Chapter 20

Introduction to Net-Driven Decomposition Techniques

As stated in previous chapters, a fundamental question in the use of stochastic Petri net models for performance evaluation, even under Markovian stochastic interpretation, is the so called *state explosion problem.* A general approach to deal with (computational) complexity is to use a *divide and conquer* strategy, what requires the definition of a decomposition method and the subsequent composition of partial results to get the full solution. On the other hand, the trade-off between computational cost and accuracy of the solution leads to the use of approximation or bounding techniques (for instance, throughput bounds can be computed in polynomial time on the number of transitions and places, see Chapter 17). In this context, a pragmatic compromise to be handled by the analyzer of a system concerns the definition of faithful models, that may be very complex to exactly analyse (what may lead to the use of approximation or just bounding techniques), or simplified models, for which exact analysis can be, eventually, accomplished. Divide and conquer strategies can be used with exact, approximate, or bounding techniques.

The techniques for performance evaluation present in the literature consider either implicit or explicit decomposition of net models. In [7] (see Chapter 17), an implicit decomposition into P-semiflows is used for computing throughput bounds for arbitrary pdf of time durations. In [19] (see Chapter 12), a decomposition into disjoint *modules* (usually subnets generated by P-semiflows) is defined by the analyzer or provided by model construction; the computational technique uses directly the information provided by the modules to compute exact global limit probability distribution vector. In [8], a decomposition into *modules* (connected through buffers) should also be provided. In this case, the modules are complemented with an abstract view of their environment in the full model, and the approximate solution is computed through an *iterative technique* looking for a fixed point (details will be presented in Chapter 23). In the sequel of this Chapter and in a general context, *components* will refer to (eventually) complemented modules. They are just the elements used to build the full solution.

In order to get efficient techniques, the decomposition and, eventually, complementation process should be net-driven (i.e., derived at net level). For this, we shall use PN's structure theory concepts and techniques (e.g., P-semiflows, implicit places, etc.).

The Chapter is organised as follows. The main ideas behind net-driven decompositions of PN's are introduced in Section 20.1: the conservative and consistent components (Section 20.1.1), an example of implicit search technique into conservative components (Section 20.1.2), an explicit decomposition of nets, designating the modules to be used (Section 20.1.3), and the complementation of modules to get components (that include information from the environment), using implicit places (Section 20.1.4). A taxonomy for net-driven decomposition techniques is proposed in Section 20.2, providing a framework for the consideration of a significant number of performance evaluation methods. Several representative examples of techniques present in the literature are briefly overviewed and classified according with the classification criteria. Some concluding remarks are included in Section 20.3.

20.1 Basic Concepts and Techniques

In Section 20.1.1, the basic standard components of nets, conservative and consistent components, are derived from vectors (P- and T-semiflows) defined from algebraic properties of the incidence matrix $\mathbf{C}$ of the net. An implicit search technique into conservative components of a net is presented in Section 20.1.2. In other cases the decomposition must be done explicit, designating clearly the modules to be used (Section 20.1.3). Finally, the obtained modules can be complemented in order to (partially) re-introduce *synchronic dependences* that the decomposition removed. This is the topic of Section 20.1.4, where implicit places are used for the mentioned purpose.

20.1.1 Standard components

When a transition t is enabled at $\mathbf{m}$ ($\mathbf{m}[p] \geq \mathbf{Pre}[p,t]$) the new marking reached by its firing ($\mathbf{m} \xrightarrow{t} \mathbf{m}'$) is $\mathbf{m}' = \mathbf{m} + \mathbf{C}[P,t]$. If we integrate it over a sequence σ of fired transitions from the initial state $\mathbf{m_0}$ and yielding $\mathbf{m}$, denoting by $\boldsymbol{\sigma}$ the *firing count vector* of sequence σ ($\boldsymbol{\sigma}[t] = \#(t,\sigma)$ is the number of times t occurs in the sequence), we obtain (see Chapter 2):

$$\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma} \tag{20.1}$$

This equation resembles the state equation of a discrete time linear system, therefore it is frequently referred as the *net state equation.* Observe that some integer solutions of this equation may be non reachable in the net system (those are called "spurious solutions").

Premultiplying the state equation by $\mathbf{y} \geq \mathbf{0}$ such that $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$ —vector $\mathbf{y}$ is called a *P-semiflow*— then, for every initial marking $\mathbf{m_0}$, every reachable marking $\mathbf{m}$ satisfies:

$$\mathbf{y} \cdot \mathbf{m} = \mathbf{y} \cdot \mathbf{m_0} + \mathbf{y} \cdot \mathbf{C} \cdot \boldsymbol{\sigma} = \mathbf{y} \cdot \mathbf{m_0} = k \qquad (20.2)$$

This provides a "token conservation law": for every reachable marking the weighted sum of tokens in $\|\mathbf{y}\|$ remains constant, where $\|\mathbf{y}\| = \{p \mid \mathbf{y}[p] > 0\}$ is the *support* of vector $\mathbf{y}$.

