexity of IT systems requires the early verification and validation of the system design in order to avoid the costly redesign. Furthermore, the efficiency of system operation can be improved by solving system optimization problems (like resource allocation and scheduling problems). Such combined optimization and validation, verification problems can be typically expressed as reachability problems with quantitative or qualitative measurements. The current paper proposes a *solution to compute the optimal trajectories for Petri net-based reachability problems with cost parameters.* This is an improved variant of the basic integrated verification and optimization method introduced in [11] combining the efficiency of *Process Network Synthesis* optimization algorithms with the modeling power of *Petri nets.*

1 Introduction

As the quality of service delivered by IT systems becomes more and more crucial to production and the life of the society, their correct and efficient operation has to be proved already during the design phase. Validation and verification methods are known to assure the correctness of the services, while optimization may serve to calculate the quantitative performance characteristics of a system by estimating its quantitative boundaries and to minimize operation costs.

The fulfillment of the requirements have to be addressed already during the early design phases in order to avoid costly redesigns. Recently, system designers use the *Unified Modeling Language (UML)* that became the standard object-oriented modeling language since it provides a semi-formal, concise description of complex systems including the means to model both its static and dynamic behavior. UML can be extended to incorporate quantitative measures, requirements, and constraints.

While UML is a proper means for system and requirements modeling the analysis itself has to be carried with the help of some mathematical model analysis

*This work was supported by project OTKA T038027.

†Budapest University of Technology and Economics, Department of Measurement and Information Systems, Magyar tudósok körútja 2., H-1117, Budapest, Hungary. Phone: +36-1-463-3579, +36-1-463-3595 Fax: +36 1 463-2667 E-mail:{gyapay,pataric}@mit.bme.hu

tool. Recent research efforts aim at an automated transformation from UML models to mathematical analysis tools [27]. One of the most challenging problems in mathematical analysis of IT systems is the simultaneous analysis of the dynamical behavior of the system and its impact to the quantitative measures as it meets a combination of a mathematical paradigm describing the control logic of the application and the quantitative impact of it.

Petri nets are used in the design of IT systems either as a direct modeling paradigm or derived from engineering models. They describe the system logic, i.e. they define the potential behavior of the system by identifying the trajectories in the system state space in a compact form. However, the requirement is frequently not only to provide a correct behavior of the system, but to perform some functionality in an optimal way. Therefore, Petri nets first need to be extended with cost or time parameters. Then the reachability problem can be formulated as an *optimal trajectory problem*: to derive the optimal trajectory from the initial state to a given state. Unfortunately, the optimal trajectory problem has been solved only for rather restricted classes of Petri nets without practical relevance.

Our overall objective is to derive from the UML specification of the application and to estimate the worst case characteristics of a system upon *temporal constraints* on its operation sequence. (i) At first, an automated transformation on attributed graph grammars maps the UML models into Petri nets (discussed in [10]). (ii) Secondly, a basis of all firing sequences representing possible operations (trajectories from an initial state to a given state that satisfy user-defined constraints) are estimated as a basis spanning the state space of the feasible solutions to the optimization objective. (iii) The next step is to compute a candidate trajectory that represents an optimal operation by combining trajectories generated in the previous step. (iv) In order to ensure the executability (fireability) of the candidate trajectory, post-filtering is executed by model checking (exhaustive simulation).

The contribution of the current paper compared to previous ones [11, 12] is the improved efficiency of the algorithm originating in (i) the basis-based generation of a candidate trajectory instead of using the Accelerated Branch and Bound algorithm, and (ii) a new algorithm to compute the reachability–membership function that does not require the explicit enumeration of the state space of the Petri net.

The paper is organized as follows. An introduction is given in Section 2 to Petri nets and Process Network Synthesis. The optimal trajectory problem is defined for Petri nets extended with cost parameters in Section 3. Section 4 recalls the assignments of the elements and problem semantics of the two paradigms followed by the analysis of the problems related to the direct algorithm adaptation and we recall the integrated technique in [11] to solve the Petri net-based optimization problem emphasizing the new methods for the generation of the candidate optimal trajectory in Section 5 and the reachability–membership function in Section 6. Section 7 provides an overview of related works, and finally, Section 8 reports on initial results and concludes our work.

2 Basic Notations

Hereafter only the notations and definitions of Petri nets and PNS problems necessary for understanding the concept of the solution (for more details see e.g., [20] and [7]) will be recalled.

2.1 Petri Nets

A *Petri net* is a directed, bipartite graph represented by a four-tuple $PN = \langle P, T, w, M_0 \rangle$, where P and T are the disjoint sets of place and transition nodes, respectively. Places may contain tokens, whose distribution represents the state of the net. The state of the net is described by the so-called marking. A marking is a $|P|$-dimensional vector over the naturals $\mathbb{N}$, where the i-th component $(M(p_i))$ is the number of tokens contained in $p_i \in P$. M_0 denotes the marking of the net in the initial state.

A Petri net with four places and three transitions is depicted on the left in Fig. 1 with the initial marking $M_0 = (2, 3, 0, 0)$.

The state of the net is changed by transition firings. The token flow is denoted by the weight function w assigning positive integers to the directed arcs between places and transitions $w : (P \times T) \cup (T \times P) \to \mathbb{N}$.

Let $\bullet x$ and $x\bullet$ denote the pre-set and the post-set of an element $x \in P \cup T$, respectively, s.t., $\bullet x = \{y : w(y, x) > 0\}$ and $x\bullet = \{y : w(x, y) > 0\}$.

A t transition is *enabled* at marking M (i.e. it may fire), if all the places in its pre-set contain at least as many tokens as required by the weight of the corresponding arc, i.e. if $\forall p \in \bullet t : w(p, t) \leq M(p)$ holds. The firing of the transition passes (removes and produces) the defined number of tokens (according to the weight function) from its input places ($\bullet t$) to its output places ($t\bullet$). Thus, the marking in the successor state M' can be estimated as $\forall p \in P : M'(p) = M(p) - w(p, t) + w(t, p)$.

Let us define the $|P| \times |T|$-dimensional *incidence matrix* W of the Petri net showing the change in the number of tokens in each place caused by the firing of the individual transitions, i.e. $W[p, t] = [-w(p, t) + w(t, p)], p \in P, t \in T$. Thus, $\forall p \in P : M'(p) = M(p) - w(p, t) + w(t, p) = M(p) + W(p, t)$, and $M' = M + W \cdot e_{t_j}, t_j = t$, where $e_{t_j} = (0, \ldots, 0, 1, 0, \ldots, 0)$ is a $|T|$-dimensional unit vector with 1 in its j-th component.

