A Combination of Petri Nets and Process Network Synthesis*

Szilvia Gyapay and András Pataricza
Department of Measurement and Information Systems
Budapest University of Technology, Budapest, Hungary
{gyapay,pataric}@mit.bme.hu

Abstract – *Resource allocation and scheduling optimization problems are core problems in the field of IT systems. However, such problems frequently underlie several additional constraints. The formalization of a real life problem requires a well-defined mathematical and modeling approach providing* an integrated verification and optimization. *The current paper proposes such methods adapting Process Network Synthesis algorithms to Petri net reachability problem: combining the efficiency of PNS optimization algorithms with the modeling power of Petri nets. They provide powerful techniques to compute optimal trajectories for the reachability analysis of the modeled system.*[1]

Keywords: Petri nets, Process Network Synthesis, verification, optimization.

1 Introduction

The design of IT systems requires sophisticated tools in order to simultaneously assure the productivity and the quality of the final system. Since Petri nets are an appropriate means to model safety critical and high available systems they are widely used in the design of IT systems. Petri nets have a rich mathematical background supporting verification and validation. However, the appearance of another focal task, namely, optimization requires the integration of optimization techniques as an extension to the modeling paradigm offered by Petri nets.

The expressive power of Petri nets allows us to define an *optimal trajectory problem*: cost parameters can be added to transition firings and a minimal cost solution is searched for. Temporal logic conditions can be formulated in order to confine the solution space by additional constraints. Thus, the problem is to find an optimal trajectory from a given initial state to an end state with minimal cost where the trajectory satisfies the specified temporal conditions.

The obvious solution for the optimal trajectory problem is the brute force traversal of the state space of the Petri net selecting the subsequent transition to be fired according to the objective function. This method frequently requires the exhaustive traversal of the state space leading to a combinational explosion of the computational complexity.

Avoiding the state space explosion, semi-decision methods (e.g., the state equation method) are frequently used to avoid the state space explosion by restricting the set of feasible solutions. However, semi-decision techniques are able only to prove that a system does *not* violate of its specification by showing that no solution exists upon contradicting assumption.

Problem statement and own contribution Although model checking and optimal scheduling have recently been combined for model checking tools with explicit state space traversal [8], the literature of Petri nets lacks such a combined technique. To bridge this gap, the current paper proposes an integrated optimization and validation algorithm for the optimal trajectory problem.

It is a basic problem that in complex models like the faithful models of IT systems the number of candidate operations is extremely large. This way the solution space may consist several billions of states making the use of traditional algorithms infeasible. Additionally, both the controllability and observability of an IT system are limited. For instance, during the observation of a program run in their natural environment we can observe only its interactions with the outside world but no detailed information is available on the internal state of the program. Accordingly, such special methods are required for a variety of IT related applications which are able to cope with problems: (1) characterized by a limited observability and controllability (2) where the relevant information belongs to the controllable input and observable output of the system without a special of interest on its internal state.

That kind of restrictions introduce limitations on the problem class to be solved. In the forthcoming presentation we will show how the structural limitation can be exploited in order to provide solution methodologies able to handle large scale systems as well.

Our approach exploits the efficiency of *Process Network Synthesis* (PNS) algorithms that were developed to generate the optimal manufacturing of final products from raw materials by applying operating units. In PNS problems, the

* **0-7803-7952-7/03/$17.00 © 2003 IEEE.**
[1] **This work was partially supported by project OTKA T038027.**

so-called *Accelerated Branch and Bound* algorithm (ABB) generates the optimal (either minimal or maximal) solution for the resource allocation problem.

Based on the graphical and semantical resemblance between Petri net reachability problem and PNS problems, an obvious idea is to adapt the ABB algorithm in order to solve the optimal trajectory problem. However, the ABB algorithm does not provide a fireable optimal trajectory due to its different background (see Section 4.2).

Therefore, a gradual filtering method consisting of semi-decision techniques is introduced in order to eliminate the spurious solutions as soon as possible. The method contains four consecutive steps.

At first, the adapted ABB algorithm generates a candidate optimal solution for the optimal trajectory problem. If it exists, the reached end state is estimated by the state equation and its reachability from the initial state is decided subsequently. Finally, if the candidate solution was not rejected by the previous check, model checking tool *SPIN* is used to prove the fireability of the candidate solution and specified temporal conditions. If the check is negative the next best solution is delivered by the ABB algorithm and the introduced check is performed again.