Besides the invariant laws, a major interest of P-semiflows is the *decomposed view* of the model that they provide. The P-subnet generated by the support of a P-semiflow is called a *conservative component* of the net, meaning that it is a part of the net that conserves its weighted token content. A conservative component allows neither incoming nor outcoming of tokens (customers, resources, servers...). Thus conservative components are *closed* subsystems. If there exists a $\mathbf{y} > \mathbf{0}$ such that $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$, then the net is said to be *conservative*, thus bounded for every initial marking. It is important to realise that we introduce three notions that should be differentiated:

- the P-semiflow (a vector: $\mathbf{y} \geq \mathbf{0}$, $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$),
- the token conservation law or marking invariant (an equation: $\mathbf{y} \cdot \mathbf{m} = \mathbf{y} \cdot \mathbf{m_0}$), and
- the conservative component (a net: the subnet generated by $\|\mathbf{y}\|$ and its input and output transitions).

By definition of P-semiflow, if there is at least one for a given net, then there exists an infinite number of them. All the information contained in the token conservation laws can be extracted from a subset of P-semiflows called *minimal P-semiflows.* A P-semiflow is said to be minimal when its positive entries $\mathbf{y}_i$ are relatively prime and no P-semiflow $\mathbf{y}'$ exists such that $\|\mathbf{y}'\| \subset \|\mathbf{y}\|$. The set of all the minimal P-semiflows, called the *fundamental set* of P-semiflows, is unique (see [16] for its computation).

Let us consider the net depicted in Figure 20.1.a. The computation of minimal P-semiflows gives a fundamental set with two elements: $\mathbf{y_1} = (1, 0, 1, 1, 0)$ and $\mathbf{y_2} = (1, 1, 0, 0, 1)$. The corresponding token conservation laws are: $\mathbf{m}[\mathsf{p1}] + \mathbf{m}[\mathsf{p3}] + \mathbf{m}[\mathsf{p4}] = 2$ and $\mathbf{m}[\mathsf{p1}] + \mathbf{m}[\mathsf{p2}] + \mathbf{m}[\mathsf{p5}] = 1$. The two conservative components associated to the above P-semiflows are depicted in Figure 20.1.b and Figure 20.1.c.

The dual notion of P-semiflows are T-semiflows. If $\mathbf{x} \geq \mathbf{0}$ is such that $\mathbf{C} \cdot \mathbf{x} = \mathbf{0}$:

$$\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \mathbf{x} = \mathbf{m_0} \qquad (20.3)$$

T-semiflows correspond to cyclic sequences in the sense that the firing count vector of a cyclic sequence of a bounded system is a T-semiflow. Nevertheless, it should be pointed out that eventually for a given initial marking may be *not* possible to fire a sequence whose firing count vector is a given T-semiflow.

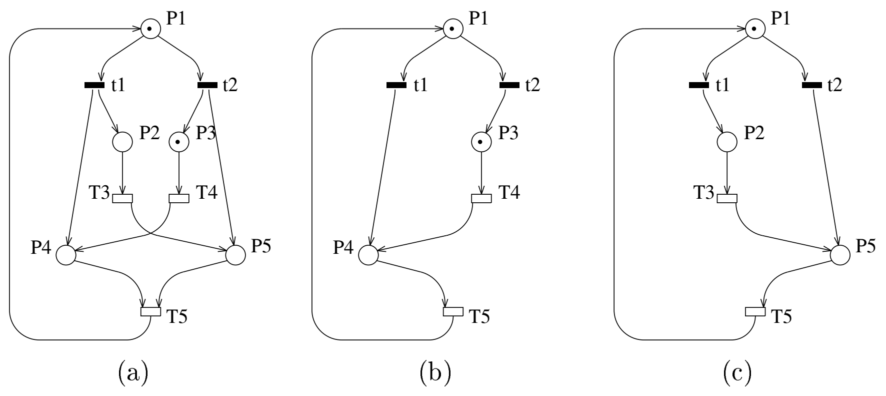


Figure 20.1: A net (a) and its conservative components (b) and (c).

For example, for the net in Figure 20.1.a., two minimal T-semiflows can be computed (using the same algorithm for computing P-semiflows but applied on the transposed incidence matrix): $\mathbf{x_1} = (1, 0, 1, 0, 1)$ and $\mathbf{x_2} = (0, 1, 0, 1, 1)$.

The T-subnet generated by the support of a T-semiflow is called a *consistent component* of the net. Consistent components provide an alternative decomposed view of the net.

In the case that $\mathbf{x} > \mathbf{0}$ such that $\mathbf{C} \cdot \mathbf{x} = \mathbf{0}$ exists, the whole net is a consistent component, and it is said that the net is *consistent*.