The effect of firing a transition can be extended for a firing sequence $s = \langle t_{i_1}, t_{i_2}, \ldots \rangle, 1 \leq i_k \leq |T|, k = 1, 2, \ldots$. The so-called *transition occurrence vector* (the Parikh vector of the sequence) of the firing sequence is a $|T|$-dimensional vector σ_s, where its jth component counts the number of firing transition t_j (i.e. $\sigma_s[j] = |t_{i_k} \in s : i_k = j|$. Then the firing sequence s starting from M_0 leads to marking M defined by the following so-called *state equation*.

$$M = M_0 + W \cdot \sigma_s. \tag{1}$$

Another mathematical paradigm, Process Network Synthesis was investigated in order to develop efficient search methods for the optimization of Petri net sequences with quantitative parameters.

2.2 Process Network Synthesis

Process Network Synthesis (PNS) algorithms were elaborated in chemical engineering to estimate an optimal resource allocation for the production of desired products from given raw materials.

A PNS problem is represented by the so-called *P-graph* [6]. A P-graph is a directed bipartite graph where the two sets of disjoint nodes are materials (or material stores) and operating units. An operating unit consumes its input materials (connected by incoming edges) in order to produce its output materials (connected by outgoing edges). Then the PNS problem is to produce all products in the required amount from available raw materials by the defined operating units of minimal cost.

The solution of the problem is a *sub-P-graph* or a *solution structure*. In order to ensure the production of the products a feasible network has to satisfy five axioms [8]: (A1) every final product is represented in the graph, (A2) a material has no input operating units if and only if it represents a raw material, (A3) every operating unit represents an operating unit defined in the synthesis problem, (A4) every operating unit has at least one path leading to a final product, and (A5) if a material belongs to the graph, it must be an input to or output from at least one operating unit in the graph.

In PNS problems, costs are assigned to raw materials and operating units (as fix and operating costs). Naturally, an *optimal network* for the PNS problem is a solution structure together with operating rates (assigned to the operating units) such that the production of the products is of a minimal cost. Based on the specific structure of the PNS problem, the combinatorial optimization problem can be solved efficiently by exploiting of the features given by the five axioms. The optimal solution structure of a PNS problems is synthesized by three algorithms [7].

The *Maximal Structure Generation (MSG)* and the *Solution Structure Generation (SSG)* algorithms collect the *structurally* feasible sub-P-graphs generating a topology spanning the state space for further optimization. The MSG algorithm generates the superstructure of the feasible solutions. MSG achieves a significant reduction of the solution space in polynomial time by excluding those materials and operating units from the initial P-graph that violate any of the five axioms.

Then the SSG algorithm computes the structurally feasible networks: it builds up a set of possible operating units recursively starting from the products. The algorithm maintains a set of materials *'to be produced'*. The iteration consists of two main steps: at first, a material from the set 'to be produced' is selected and excluded. Then the operating units producing the selected material are taken into consideration: a subset of them is added to the previously selected operating units and the algorithm calls itself recursively.

The SSG algorithm generates exactly the set of solution structures satisfying the five axioms as the spanning graph of the set of the selected operating units. The resulting solution structures formulate a logic basis for the optimization problem that is closed under unification: an arbitrary solution structure can be generated by this elementary solutions under the operations of unifications. This corresponds

to the fact, that if two recipes produce some material according to the five axioms, then their simultaneous operation is a proper solution, as well. Please observe, that the SSG algorithm neglects the quantitative parameters: the solution structures are structural solutions (given by the contained operating units).

PNS problems over linear constraints and objective functions can be represented as a mixed integer linear programming. The *Accelerated Branch and Bound (ABB)* algorithm [26] solves this programming problem by exploiting the additional structural properties (the same way as in the SSG algorithm): the algorithm delivers a structurally feasible network with the operating rates of the individual operating units.

In the ABB algorithm, the *numerical cut* (bounding) is conventional (for more details on Branch–and–Bound algorithms see [19]): if there is a known solution of a lower cost, a branch of a greater value is cut, and only branches with lower value are taken into consideration. In contrary to a conventional B&B algorithm, *logical cut* (branching) is based on the structural properties of the problem: branching is performed by the binary variables indicating whether a particular operating unit is involved in the candidate solution according to the building process of SSG. This way the ABB algorithm reduces the search for combinations to the elementary solutions (recipes) instead of performing trials with individual elementary operations (i.e. the fixing of binary variables in a certain order used in the conventional B&B techniques). Thus, an essential improvement can be achieved by algorithm ABB by the logical cuts in contrast to the conventional algorithm that traverses all the $2^{|T|}$ problem nodes in worst case.

In the following section we discuss how PNS algorithms can be adapted to the Petri net optimal trajectory problem.

3 Optimal Trajectory Problem

System optimization and verification, validation problems can be formulated as combined reachability and optimization problems: (i) it has to be decided, whether a particular state of the system is reachable from the initial state using the available resources, and (ii) if the state is reachable, an optimal trajectory has to be computed. For instance, let us take a resource allocation problem modeled by a Petri net. Program states can be modeled by a set of Petri net places, and resource allocation to tasks can be modeled by transitions. Hence, the problem of finding an optimal trajectory between the initial and the desired states can be translated into a so-called Petri net partial reachability problem.

A state (marking) is reachable from an initial state if there exists a firing sequence between the two markings such that the trajectory is compliant in each individual step with the firing condition. Frequently, in engineering problems only the marking of a subset of places is relevant from practical point of view. Therefore, 'partial reachability' means that we should reach a state M that *covers* the desired (partial) state $M_{partial}$, i.e. $M \geq M_{partial}$.

An optimal trajectory problem can be mapped into the search for a trajectory

between two Petri net states with the least/most cost, time, quality indicator, etc. As transitions typically represent operations in the system we restrict ourself to cost functions linear in the number of executed transitions (however, this approach can be adapted to other quantitative or even qualitative parameters).

A *Petri net with cost parameters* is a $PN_c = \langle PN, \vec{c} \rangle$, where the cost function $\vec{c} \in (\mathbb{R}^+ \cup \{0\})^{|T|}$ assigns costs to the firing of the individual transitions. Thus, *the cost of the firing sequence* can be interpreted as the sum of the cost values of the transitions in the sequence, formally, $c_s = \vec{c} \cdot \sigma_s$, where σ_s is the transition occurrence vector of the firing sequence s, and c_s denotes the cost of the firing sequence.

This way the optimal trajectory problem can be formulated as follows: a *Petri net optimal trajectory problem* $R_{PN} = \langle PN_c, M \rangle$ is a Petri net with cost parameters and a target marking M (i.e. a partial marking s.t., $M(p) = 0$ for each place $p \in P$ that is irrelevant) where the problem is to find a *fireable* (executable) trajectory s such that there exists a marking $M' \geq M$ that is reachable from M_0 by firing sequence s such that c_s is minimal. Then σ_s is called an *optimal transition occurrence vector.*

Example. An example Petri net is shown in Fig. 1 with cost parameters depicted as transition labels. The optimal trajectory problem shown is to find a trajectory of minimal cost from the given initial marking $M_0 = (2, 3, 0, 0)$ to a marking in which place p_4 contains at least one token.