The rest of the paper is structured as follows. Section 2 and Section 3 give a short introduction into Petri nets and Process Network Synthesis, respectively. Then the resemblance of the two approaches is discussed in Section 4 that is followed by the analysis of the direct algorithm adaptation. The integrated technique is given in Section 5. Finally, we conclude our work by summarizing ongoing research activities.

2 Petri Nets

2.1 Basic definitions

Petri nets are directed, bipartite graphs represented by a four-tuple $PN = \langle P, T, w, M_0 \rangle$, where P and T are the sets of place and transition nodes, respectively. Places may contain tokens, whose distribution describes the state of the net, represented by a $|P|$-dimensional vector over naturals called marking, where the i-th component ($M(p_i)$) denotes the number of tokens contained in p_i. M_0 denotes the initial marking of the net.

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 edges 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, such that $\bullet x = \{y \in P \cup T : w(y, x) > 0\}$ and $x\bullet = \{y \in P \cup T : (x, y) > 0\}$.

At marking M a t transition is enabled (i.e., may fire) at marking M, if its input places hold at least as many tokens as required by the weight of the corresponding edges, i.e., if $\forall p \in \bullet t : M(p) \geq w(p, t)$ holds. The firing of a transition passes (removes and produces) the defined number of tokens from its input places ($\bullet t$) to its output places ($t\bullet$), respectively. Formally, the reached marking can be computed as $\forall p \in P : M'(p) = M(p) - w(p, t) + w(t, p)$

The Petri net model of a simplified transportation system with three suppliers and four pieces of goods in the initial state can be seen in Fig. 1. All the edges are 1-weighted, i.e., each supplier in `supp` is able to transport one piece of goods to the destinations stores `s1`, `s2`, and `s3` through routes `r1`, `r2`, and `r3` such that `r3` cannot be reached directly from the starting point `supp` but only either through `r1` or `r2`. After shipping the goods, the suppliers return back to the starting point.

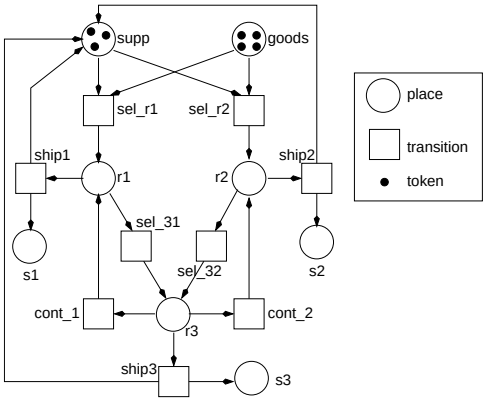

Figure 1: Petri net model of a transportation system

The affect of firing a transition can be extended for a firing sequence $s =< t_{i_1}, t_{i_2}, \ldots, t_{i_n} >$. Let us define the adjacency matrix of the net as $W = [-w(p_i, t_j) + w(t_j, p_i)]_{ij}$, where W is a $|P| \times |T|$ dimensional matrix.

Let us also define the Parikh vector of a firing sequence as a $|T|$-dimensional vector σ, where the i-th component counts the firing number of transition t_i in the firing sequence (i.e., $\sigma(i) := |t_{j_k} \in s : j_k = i|$. Then a firing sequence having the Parikh vector σ leads to the marking M' computed by the so-called state equation, $M' = M + W \cdot \sigma$.

2.2 Optimal trajectory problem

In Petri net-based analysis, a focal problem is the so-called reachability problem. The reachability problem in Petri nets is to decide whether a given state is reachable from a given initial state by a fireable trajectory. Frequently, in complex system models only the state of a subset of places is relevant from the practical point of view. This way the reachability problem is restricted during the partial reachability analysis to decide whether a state covering a given substate is reachable.

Due to their expressive power, Petri nets facilitate the modeling of resource allocation and scheduling problems by introducing quantitative parameters, like cost. As transitions typically represent operations in the system, we restrict ourselves to cost functions linear in the number of executed transitions. This way, the state equation forms a *(mixed) integer*

linear programming problem describing an optimal trajectory problem.