When a net is not consistent it cannot be lively and boundedly marked (see, for example, [32]).

20.1.2 P-Semiflows and Performance Bounds: An Implicit Search in Conservative Components.

The *visit ratio* of transition t_j with respect to t_i, $\mathbf{v}^{(i)}[t_j]$, is the average number of times t_j is visited (fired) for each visit to (firing of) the reference transition t_i (see Chapter 8). The vector of visit ratios of a live and bounded system must be a T-semiflow (the input weighted flow to each place must be equal to the output weighted flow): $\mathbf{C} \cdot \mathbf{v}^{(i)} = \mathbf{0}, \mathbf{v}^{(i)} \geq \mathbf{0}$ (otherwise stated: $\mathbf{v}^{(i)} = \sum_j \lambda_{ij} \mathbf{x_j}$, $\mathbf{v}^{(i)}[t_i] = 1$). The visit ratios of transitions in *equal conflict*, i.e., such that their preconditions are the same ($\mathbf{Pre}[t_j] = \mathbf{Pre}[t_k]$) must be fixed by the corresponding routing rates. For instance, for the net in Figure 20.1.a, the following equation holds: $r_1 \mathbf{v}^{(i)}[\mathtt{t2}] - r_2 \mathbf{v}^{(i)}[\mathtt{t1}] = 0$, where r_1 and r_2 are the routing rates of `t1` and `t2`. In summary, the next result can be stated [12]: The vector of visit ratios with respect to transition t_i of a live and bounded net system must be a solution of:

$$\mathbf{C} \cdot \mathbf{v}^{(i)} = \mathbf{0}; \;\; \mathbf{R} \cdot \mathbf{v}^{(i)} = \mathbf{0}; \;\; \mathbf{v}^{(i)}[t_i] = 1 \tag{20.4}$$

where $\mathbf{R}$ is a matrix that relates the visit ratios of transitions in equal conflict ($\mathbf{Pre}[t_j] = \mathbf{Pre}[t_k]$) according to the corresponding routing rates.

Equations in the above statement have been shown to characterize a unique vector $\mathbf{v}^{(i)}$ for important net subclasses [7, 12] (a condition over the rank of $\mathbf{C}$ and the number of equal conflicts underlies these cases).

The *average service demand* from transition t_j with respect to t_i is defined as $\overline{\mathbf{D}}^{(i)}[t_j] = \mathbf{s}[t_j]\,\mathbf{v}^{(i)}[t_j]$.

From the classical Little's law (applied to the places of a SPN) and using the token conservation laws derived from P-semiflows the following result can be stated [7] (see Chapter 17): For any net system with infinite-server semantics assumed for transitions, a lower bound for the average interfiring time $\Gamma[t_i]$ of a transition t_i can be computed by solving the following linear programming problem (LPP):

$$\begin{array}{rll} \Gamma[t_i] \geq & \text{maximum} & \mathbf{y} \cdot \mathbf{Pre} \cdot \overline{\mathbf{D}}^{(i)} \\ & \text{subject to} & \mathbf{y} \cdot \mathbf{C} = \mathbf{0} \\ & & \mathbf{y} \cdot \mathbf{m_0} = 1 \\ & & \mathbf{y} \geq \mathbf{0} \end{array} \tag{20.5}$$

If the solution of the LPP (20.5) is unbounded and since it is a lower bound for the average interfiring time of transition t_i, the non-liveness can be assured (infinite interfiring time). If the visit ratios of all transitions are non-null (i.e., $\mathbf{v}^{(i)} > \mathbf{0}$), the unboundedness of the above problem implies that a total deadlock is reached by the net system. Anyhow, the unboundedness of the solution of (20.5) means that there exists an unmarked P-semiflow, and obviously the net system is non-live: if $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$ and $\mathbf{y} \cdot \mathbf{m_0} = 0$, then $\forall \mathbf{m}\ \forall p \in \|\mathbf{y}\|$: $\mathbf{m}[p] = 0$, and the input and output transitions of p are never firable.

The basic advantage of the LPP (20.5) lies in the fact that the *simplex method* for the solution of an LPP has almost linear complexity in practice, even if it has exponential worst case complexity. In any case, algorithms of polynomial worst case complexity can be found in [27].

In order to interpret the LPP (20.5), let us rewrite it as the following fractional programming problem where the interpretation in terms of P-semiflows is even more clear:

$$\begin{array}{rll} \Gamma[t_i] \geq & \text{maximum} & \dfrac{\mathbf{y} \cdot \mathbf{Pre} \cdot \overline{\mathbf{D}}^{(i)}}{\mathbf{y} \cdot \mathbf{m_0}} \\ & \text{subject to} & \mathbf{y} \cdot \mathbf{C} = \mathbf{0} \\ & & \mathbf{y} \geq \mathbf{0} \end{array} \tag{20.6}$$

Therefore, the lower bound for the average interfiring time of t_i in the original net system given by (20.6) is computed *looking at the "slowest subsystem" generated by the P-semiflows*, considered in *isolation* (with delay nodes).