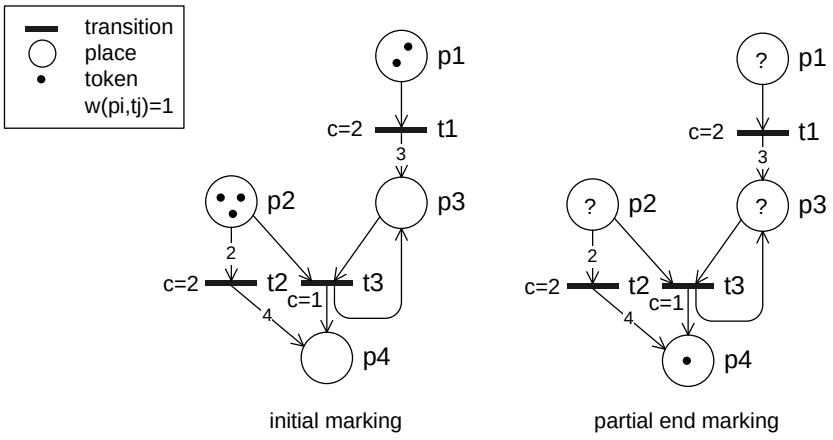

Figure 1: Example optimal trajectory problem

An obvious way to generate the optimal trajectory is the exhaustive traversal of the state space starting from the initial state. However, the exhaustive search for such a trajectory may easily result in state space explosion in case of complex systems. *In order to solve the Petri net reachability problem without state space explosion*, several techniques are discussed in the literature like unfolding-based

solutions for 1-bounded (safe) Petri nets [4], symbolic analysis using Binary Decision Diagrams (BDD) [21], or linear algebra–based algorithms [3, 24].

As an alternative solution, the state equation–based (Equation 1) method constitutes a mixed integer linear programming problem, as the components of the transition occurrence vector σ are nonnegative integers.

From the point of view of the reachability problem, the state equation method represents a *semi–decision* method as it formulates a necessary condition: if there is no σ that satisfies the inequality $M \leq M_0 + W \cdot \sigma$, then there is no firing sequence s from M_0 to a state that covers M. Otherwise, since the transition occurrence vector contains only reduced information on a trajectory by omitting the order of the individual firings, a solution vector σ of the state equation can represent a *spurious solution*. In this case, no appropriate firing sequence s exists such that $\sigma_s = \sigma$, despite the fulfillment of the state equation. Therefore the elimination of spurious solutions necessitates a separate check of the fireability of the solution transition vectors.

4 PNS Algorithms in the Solution of the Optimal Trajectory Problem

In order to support the modeling and solution of verification and optimization of real–time systems, we aimed at combining the PNS approach with the Petri net modeling in previous papers [12] and [11]. Although the two approaches are very similar, and PNS algorithms can be adapted to the optimal trajectory problem, the simple merging of ideas does not provide a proper solution in general.

4.1 The Optimal Trajectory Problem as a PNS Problem

As PNS algorithms focus only on the production of the desired end products without taking into account the intermediate byproducts, the PNS problem constitutes a partial reachability problem with cost parameters for the output places. Thus, a Petri net optimal trajectory problem *could* be described as a PNS problem, where (i) the places that are marked in the initial marking are the raw materials, (ii) the places that have to be marked at the target marking are the products, (iii) a transition is an operating unit mapping the pre- and post-set of the transition into the input and output materials of the operating unit, respectively (preserving the weight function), (iv) the cost of a transition is the operation cost of the corresponding operating unit, (v) and the result vector of the ABB algorithm can be interpreted as an optimal transition occurrence vector for the optimal trajectory problem.

The operating rates of operating units in chemical engineering may take continuous values, which is not allowed in case of a Petri net transition. Therefore the ABB algorithm cannot be directly used for the solution of the optimal trajectory problem but it can be reformulated so that the algorithm searches for integer

operating rates. The current paper proposes another method to deliver an integer solution based on the solution structures generated by the SSG algorithm (see Section 5).

On the other hand, thanks to their expressive power, Petri nets cover a wider range of models than P-graphs, i.e. not all Petri net reachability problems can be directly described as a PNS problem. One reason for the limited expressiveness of the PNS problems are the constraints introduced by the axioms (see Section 2.2) for the translated reachability problem. In [11] the problem of *'produced raw materials'* and *'catalysts'* is discussed.

In chemical engineering, *catalysts* play an essential role in production processes: they are both consumed and produced during the manufacturing process resulting in a total of zero change in their amount. However, while such catalysts cannot be assessed based upon material bill like equations as those used in the PNS model, PNS algorithms assume those materials to be available in the beginning.

In the Petri net optimal trajectory problem *catalysts* are tokens in places that participate in a cycle. In contrary to PNS problems, these catalysts may be required to the fireability of a sequence. Thus, it can occur, that there is no firing sequence in the solution structure of the PNS problem of the translated Petri net reachability problem. Thus, our problem has two different aspects requiring optimality, and fireability of the solution in order to be feasible.

4.2 Previous Results and New Contribution

In [11] we proposed an integrated algorithm. The algorithm consists of a collection of semi–decision methods that gradually eliminate the spurious non-fireable solutions as early as possible. The systematic search follows a best-first approach: the candidate solution nearest to the quantitative optimum is estimated as a transition occurrence vector and its feasibility (fireability) is checked until a solution is found. To provide a quick optimization technique for the reachability problem of general Petri nets, the optimization by the ABB algorithm (over integer variables) is complemented by a subsequent fireability check. This check is composed of a set of two subchecks: (i) semi–decisions like one use the *reachability–membership function* which is parameterized by two markings M_1, M_2 and evaluated to true if M_2 is reachable from M_1 or a fast *reachability check* of the marking reached by the solution transition occurrence vector from the initial marking (that is calculated by the state equation) eliminating a major part of spurious solutions, and (ii) an explicit *fireability check* of the transition occurrence vector providing the final evidence on the feasibility of solutions by using model checker SPIN [15].

Depending on the result of the checks, the algorithm either terminates delivering a fireable optimal solution or a new search is performed by the ABB algorithm for the second optimal solution. The main advantage of this approach is that both the reachability–membership function and the generated SPIN code can be reused to check the next candidate solution.

The new contribution in the current paper compared with previous works [11], [12] is (i) the generation of an integer candidate transition occurrence vector using a

Branch–and–Bound algorithm where branching is based on the *logical combinations of the basis solution structures* as variables (Section 5), and (ii) the introduction of an efficient reachability–membership function computation in the form of the *transitive closure of the single-step transition relation* (Section 6).

5 Solution Structure-based Optimization

In [11] we used the Accelerated Branch and Bound algorithm to generate the transition occurrence vector of an optimal candidate trajectory. However, the current ABB implementation potentially violates the required integrality of the Petri net reachability problem, i.e. it may return a non–integer as an optimal solution.