The objective of the optimization problem is to find a fireable (executable) trajectory from the initial state to the final (sub)state (or to the set of final states in the case of a partial reachability-styled problem) of an optimal cost.

In our running example, an optimal trajectory problem can be formulated as follows. The task of the suppliers is to place goods in the stores in the predefined amount, e.g., initially, there are three suppliers and four pieces of goods, and $1, 1, 1$ pieces of goods have to be placed in `s1`, `s2`, and `s3`, respectively. This task constitutes a partial reachability problem (because we do not care about where the suppliers stop their shipping).

Introducing shipping or traveling costs into the model as the cost of the firing of the corresponding transition, the optimal trajectory problem is to find the optimal shipping itinerary. In Fig. 2, cost values of the transition firings are shown in the rectangulars representing the transitions.

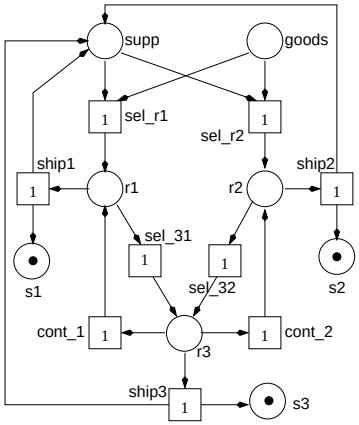

Figure 2: Optimal trajectory problem: desired end substate and cost values

One of the main advantage of the introduced modeling approach is, that optimization and verification problems (like the question whether the suppliers return back to their starting point) are modeled simultaneously. In addition, we can formulate temporal logic conditions like the following: if an order of shipping two pieces of goods occurs then it has to be executed eventually (by some suppliers).

There are two obvious ways to solve the optimal trajectory problem: the traversal of the state space of the problem or the computation of the optimal solution of a linear programming problem. However, in case of large, complex systems, the exhaustive search for a trajectory can result in state space explosion. On the other hand, the second way provides only a semi-decision technique to determine the optimal trajectory for the optimal trajectory problem (see Section 5).

In order to develop efficient search methods, another similar mathematical paradigm, the Process Network Synthesis methods was investigated.

3 Process Network Synthesis

In chemical engineering, PNS algorithms are used to determine an optimal resource allocation and scheduling for the production of desired products from given raw materials [1].

3.1 Problem definition

A PNS problem is represented by the so-called P-graph. A P-graph $\langle M, O\rangle$ is a bipartite graph where the two sets of disjoint nodes are materials M and operating units O, respectively. Materials can be raw materials (R), products (P) or intermediate materials ($M \setminus R \setminus P$). An operating unit consumes its input materials in order to produce its output materials.

Then, the PNS problem (P, R, O) of a P-graph $\langle M, O\rangle$ constitutes to produce all products in P from raw materials R by operating units O.

In Fig. 3 an example P-graph is shown where $M = \{A, B, C, D, E, F, G, H\}, O = \{1, 2, 3, 4, 5, 6\}$. Product $P = \{G\}$ has to be produced by the operating units, from raw materials $R = \{A, B, D\}$. The rest of the materials are intermediate materials C, E, F or byproduct H.

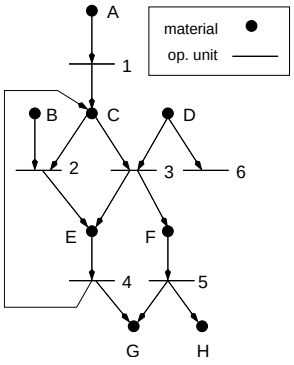

Figure 3: Example P-graph

The solution of the problem is represented by a sub-P-graph where all of the desired products are present and they are produced by the involved operating units. A feasible solution structure has to satisfy certain properties. These requirements are expressed by the following five axioms.

1. (A1) Every final product is represented in the graph.
2. (A2) A material has no input if and only if it represents a raw material.
3. (A3) Every operating unit represents an operating unit defined in the synthesis problem.
4. (A4) Every operating unit has at least one path leading to a final product.
5. (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.

Obviously, the constraints in the PNS problem formulation *do not introduce any restriction on the intermediate material stores after finishing the production*. However, the proper selection of the objective function will confine the solution space to 'reasonable solutions'. This way a PNS problem strongly resembles to a partial reachability problem as it accepts all of the solutions that produce the desired products from the given raw materials not regarding the produced but unused intermediate materials.

The synthesis of the optimal solution structure of a PNS problem is performed by three PNS sub-algorithms: the Maximal Structure Generation algorithm generating the superstructure of the feasible solutions according to the above axioms, the Solution Structure Generation algorithm computing all the structurally feasible networks, and finally, the Accelerated Branch and Bound algorithm dealing with the search for optimal production.

3.2 PNS algorithms

Based on the specific structure of the PNS problem, the combinatorial optimization problem can be solved efficiently by the exploitation of the features given by the five axioms.

Maximal Structure Generation (MSG) Algorithm The MSG algorithm excludes those materials and operating units that violate any of the five axioms thus achieving a significant reduction of the solution space in polynomial time. The resulting maximal structure is the union of all solution structures and it is itself a feasible network.

The maximal structure of the example in Fig. 3 is delivered by excluding the operating unit 6 because it does not satisfy axiom $(A4)$: there is no path from operating unit 6 to any desired product. The main advantage of allowing redundant problem formulations is on the engineering side as the engineer can simply formulate all the production possibilities without dealing with the extra constraints confining the solution space.

Solution Structure Generation (SSG) Algorithm After removing the redundant elements from the P-graph all the solution structures are generated.

During the computation, the SSG algorithm maintains a 'to be produced' set of materials. Initially, this set contains only the products. Then the algorithm recursively builds up a set of possible operating units.

The iteration consists of two main steps: at first, a material from the set 'to be produced' is selected (and also excluded from the set). Then, the operating units producing the selected material are taken into consideration: a subset of them is added to the already marked operating units. After that, the set 'to be produced' is extended with their input materials. Finally, the algorithm calls itself recursively.

The algorithm delivers all of base solutions that are closed under unification and an arbitrary solution structure can be generated combining these elementary solutions under the operations of unifications. Algorithm SSG generates every solution structure and only the solution structures of the problem as the span graph of the set of the selected operating units.

Accelerated Branch and Bound (ABB) Algorithm While the SSG algorithm generates all the solution structures regarding to the structural properties and neglecting the quantitative parameters, the ABB algorithm computes the optimal solution network of a minimal cost fulfilling simultaneously the quantitative constraints added to the operations and the material stores.

When selecting linear constraints and objective functions PNS problems can be interpreted as integer linear programming problems. The ABB algorithm solves this integer linear programming problem exploiting the additional structural properties (the same way as in the SSG algorithm) in the following way.

- *Bounding (numerical cut)* is conventional: if there is a known solution, its cost value serves as a lower limit for the further computations, i.e., a branch with a greater value is cut, and only the branches with lower value are taken into consideration.

- *The branching method (logical cut)* is based on algorithm SSG: branching is done at the selection of the operating units that produce the selected material from the 'to be produced' set, i.e., the solution structures generated by algorithm SSG provide a basis for algorithm ABB.

  The main advantage of the ABB algorithm is that it uses combinations of the elementary solutions instead of performing trials with individual elementary operations. This way the set over which the combinations are searched for is reduced to the set of a few solution structures instead of the huge number of potential elementary operations.

The result of the algorithm is a vector representing the contained operating units in the optimal network together with their operating rate. Exploiting the specific structure of the problem, the ABB algorithm achieves an essential improvement by the structural cuts in contrast to the conventional algorithm that traverses all of the $2^{|O|}$ vectors representing the subsets of operating units in the worst case.

In the following section we will discuss the adaptability of PNS methods for the Petri net reachability problem.

4 Adaptation of PNS algorithms to Petri nets

Based on the resemblance between PNS approach and Petri net partial reachability problem, our aim was to apply PNS algorithms in order to solve the optimal trajectory problem [3].

Petri Nets	P-graph of a PNS problem
places (P)	material containers
transitions	operating units
weight of incoming and outcoming edges of the corresponding transition	production rate for operating units
characteristic vector of the Parikh vector	set of the corresponding operating units
Parikh vector	result vector of the ABB algorithm

Table 1: Correspondence between Petri net and P-graph elements

4.1 Resemblance of problem definitions

For the sake of simplicity, we discuss the case when cost parameters are only assigned to transition firings in the optimal trajectory problem.