While the acceleration in the original ABB algorithm lies in the exploitation of the *correlation between the binary variables* assigned to the transitions (or to the operating units) our new Branch–and–Bound solution generates an optimal candidate trajectory for the Petri net optimal trajectory problem *based on the logic basis of the Petri net structure*. While the bounding step is identical in both approaches, instead of making decisions on the binary variables and their correlation at branching, our method considers *complete solution structures as a single boolean variable*.

The set of the logically feasible solution structures (satisfying Axioms (A1)-(A5) in Section 2.2) is closed under the logical unification. A basis of this set can be generated such that all solution structures can be composed as the logical union of some of these basis solution structures. This way our method covers all the feasible solutions.

Let be given an optimal trajectory problem $R_{PN} = (P, T, w, M_0, \vec{c}, M_p)$. At first, we generate a basis of the solution structures by a slightly modified version of the SSG algorithm. The modification lies in the collection step of a *basis* of the solution structures: a subset of the solution structures is called *solution structure basis* such that all other solution structures can be calculated as the disjunction (or union) of some of the basis solution structures. Since the set of the solution structures is closed under unification, this subset does exist and it can be computed. Thus, all new solution structure generated by the SSG algorithm has to be evaluated, whether it can be synthesized as the union of some previously generated solution structures or not.

The solution structure basis for the Petri net reachability problem constitute a set of characteristic vectors $S(PN)_{basis} = \{\theta_i\}$ representing the solution structures such that for all $j : 1 \leq j \leq |T|$:

$$\theta_i(j) = \begin{cases} 1 & \text{if } t_j \text{ is contained in the corresponding solution structure} \\ 0 & \text{otherwise.} \end{cases}$$

The candidate optimal transition occurrence vector σ_{PN_R} for the Petri net is an integer solution of the following mixed integer linear programming problem such

that σ_{PN_R} is generated as the *logical combination* (union) of the elements of the logic basis (i.e. the characteristic vectors of the solution structure basis).

$$\text{minimize } c \cdot \sigma_{PN_R}, \tag{2}$$

$$\text{subject to } M \leq M_0 + W \cdot \sigma_{PN_R}, \tag{3}$$

and the characteristic vector

$$\chi(\sigma_{PN_R}) \text{ of } \sigma_{PN_R} = \bigvee_{i=1}^{|S(PN)_{basis}|} \alpha(i) \cdot \theta_i \tag{4}$$

$$\sigma_{PN_R} \in \mathbb{N}^{|T|}, \alpha \in \mathbb{B}^{|S(PN)_{basis}|}, \mathbb{B} = \{0, 1\} \tag{5}$$

The above problem can be solved by a Branch–and–Bound (B&B) algorithm. The starting problem consists of the objective function Eq. 2 and the constraint Eq. 3. Then the B&B tree is built by adding constraints due to Eq.4.

Eq.4 ensures that the logic structure of the candidate optimal transition occurrence vector is composed of some basis solution structures: the characteristic vector of the candidate solution corresponds to the logical disjunction of some basis solution structures. Numerically, there is a corresponding $\alpha \in \mathbb{B}^{|S(PN)_{basis}|}$ vector for all subproblem such that $\forall 1 \leq i \leq |T| : 0 = \sigma_{PN_R}(i) \iff \nexists 1 \leq k \leq |S(PN)_{basis}| : \alpha_k = 1 \wedge \theta_k(i) = 1$. In other words, exactly those transitions are included in the solution transition occurrence vector that are contained by some of the composing basis solution structures.

Branching. In the B&B algorithm branching is performed by the binary variables α_i considering the corresponding solution structure θ_i. The main difference between the ABB algorithm (Section 2.2) and our B&B algorithm is that the ABB algorithm decides on the binary variables according to the individual transitions while we search the solution over the logically feasible solution structures (i.e. over the set of the binary variables!).

Generation of the Integer Solution. since we need integer solutions, if we reach a leaf, i.e. the logical structure of the candidate solution is fixed, we solve the given restricted MILP problem.

Bounding and Pruning. Bounding is traditional: if the LP relaxation is infeasible or returns a worse cost value than a best known value, the tree is pruned.

An additional advantage of our algorithm is that the next best solution can be easily computed using the already built B&B tree.

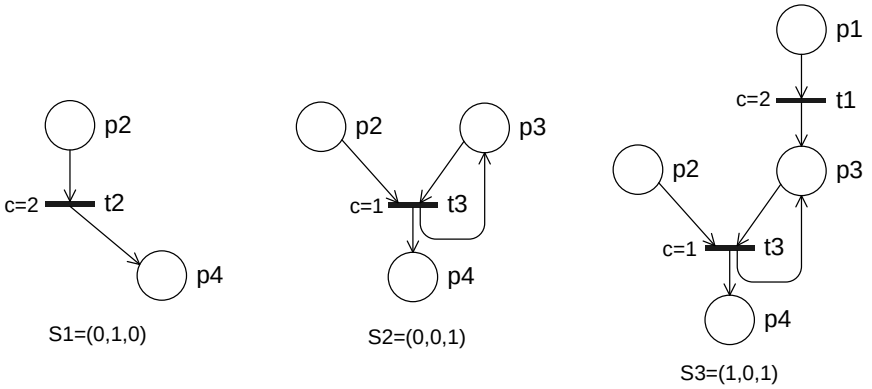

Figure 2: Solution structure basis for the reachability problem of the running example generated by the SSG algorithm

Example. Fig. 2 shows the solution structures in the solution structure basis for the reachability problem generated by the modified SSG algorithm. Please observe, that they are mutually independent under unification and there is only a single additional solution structure that can be composed as the logical union of some basis solution structures: the union of the three solution structures.

According to the example in Fig. 1, the candidate optimal transition occurrence vector σ_{PN_R} has to satisfy the following inequality system.

$$\text{minimize } (2,2,1) \cdot \sigma_{PN_R} \tag{6}$$

$$\text{subject to } (0,0,0,1) \leq (2,3,0,0) + \begin{pmatrix} -1 & 0 & 0 \\ 0 & -2 & -1 \\ 3 & 0 & 0 \\ 0 & 4 & 1 \end{pmatrix} \cdot \sigma_{PN_R}, \tag{7}$$

$$\chi(\sigma_{PN_R}) = \bigvee_{i=1}^{3} \alpha(i) \cdot \theta_i \tag{8}$$

$$\text{where } \sigma_{PN_R} \in \mathbb{N}^{|T|}, \alpha_i \in \mathbb{B} : \forall 1 \leq i \leq 3, \tag{9}$$

$$\text{and } \theta_1 = (0,1,0),\ \theta_2 = (0,0,1),\ \theta_3 = (1,0,1). \tag{10}$$

The Branch-and-Bound algorithm solved 7 LP relaxation, and 1 MILP problem, and the solution is $\sigma_{PN_R} = (0,0,1)$. The corresponding B&B tree is shown in Fig. 3.