Let $\langle P, T, w, M_0 \rangle$ denote a Petri net model with a cost function $c : T \to \mathbb{R}_0^+$ associated to the transitions. Let M denote the (sub)state to be reached through an optimal trajectory. The analogy of the to Process Network Synthesis problem elements is sketched in Table 1.

Please, observe, that PNS problems focus only on the determination of the optimal production of the desired end products without considering the unspent intermediate materials and byproducts unless the objective function requires the zero amount of them. Since the start and end states of an optimal trajectory have only to satisfy the condition to cover the given initial and end (sub)states not regarding to the token numbers in the other places, optimal trajectory problems can be expressed as PNS problems.

Table 2 describes a Petri net optimal trajectory problem as a PNS problem.

4.2 Adaptation problems

Petri nets cover a wider range of models than those which can be represented by PNS problems. Namely, P-graphs and Petri nets have the similar structure but not all Petri net reachability problems can be directly 'transformed' into a PNS problem and thus performing the ABB algorithm on it. One reason for the limited expressiveness of the PNS problems are the constraints introduced by the axioms (see Section 3.1) in the transformed reachability problem.

'Produced raw materials' In PNS problems all the raw materials have to be represented by source places which are initially marked and do not have incoming edges. In the initial marking of the Petri net model an arbitrary place may contain tokens initially. In case of an initially marked place that is not a source place, the direct transformation of the Petri net would result in a raw material that is produced prohibiting axiom $(A2)$.

However, formally any Petri net with a known initial marking can be transformed to the convenient PNS format in such a way that an additional source place is initially marked having a single output transition placing tokens into each place which was marked in the original net. Hence 'produced raw materials' are eliminated.

Revisiting our Petri net example in Section 2.1, the convenient PNS model of the derived Petri net satisfying the PNS axioms is depicted in Fig. 4. Please observe, that place `goods` does not require such an initialization because it has no input transition.

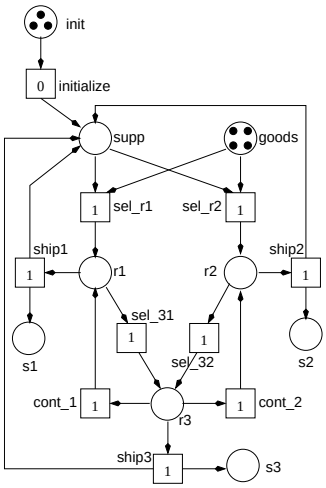

Figure 4: Petri net model with no 'produced raw material'

The other four axioms refer to the redundancy of a material or an operating unit. These redundancy requirements have to be also hold by Petri nets. Now, let us examine what solutions are generated for the above Petri net.

Let us imagine an optimal trajectory problem, with four pieces of goods, initially (we do not care about the number of suppliers in place `supp`, i.e., about the number of tokens in place `init`). The task is to ship $1, 1, 1$ pieces of goods to `s1`, `s2`, and `s3`, respectively. It is easy to observe, that there is *no* trajectory from the given initial state if place `init` contains no token.

However, the optimal solution generated by the ABB algorithm (firing transitions `sel_r1`, `sel_r2`, `sel_31`, `ship1`, `ship2` and `ship3`, $2, 1, 1, 1, 1, 1$ many times, respectively) does not contain transition `initialize`, that is, it is not fireable from the given initial state.

optimal trajectory problem	PNS problem
places marked in the initial state	raw materials
places marked at the state to be reachable	desired products
number of tokens	quantity of materials
costs assigned to transitions	operation cost of operating units

Table 2: Petri net (partial) reachability as a PNS problem

Catalysts Since PNS algorithms are originally developed to solve chemical resource allocation problems, they do not care about the presence of catalyst materials. In the transformed PNS problem place `supp` behaves as a catalyst: it is both consumed and produced during the manufacturing process resulting in a total of zero change in amount.

However, while such catalysts cannot be assessed based upon material bill like equation used in the PNS model, PNS algorithms suppose those materials to be available at the beginning. Since our aim is not to model chemical production systems but to model complex IT systems, unfireable spurious solutions (involving unavailable catalysts) have to be filtered.

Finally, let us discuss the meaning of the results of the adapted ABB algorithm to the Petri net model. For PNS problems, the ABB algorithm returns a vector representing the optimal production of the desired products as the operation duration of the involved operating units. Then the required amount of raw materials, in order to produce the given optimal production is calculated. This amount is estimated as the sum of the operation rates of the operating units that consume raw materials. This sum is unique as a consequence of axiom (A2).

In order to execute the production, catalysts are also necessary, but their initial amount does not have to be calculated because they remain in the network as unused materials. Therefore the amount of the required raw materials is enough to perform the production.

In case of Petri net models, catalysts cannot be *supposed* to be available at the beginning but they have to be present 'physically' in the net. Therefore, in the optimal trajectory problem, we have to know the required amount of catalyst tokens beside necessary amount of the initial tokens. On the other hand, in many real problems the exact trajectory has to be computed providing the *order* of the execution steps that is not provided by the ABB algorithm.

5 Solution algorithm for the optimal trajectory problem

In the following we introduce a gradual filtering method to compute the optimal fireable solutions for the optimal trajectory problem.

Our problem has two orthogonal aspects: one requiring *the fireability of the solution* in order to be feasible, and the other one requiring the *optimal cost*. These two aspects have to be merged into a single algorithm. This merging can be done at different levels of granularity: a straightforward solution would be to estimate, for instance, an optimal solution, to check subsequently its fireability and to return to a search for the second best solution if the fireability constraint is not satisfied. However, such rough granular merging of the algorithms may lead to a very high level of redundant computations. This way our objective is to detect infeasibility of the solutions as early as possible.

Our proposal consists of four consecutive steps.

1. At first, a candidate optimal solution as a Parikh vector is generated by the adapted ABB algorithm (already discussed in Section 4. This is a sufficient condition as discussed in Section 4.2.

2. Secondly, if there exists a solution, the end state reached from the initial state by the candidate solution Parikh vector can definitely be calculated by means of the state equation.

3. In order to assure the fireability of the solution derived by the adapted ABB algorithm, it is tested by a reduced check partitioned into a fast symbolic reachability check,

4. and a fireability check (together with the examination of the satisfaction of the temporal conditions).

Dependent on the result of the checking methods, the algorithm either terminates delivering a fireable optimal solution or the adapted ABB algorithm generates the next optimal solution and the introduced check is performed again.

5.1 Reachability check

The reachability check of the end state from a given initial state is performed by symbolic techniques using Binary Decision Diagrams.

- One solution is the symbolic state space traversal by the so-called image computation elaborated by Pastor, Cortadella and Roig [7]. The algorithm builds the state space starting from the given initial state firing simultaneously all the enabled transitions from the set of the reachable states.

  The main disadvantage of using this method is that in case of an initial *sub*state the state space has to be generated for each candidate solution freshly.

- The other solution is the calculation of a membership function based on the transitive closure of the single step transition function. The main advantage of this solution is that the membership function can be calculated independently from the actual initial state. In this way the membership function is reusable for the next candidate solutions generated by the ABB algorithm.

In the following the sketch of an algorithm using the second approach is given.

5.2 Membership function generation

For the reachability check of the solution computed by algorithm ABB, we use the transitive closure computation introduced in [6]. As the input of the computation, the Boolean representation of Petri net transition relation is used encoding the states and the dynamic behavior of Petri nets as Boolean functions.

For the sake of simplicity, let $PN = \langle P, T, w, M_0) \rangle$ be a safe Petri net, i.e., for every marking M reachable from the initial marking M_0, for all places $M(p) \leq 1$ holds. An M marking is encoded by the set of $|P| = n$ boolean variables such that each variable has the values 1 if and only if the corresponding place is marked at the marking M, i.e., $M = (p_1, \ldots, p_n) : p_i = 1 \iff M(p_i) = 1$.

A transition t is enabled in marking M, if the following expression is evaluated as true: $E_t = \bigwedge_{p_i \in \bullet t} p_i$. After having the condition for the enabledness we may define the result marking $M' = (\delta_1^t(p_1), \ldots, \delta_n^t(p_n))$ from the marking $M = (p_1, \ldots, p_n)$ firing transition t as follows.