Initially, the α_i variables are restricted neither to 1 nor to 0. The cost value and the solution σ_{PN_R} vector of the LP relaxation is shown below the constraints for α_i. Since the solution is non–integer valued, α_1 is restricted to 1 (see $Node_1$). Then we branch the tree taking α_2 into consideration. Since the unbounded α variables are between 0 and 1, m denotes a very small value in subproblems $Node_2$, $Node_3$, and

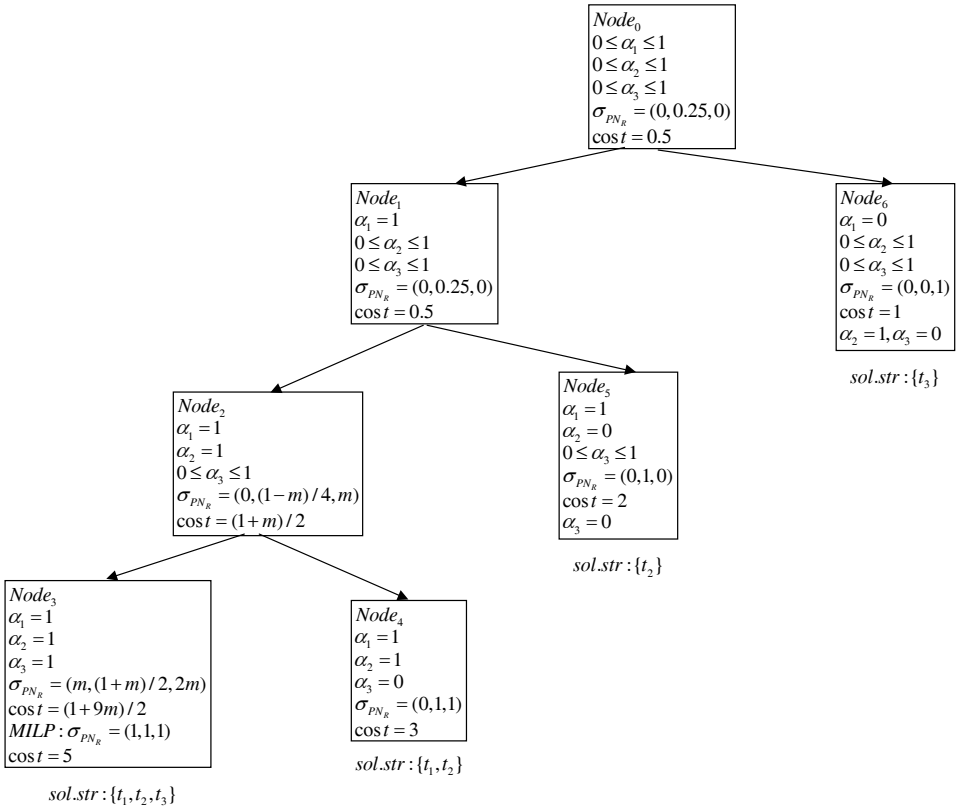

Figure 3: The B&B tree solving the optimal trajectory problem in Fig. 1

$Node_4$. In the leaf $Node_3$ the solution of the MILP problem returns the first best known solution $\sigma_{PN_R} = (1, 1, 1)$ with a cost of 5 units and the logical structure of the solution is composed of all the three solution structures.

At nodes $Node_4$, $Node_5$, ¡and $Node_6$ the solution of the corresponding LP relaxation returns integer solutions of a worse value thus, the tree can be pruned at these nodes. The candidate optimal solution is found at $Node_6$ composed of one basis solution structure θ_2. (Please note, that the depth-first algorithm required no bounding. However, if branching is done for instance in the order of $\alpha_2, \alpha_1, \alpha_3$, the optimal solution is found already at the first node.)

However, the solution ($S2$ in Fig. 2) represents a non-fireable solution from the initial marking $M_0 = (2, 3, 0, 0)$, since transition t_3 is not fireable at marking $M_0 = (2, 3, 0, 0)$ and no other transition is included in the solution. In order to eliminate these spurious solutions we introduce the following reachability and fireability checks.

6 Reachability–Membership Function Generation

We use symbolic modeling techniques introduced in [21] for the check of the candidate transition occurrence vector computed by the previous PNS algorithms-based programming problem. The efficiency of our new method lies in its parametral feature.

6.1 The Reachability–Membership Function Generation Algorithm

The states and the representation of the dynamic behavior (successor relation) of Petri nets are encoded as Boolean functions during the generation of the reachability–membership function (shortly reachability function). The algorithm generates the reachability function as the transitive closure of the single-step transition relation starting from the current set of reachable markings encoded as the disjunction of Boolean functions (collecting transitions that are enabled in the set of reachable markings). The computation of the set of new reachable states is performed by Boolean operations based on their *Binary Decision Diagram (BDD)* representation.

Due to the encoding of the reachable set by Boolean variables, the reachability function can be parameterized by the initial and the final states. The reachability function returns the value true, if the final state is reachable from the initial state and false, otherwise. The reachability function is characteristic for the Petri net, thus, it has to be estimated one by one, and *does not have to be newly generated* for the check of the next candidate solution after a spurious solution is found. The other advantage of the method compared to the previous one in [11] is that the states do not have to be explicitly enumerated as the parameters of the reachability function. Nevertheless, the main disadvantage of the method is its slightly limited scope (the Petri net has to be bounded) in order to be able to be parameterized and encoded by the function.

For the sake of simplicity, let $PN = \langle P, T, w, M_0 \rangle$ be a safe (or 1-bounded) Petri net, i.e. for every marking M reachable from the initial marking M_0, $\forall p \in P : M(p) \leq 1$ holds. Thus, the weight is 1 for each arc. (Note, that the algorithm also works for general bounded Petri nets using n Boolean variables for upper bound k s.t. $2^n \geq k$ [21], or using *Functional Decision Diagrams* ([25]).

A marking M is encoded as a set of Boolean variables $\{p_1, \ldots, p_n; n = |P|\}$ (as a characteristic function). Each variable p_i assigned to a place has a value of 1 if and only if the corresponding place is marked in M.

A set of markings can be represented by the disjunction (union) of their Boolean representations. The enabledness of a transition t in marking M is evaluated by the equation $E_t = \prod_{p_i \in \bullet t} p_i$. That means that the transition is enabled if there are at least as many tokens in its places $p \in \bullet t$ as required by the weight function that is 1 in our case. Since we supposed that the Petri net is a safe Petri net, this number of tokens is exactly 1, i.e. $p_i \equiv 1$.

The following transition function is used in order to compute the subsequent reachable states from the markings in the actual set. $\delta^t : P \rightarrow \{true, false\}$ is the *transition function* for $t \in T$, i.e. it denotes the effect of transition t on the Boolean variables after its firing.

$$\delta_i^t(p_1, \ldots, p_n) = \begin{cases} 1 & \text{if } p_i \in t\bullet \\ 0 & \text{if } p_i \in \bullet t \setminus t\bullet \\ p_i & \text{otherwise.} \end{cases}$$

The parameterized reachability function is generated iteratively as the *transitive closure of the single-step transition function*. Algorithm 1 sketches this stepwise generation of the reachability function. (Note, that the conjunction and disjunction operations on markings are interpreted as the corresponding operation on the place variables.)