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

Then the transition relation function is the following. The state $M' = (q_1, \ldots, q_n)$ is reachable from state $M = (p_1, \ldots, p_n)$ by firing a transition if the following function is evaluated to 1.

$$T(M, M') = \bigvee_{\forall t \in T} \left[\bigwedge_{i=1}^{n} \left(q_i \equiv \delta_i^t(p_1, \ldots, p_n) \right) \cdot E_t \right] \quad (1)$$

In order to compute the membership function for the reachability test, the transitive closure of the above defined T function is estimated by the algorithm published in [6]. The algorithm computes the transitive closure of a matrix representing the transition relation function, where the rows and the columns are indexed by the current state and the next state, respectively.

The algorithm, called *recursive descent procedure* is based on Shannon's theorem:

$$f = x f_x \vee \overline{x} f_{\overline{x}} \quad (2)$$

where the subscripts refer to the operation of cofactoring, e.g.,

$$f_{x,\bar{y}} = f_{|x=1,y=0} \quad (3)$$

In the sequel, recursive descent procedure is sketched (where $R \circ S$ stands for $\exists z[R(x,z)S(z,y)]$).

compute_closure(T) $\{$
if $T \in \{0, 1\}$ **then** return T;
$T = \begin{pmatrix} T_{\overline{x_1}\overline{y_1}} & T_{\overline{x_1}y_1} \\ T_{x_1\overline{y_1}} & T_{x_1 y_1} \end{pmatrix}$;
$U_1 = compute\_closure(T_{x_1 y_1})$;
$U_2 = U_1 \circ T_{x_1\overline{y_1}}$;
$E = compute\_closure(T_{\overline{x_1}\overline{y_1}} \vee (T_{\overline{x_1}y_1} \circ U_2))$;
$U_3 = T_{\overline{x_1}y_1} \circ U_1$;
$F = E \circ U_3$;
$G = U_2 \circ E$;
$H = U_1 \vee (U_2 \circ F)$;
return $\overline{x_1}\overline{y_1}E \vee \overline{x_1}y_1F \vee x_1\overline{y_1}G \vee x_1y_1H$
$\}$

The initial and the calculated end states are substituted into the generated membership function. If the test is negative, a next optimal solution is generated by the ABB algorithm. However, this reachability check provides only a semi-decision method (because the analyzed marking could be reached by another trajectory). It has to be proved that the end state can be reached by a trajectory corresponding to *this solution vector* retrieved by the ABB algorithm as a Parikh vector, i.e., a fireability check has to be performed.

5.3 Fireability check and temporal logic conditions

The last step of our algorithm is to prove the fireability of the candidate solution vector. If the solution is fireable, then as the very last step, the satisfaction of temporal logic conditions can be decided. For this purpose, the model checker *SPIN* was chosen.

SPIN was originally developed to model computer and network protocols. In the last years, SPIN was also adapted successfully into many application domains such as specification and design verification of both hardware and software systems.

The main advantage of SPIN is its own specification language *Promela* (Process Meta Language). Promela is appropriate to describe concurrent processes and it contains several control structures. The requirements to be satisfied by the system are given by linear temporal logic formulae. Verifying an LTL logic formula, SPIN reports either the satisfaction of the formula or it generates a counterexample.

To perform the fireability check, at first the Petri net model (without the cost function) is translated into a Promela model using the embedded translation of PEP (Programming Environment based on Petri nets) [2].

In order to check the fireability of the candidate Parikh vector σ, the new version of SPIN, *SPIN 4.0* [5] is used. The

new C primitives of SPIN 4.0 facilitate the use of C variables in the Promela model, thus an LTL property to be checked can be dynamically changed during the verification.

We use a $computed.\sigma$ variable that register the times of the execution of the individual transition firings (representing as atomic expressions) for each visited state. Thus, we formulate the appropriate LTL logic formulae as $always(\sigma \neq computed.\sigma)$. Namely, we let SPIN to search for a trajectory (reached state) that violates the statement. If SPIN generates a counterexample, there exists a trajectory having the candidate Parikh vector together with the order of the transitions firings, otherwise the check is negative.

In order to improve this method, we may add a depth constraint for the search space as we know that the firing sequence consists as many steps as the sum of the components of the Parikh vector.