The single-step transition function g is the disjunction of the conjunction of the transition function δ^t and the firing condition E_t for all transitions $t \in T$ (see line 1). The algorithm starts from the initial set of reachable markings, i.e. $f_0(v_1, \ldots, v_n, w_1, \ldots, w_n) = (v_1, \ldots, v_n \equiv w_1, \ldots, w_n)$ (see line 1). The actual reachability function $f_i(M_0, M)$ after the i-th iteration is evaluated to true if marking M is reachable from M_0 by a firing sequence with maximal length of i. Then the next reachability function f_{i+1} is calculated as the *disjunction of* the actual reachability function f_i and the single-step transition function g from the set of markings that are reachable in at most i steps from the initial marking (i.e. $f_i(M_0, M_1) \wedge g(M_1, M)$, see line 1. Both the single-step transition function and the reachability function are parameterized by variables $(v_1, \ldots, v_n)$, $(r_1, \ldots, r_n)$, $(w_1, \ldots, w_n)$ denoting the initial, the auxiliary and the end markings, respectively. The algorithm terminates if the computation reached a fixpoint, i.e. the transitive closure is computed (at line 1).

Since the generation of the reachability function does not store the order of the transition firings, the described method is appropriate only to check the reachability of the marking computed by the state equation. On the other hand, if we were interested only in the reachability of this end marking and we would not like to reuse the reachability function, we could improve our method in the following way.

- The sum of the components of the candidate transition occurrence vector provides the necessary number of iterations. Thus, after exceeding this number of iterations, the computation of the reachability function can be stopped.
- The transition function could be restricted only to those transitions that are involved in the candidate transition occurrence vector.

Example. In the running example (see in Fig. 1), transition t_1 is enabled if there is a token in p_1. Thus, the enabledness of transition t_1 can be expressed as follows: $E_{t_1} = p_1 = (p_1 \equiv 1)$. The firing of transition t_1 removes the token from p_1 and produces one token to place p_3, while the marking of places p_2 and p_4 do not change. Namely, the transition function for t_1 is $\delta^{t_1}(p_1, p_2, p_3, p_4) = (0, p_2, 1, p_4)$.

Algorithm 1 Reachability–membership function generation

1: Let be a safe Petri net $PN = \langle P, T, w, M_0 \rangle$ given.
2: Compute the ROBDD of the transition function g:
3: $g(v_1, \ldots, v_n, w_1, \ldots, w_n) = \bigvee_{\forall t \in T} \left[\bigwedge_{i=1}^{n} (w_i \equiv \delta_i^t(v_i)) \cdot E_t \right]$
4: $\{g(v_1, \ldots, v_n, w_1, \ldots, w_n) = 1 \iff \exists t \in T : \forall i : 1 \leq i \leq n : \delta_i^t(v_i) \equiv w_i$ and $E_t \wedge v_1 \ldots v_n = 1.\}$
5: Compute the ROBDD of the 0-th iteration of the reachability function:
6: $f_0(v_1, \ldots, v_n, w_1, \ldots, w_n) = (v_1 \ldots v_n \equiv w_1 \ldots w_n) = \bigwedge_{i=1}^{n} (v_i \equiv w_i)$
7: $i \leftarrow 0$
8: **repeat**
9: Compute the ROBDD of f_{i+1} using the ROBDDs of f_i and g:
10: $f_{i+1}(v_1, \ldots, v_n, w_1, \ldots, w_n) = f_i(v_1, \ldots, v_n, w_1, \ldots, w_n) \vee f_i(v_1, \ldots, v_n, r_1, \ldots, r_n) \wedge g(r_1, \ldots, r_n, w_1, \ldots, w_n)$
11: {the reachability function $f_i(M_0, M)$ is evaluated to true if and only if M is reachable from M_0 by a firing sequence with maximal length of i.}
12: $i \leftarrow i + 1$
13: **until** $f_i \equiv f_{i+1}$.

According to the above algorithm, in our running example functions f_0, g, and the reachability function f are the following. Lines 12-14 in the definition of function g refer to the effect and the enabledness of transition t_1, t_2, and t_3, respectively. The reachability function f reflects the firing sequences on a parameterized marking: the 'sub-equation' in line 16 is evaluated to true only for end markings $M = (w_1, w_2, w_3, w_4)$ that are the same as the initial marking $M_0 = (v_1, v_2, v_3, v_4)$ while the Boolean expressions in lines 17-19 and 20-21 are evaluated for markings reachable by a firing sequence of length 1, and 2, respectively.

$$f_0(v_1v_2v_3v_4, w_1w_2w_3w_4) = (w_1w_2w_3w_4 \equiv v_1v_2v_3v_4) \tag{11}$$
$$g(v_1v_2v_3v_4, w_1w_2w_3w_4) = (w_1w_2w_3w_4 \equiv 0v_21v_4) \wedge (v_1 \equiv 1) \vee \tag{12}$$
$$(w_1w_2w_3w_4 \equiv v_10v_31) \wedge (v_2 \equiv 1) \vee \tag{13}$$
$$(w_1w_2w_3w_4 \equiv v_1011) \wedge (v_2 \equiv 1) \wedge (v_3 \equiv 1) \tag{14}$$

$$f(v_1v_2v_3v_4, w_1w_2w_3w_4) = \tag{15}$$
$$(w_1w_2w_3w_4 \equiv v_1v_2v_3v_4) \vee \tag{16}$$
$$(w_1w_2w_3w_4 \equiv 0v_21v_4) \wedge (v_1 \equiv 1) \vee \tag{17}$$
$$(w_1w_2w_3w_4 \equiv v_10v_31) \wedge (v_2 \equiv 1) \vee \tag{18}$$
$$(w_1w_2w_3w_4 \equiv v_1011) \wedge (v_2 \equiv 1) \wedge (v_3 \equiv 1) \vee \tag{19}$$
$$(w_1w_2w_3w_4 \equiv 0011) \wedge (v_1 \equiv 1) \wedge (v_2 \equiv 1) \vee \tag{20}$$
$$(w_1w_2w_3w_4 \equiv 0011) \wedge (v_1 \equiv 1) \wedge (v_2 \equiv 1) \wedge (v_3 \equiv 1) \tag{21}$$

6.2 Implementation of the Reachability Function: Binary Decision Diagrams

In order to gain efficiency, the reachability function computation requires an efficient representation of Boolean functions. As Binary Decision Diagrams (BDDs) [2] provide an efficient form to manipulate Boolean functions, we express the above Boolean functions by means of them. BDDs are directed acyclic graphs with two leaf nodes that represent Boolean functions 0 and 1. The other nodes represent the Boolean variables and each of them has two outgoing edges labeled by 1 'then' or 0 'else'. Thus, the evaluation of a Boolean function is performed by the traversal of the corresponding BDD according to the actual values of the variables.