In addition, introducing costs of the transitions as new variables promises a further improvement providing numerical cuts beside the logical cuts performed by SPIN. Namely, the search is terminated if either the given depth is reached or the corresponding cost of the given Parikh vector is proceeded.

Recently, the Branch and Bound technique was implemented using SPIN in [8] in order to provide optimal scheduling and verification. Although the discussed approach is appropriate to perform the search for optimal scheduling (especially combined with embedded heuristics into the Promela model), it can be very inefficient if there is no proper solution.

5.4 Running example

Let us revisit our example in Section 4.2 (depicted in Fig. 4). We have four pieces of goods, initially (and we do not care about the number of suppliers). The task is to ship $1, 1, 1$ pieces of goods to stores `s1`, `s2`, and `s3`, respectively.

As discussed, the candidate Parikh vector generated by the ABB algorithm with optimal cost 7 is the following. Transitions `sel_r1`, `sel_r2`, `sel_31`, `ship1`, `ship2` and `ship3` fire $2, 1, 1, 1, 1, 1$ times, respectively. The generated solution yields the initial state that has three tokens in place `goods`, while places `s1`, `s2` and `s3` contain $1, 1, 1$ tokens in the calculated end state, respectively.

The check of reachability fails because there is no fireable trajectory from this initial state therefore, the calculated end state cannot be reached.

The Parikh vector of the next best solution generated (with optimal cost 7) is as follows. Transitions `initialize`, `sel_r1`, `sel_r2`, `sel_31`, `ship1`, `ship2` and `ship3` fire $3, 2, 1, 1, 1, 1, 1$ times, respectively. The generated solution yields the initial state that has three-three tokens in places `init` and `goods`, while places `supp`, `s1`, `s2` and `s3` contain $3, 1, 1, 1$ tokens in the calculated end state, respectively. With the calculated initial and end states, both of the reachability and the fireability check (yielding the firing order $3\times$`initialize`, $2\times$`sel_r1`, `ship1`, `sel_31`, `ship3`, `sel_r2`, `ship2`) are positive, i.e., we found the optimal trajectory.

6 Ongoing research activities

In order to make the use of the above algorithm available for system designers who is not familiar with mathematical modeling and optimization problems, we are aiming to develop an automatized framework for *UML-based* modeling of optimization and verification problems [4].

The Petri net model corresponding to the UML representation is derived by mathematical model transformations automatically [9], and the Petri net model of the problem is solved by the proposed technique. In this way, system architect may design a system in their well-known design environment and our combined verification and optimization method is carried out in a push-button way.

References

[1] F. Friedler, J. B. Varga, E. Feher, and L. T. Fan. Combinatorially Accelerated Branch–and–Bound Method for Solving MIP Model of Process Network Synthesis, Nonconvex Optimization and its Applications. *State of the Arts in Global Opimization, Computational Methods and Applications*, pages 609–626, 1996.

[2] B. Grahlmann and C. Pohl. Profiting from Spin in PEP. In *SPIN'98 Workshop*, 1998.

[3] S. Gyapay and A. Pataricza. Optimization Methods for Reachability Analysis of Petri Net Models. In *Proc. of FORMS-2003, Budapest, Hungary, May 15-16)*, pages 53–60. L' Harmattan, Budapest, 2003.

[4] S. Gyapay, A. Pataricza, J. Sziray, and F. Friedler. Petri Net-based Optimization of Production Systems. In 6^{th} *IEEE Int. Conf. on Intelligent Engineering Systems*, pages 465–469. Opatija, Croatia, May 26–28 2002.

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

[6] Y. Matsunaga, P. C. McGeer, and R. K. Brayton. On Computing the Transitive Closure of a State Transition Relation. In *30th ACM/IEEE Design Automation Conference, Dallas, Texas, United States*, pages 260–265, 1993.

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

[8] T. C. Ruys. Optimal Scheduling using Branch and Bound with SPIN 4.0. In *Proc. 10th Int. SPIN Workshop, Portland, OR, USA, May 9-10, 2003. Proceedings*, vol. 2648 of *LNCS*, pages 1–17. Springer, 2003.

[9] D. Varró, G. Varró, and A. Pataricza. Designing the automatic transformation of visual languages. *Science of Computer Programming*, 44(2):205–227, August 2002.