To generate the reachability function (see Algorithm 1) as a BDD, we use Reduced Ordered BDDs (ROBDD) where the equivalent branches of the tree are merged and the redundant variables are excluded from the tree. Then the disjunction and conjunction operations in the calculation of g, f_0, and f_{i+1} (in lines 1, 1, 1) can be directly executed on the ROBDD representations. However, the ROBDDs to be merged have to contain the same variables in the same order. Nevertheless, arbitrary variables can be added to the ROBDD of functions f_i and g thus, the computation of f_{i+1} is performed by using the ROBDDs of f_i and g with variables $v_1, \ldots, v_n, r_1, \ldots, r_n, w_1, \ldots, w_n$. Since variables $r_1, \ldots, r_n$ in g can be substituted by variables $v_1, \ldots, v_n$, the resulting ROBDD of f_{i+1} contains only variables $v_1, \ldots, v_n, w_1, \ldots, w_n$ satisfying the definition of f such that it is interpreted on the variables of the initial and the end markings.

7 Related work

The use of integer programming methods in the analysis of Petri nets is not a novel idea. In [18] *deadlock detection* was reduced to a mixed integer linear programming problem. In [17] the authors presented a further development of this approach to prove *deadlock detection, mutual exclusion, and marking reachability and coverability.* In contrary to our approach, this solution was based on the *unfolding* of the Petri net. Another unfolding-based solution is discussed for *safe* Petri nets in [4].

Linear algebraic algorithms were used to solve the Petri net reachability problem without state space explosion in [3, 24]. Although these techniques are powerful, in general they provide only semi–decision techniques to decide the reachability of a given marking while we are dealing with general Petri nets.

Several papers [14, 9] use *stochastic Petri nets* or *timed Petri nets* to model and solve scheduling and optimization problems (e.g. in the field of manufacturing systems). These approaches use mainly simulation and performance evaluation in order to solve the problem (for instance, in [28] profit function values are represented as a function according to some given restrictions using simulation of stochastic Petri nets). Since we do not only want to solve the optimal trajectory problem (where we have fixed parameters assigned to the transitions), but we also aim at *simultaneous verification and optimization*, these techniques are not appropriate

for our purposes.

Model checking methods were used to solve scheduling and optimization problems in several papers. These papers are common in the feature that the problem is translated into a reachability condition that can be easily encoded into a temporal logic expression to be verified by the model checking tool. In [5] timed automata was used to model a steel plant and the corresponding scheduling problem. The scheduling problem was solved using the *UPPAAL* tool, which is a model checker for networks of timed automata. [23] and [13] deal with time optimization problems using the SPIN model checker. The main idea of this solution (originated from [23]) is to interpret Branch-and-Bound techniques encoding both the bounding and branching conditions into the linear temporal expression to be verified. Although the optimal trajectory problem could be solved using only SPIN based on the above technique the pre–optimization in our method using linear programming solutions significantly reduces the search space for the optimal solution.

In [16] the authors analyzed the *executability* of a given process network solution,where operating units consume exactly one unit of their input materials and produce exactly one unit of their output materials. This modified PNS problem was solved using an *automaton theoretical approach* such that the problem is transformed into a problem to find the shortest path in the weighted transition graph of the automaton constructed from the PNS problem. In case of Petri nets where the weights of the arcs are restricted to one, the fireability of the candidate solution transition occurrence vector can be proved using this method.

8 Conclusion and Initial Results

Finally, our method was tested on an example Petri net with 11 places and 7 transitions. The example originates in [1]: we reformulated the PNS problem as a Petri net optimal trajectory problem. The example Petri net is shown in Figure 4, where the weight of the arcs are written on the edges.

The original SSG algorithm generates 19 solution structures (see the solution structures in [1]) while our modified algorithm computes only 7 basis solution structures. The Branch–and–Bound algorithm (described in Section 5) executes 13 LP relaxations and 5 MILP problem (at the leaves of the tree). This number of LP relaxations and MILP problem solving is still less than the number of the MILP problems to be solved if we would like to find the optimal solution calculating the optimal solution for all the 19 solution structures. The SSG basis-based algorithm returns the solution $\sigma_{PN_R} = (0, 1, 0, 0, 1, 0, 1)$ with cost 1085, i.e. the candidate optimal transition occurrence vector contains transitions $t2$, $t5$, and $t7$ (each exactly once).

As a next step, we calculate the end marking by the state equation. Then we check whether there exists a trajectory from the given initial marking to the calculated end marking using the reachability function. Substituting the initial and the end marking into the reachability function, the answer is *positive*, i.e. the end marking is reachable from the initial marking. In case of a negative answer, the

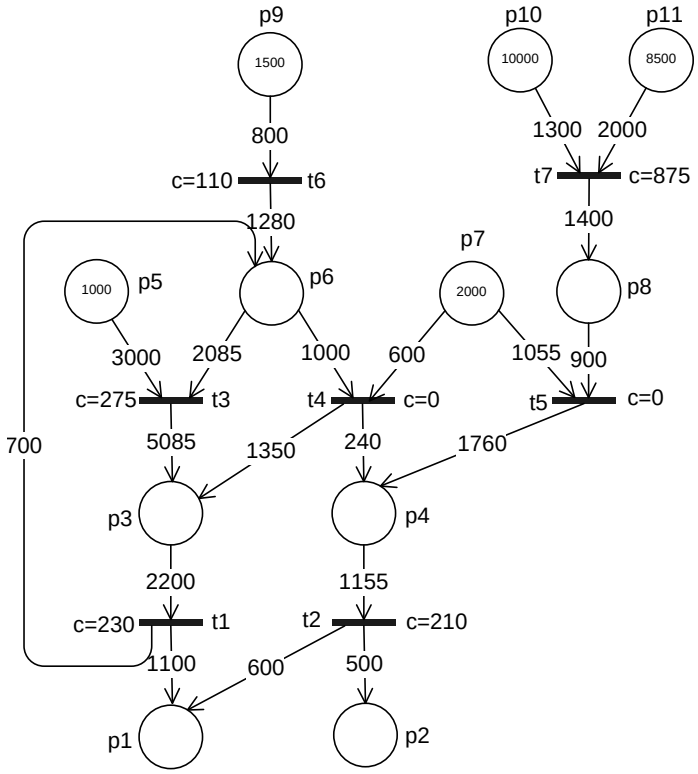

Figure 4: Example optimal trajectory problem

next optimal solution should be generated by our Branch–and–Bound algorithm for which the previous calculations could be reused, as well as the reachability function for the next check.

Finally, we generate the fireable trajectory according to the candidate optimal transition occurrence vector, that is the following transition sequence: $\langle t7, t5, t2 \rangle$.

Since the solution of the optimal trajectory problem needs the generation of the fireable trajectory, the efficiency of our method cannot be easily compared to other approaches. In addition, several structural specialties could be explored, for which dedicated algorithms give much more efficient results. In Table 1 the number of LPs and MILPs solved are shown both in case of ABB (however it does not return an integer solution!!!) and in case of our method, while the execution time values (on a Pentium IV) for the individual phases of the algorithm are given in Table 2.

Conclusion In the current paper we introduced a solution structure-based method for optimization purposes to solve the Petri net optimal trajectory problem and the generation of a reusable, initial marking–independent reachability function

Table 1: Initial results: comparison of number of LPs and MILPs solved by the ABB algorithm and the solution structure basis-based B&B algorithm

algorithm	number of LPs	number of MILPs
ABB		0
B&B algorithm based on the solution structure basis	13	5

Table 2: Initial results: execution time for the example optimal trajectory problem

step	execution time (sec)
Generation of the solution structure basis $S(PN)_{basis}$	0.01
Generation of the optimal candidate transition occurrence vector σ_{PN_R}	0.72
Generation of the reachability function	4.56
Trajectory generation using SPIN	0.01

to check the reachability of the target state corresponding to the candidate optimal solution structure. As further work, we aim to develop a similar framework for time optimization with Petri nets using dedicated linear programming problem solutions developed for the so-called S-graphs [22].

References

[1] M. H. Brendel, F. Friedler, and L. T. Fan. Decision–Mapping: A Tool for Consistent and Complete Decisions in Process Synthesis. *Computers Chemical Engineering*, 24:1859–1864, 2000.

[2] R. Bryant. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[3] J. Desel. *Petrinetze, Lineare Algebra und lineare Programmierung*, volume 26 of *Teubner-Texte zur Informatik*. B. G. Teubner Stuttgart-Leipzig, 1998.

[4] J. Esparza and C. Schröter. Unfolding Based Algorithms for the Reachability Problem. *Fundamenta Informaticae*, 46:1–17, 2001.

[5] A. Fehnker. Scheduling a Steel Plant with Timed Automata. In *Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA'99)*. IEEE Computer Society Press, 1999.

[6] F. Friedler, L. T. Fan, and B. Imreh. Process Network Synthesis: Problem Definition. *Networks*, 28(2):119–124, 1998.

[7] F. Friedler, K. Tarjan, Y. W. Huang, and L. Fan. Combinatorial Algorithms for Process Synthesis. *Computers Chemical Engineering*, 16:313–320, 1992.

[8] F. Friedler, K. Tarjan, Y. W. Huang, and L. Fan. Graph-Theoretic Approach to Process Synthesis: Axioms and Theorems. *Chemical Engineering Science*, 47(8):1973–1988, 1992.

[9] T. Gu, P. A. Bahri, and G. Cai. Timed Petri-net based formulation and an algorithm for the optimal scheduling of batch plants. *International Journal of Applied Mathematics and Computer Science*, 13(4):527–536, 2003.

[10] S. Gyapay. Model Transformation from General Resource Models to Petri Nets using Graph Transformation. Technical report, Technical University of Berlin, Dept. of Computer Science, July, 2004. ISSN 1436-9915.

[11] S. Gyapay and A. Pataricza. A combination of Petri nets and Process Network Synthesis. In *2003 IEEE International Conference on Systems, Man & Cybernetics, Invited Sessions/Track on "Petri Nets and Discrete Event Systems"*, pages 1167–1174, Washington, D.C., USA, October 5-8 2003.

[12] S. Gyapay, A. Pataricza, J. Sziray, and F. Friedler. Petri Net-based optimization of production systems. In A. Lovrenčić and I. J. Rudas, editors, 6^{th} *IEEE International Conference on Intelligent Engineering Systems*, pages 465–469. Organized and published by Faculty of Organization end Informatics, University of Zagreb, Croatia, May 26–28 2002.

[13] S. Gyapay, A. Schmidt, and D. Varró. Joint Optimization and Reachability Analysis in Graph Transformation Systems with Time. *Electronic Notes in Theoretical Computer Science*, 109:137–147, 2004.

[14] I. Hatono, K. Yamagata, and H. Tamura. Modeling and Online Scheduling of Flexible Manufacturing Systems Using Stochastic Petri Nets. *IEEE Trans. Softw. Eng.*, 17(2):126–132, 1991.

[15] G. J. Holzmann. *The SPIN Model Checker - Primer and Reference Manual*. Addison-Wesley, Boston, USA, 2003.

[16] B. Imreh. Automaton theory approach for solving modified PNS Problems. *Acta Cybernetica*, 15(3):327–338, 2002.

[17] V. Khomenko and M. Koutny. Verification of Bounded Petri Nets Using Integer Programming. Technical report, Department of Computing Science, University of Newcastle upon Tyne, 2000. Technical Report CS-TR-711.

[18] S. Melzer and S. Römer. Deadlock Checking Using Net Unfoldings. In *CAV '97: Proceedings of the 9th International Conference on Computer Aided Verification*, pages 352–363. Springer-Verlag, 1997.

[19] M. Minoux. *Mathematical Programming: Theory and Algorithms*. John Wiley & Sons, 1986.

[20] T. Murata. Petri nets: Properties, analysis and applications. In *Proc. IEEE*, volume 77, pages 541–580, 1989.

[21] E. Pastor, J. Cortadella, and O. Roig. Symbolic Analysis of Bounded Petri Nets. *IEEE Transactions on Computers*, 50(5):432–448, 2001.

[22] J. Romero, L. Puigjaner, T. Holczinger, and F. Friedler. Scheduling Intermediate Storage Multipurpose Batch Plants Using the S-Graph. *AIChE Journal*, 50(2):403–417, 2004.

[23] T. C. Ruys. Optimal scheduling using Branch-and-Bound with SPIN 4.0. In *Proc. 10th International SPIN Workshop*, volume 2648 of *LNCS*, pages 1–17, Portland, Oregon, USA, May 9–10 2003. Springer.

[24] M. Silva, E. Teurel, and J. M. Colom. Linear algebraic and linear programming techniques for the analysis of Place/Transition net systems. In G. R. W. Reisig, editor, *Lectures on Petri Nets I:Basic Models*, volume 1791 of *LNCS*, pages 309–373. Springer, 1998.

[25] R. S. Stankovic. Functional Decision Diagrams for Multiple-Valued Functions. In *25th IEEE International Symposium on Multiple-Valued Logic*, pages 284–189, 1995.

[26] J. B. Vajda, F. Friedler, and L. Fan. Parallelization of the Accelerated Branch and Bound Algorithm of Process Synthesis: Application in Total Flowsheet Synthesis. *Acta Chimica Slovenica*, 42(1):15–20, 1995.

[27] D. Varró and A. Pataricza. VPM: Mathematics of metamodeling is metamodeling mathematics. *Journal of Software and Systems Modelling*, 3(2):187–210, 2003.

[28] A. Zimmermann, D. Rodríguez, and M. Silva. Modelling and Optimization of Manufacturing Systems: Petri Nets and Simulated Annealing. In *Proceedings of the 1999 European Control Conference ECC99*, Karlsruhe, Germany, August 1999.