Let us consider again the net system of Figure 20.1.a. Assuming that the vector of average service times of transitions (for arbitrary pdf's; i.e., Markovian assumption is not needed) is $\mathbf{s} = [0, 0, 1, 10, 3]$ and that the routing rates associated with `t1` and `t2` are identical, then the system (20.4) gives $\mathbf{v}^{(5)} =$

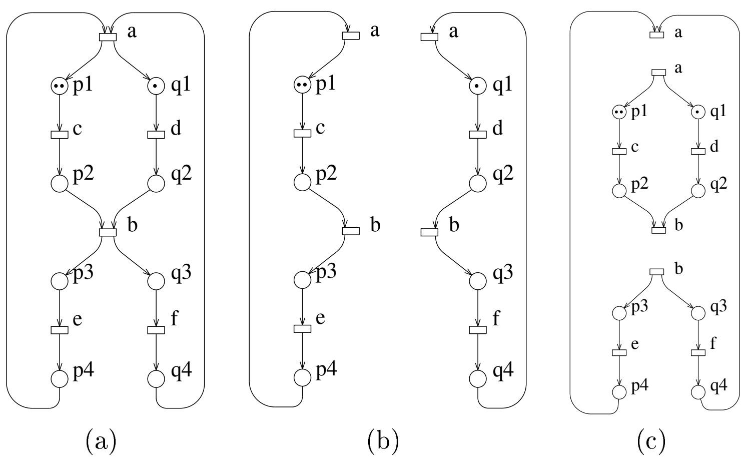


Figure 20.2: A SPN (a) and two possible decompositions (b,c) obtained by cutting through transitions a and b.

$(0.5, 0.5, 0.5, 0.5, 1)$ and the vector of average service demands for transitions (normalized for T5) is $\overline{\mathbf{D}}^{(5)} = (0, 0, 0.5, 5, 3)$.

The application of (20.5) or (20.6) to the minimal P-semiflows $\mathbf{y_1}$ and $\mathbf{y_2}$ gives:

$$\Gamma[\mathsf{T5}] \geq \max\{(5+3)/2, 0.5+3\} = 4 \qquad (20.7)$$

The quantities under the max operator in (20.7) represent, for this particular case, the average interfiring time of transition T5 at each of the two subnets depicted in Figures 20.1.b and 20.1.c (embedded queueing networks in this particular case) assuming that all the nodes are delay stations (infinite-server semantics).

The reader should notice that the above linear programming problem makes an *implicit search* of the slowest subsystem among those defined by the minimal P-semiflows; it is a *bottleneck* analysis.

20.1.3 Explicit Decomposition of Nets: Modules

The first step in any technique based on the divide and conquer approach for the analysis on SPN's is the decomposition of the model into two or more "pieces" or *modules.* Since PN's are bipartite graphs, the decomposition will consist basically in cutting the graph by selecting a set of vertices (places or transitions) that are going to be the *interface* (or border) among the different modules.

A typical way of decomposing systems is to consider *rendez vous* synchronizations. In PN terms, the basic case is obtained by cutting a transition in a "parallel" way with respect to the flow of tokens and leaving a process in each side (the number of input and output places in each side is "balanced"). Consider, for instance, the system depicted in Figure 20.2.a and assume that transitions a and b are going to define the cut. In Figure 20.2.b, the two obtained processes are depicted.

An alternative way of decomposing systems is to consider the splitting through buffers or mailboxes, leaving in one side the inputs and in the other the outputs. In this case, the cut is "orthogonal" with respect to the flow of tokens, in net terms. By refining the place that represents the buffer into a place-transition-place sequence, the above cut can be expressed by cutting also orthogonally to the flow of tokens in the introduced transition. For the same system in Figure 20.2, a cut orthogonal with respect to the flow of tokens through transitions a and b is depicted in Figure 20.2.c.

The first kind of compositions/decompositions introduced above are usually called *synchronous* while the second kind are referred as *asynchronous*. Figure 20.3.b shows an example of asynchronous decomposition of the system in Figure 20.3.a through places b1, b2, b3, b4. The model is split into three modules (subnets generated by places labeled with a, c and d, respectively).

A general cut of a PN through a transition can be defined by *partitioning* the set of input and output places of the transition into two or more subsets, each of one belonging to a different module. For example, a cut of the system in Figure 20.3.a through transitions TI1 (b1,b2/c2,c4), TI2 (b3,b4/c5,c6), TI5 (b2/d1,d2,d6,d10), TI6 (b4/d1,d5,d9,d13) is shown in Figure 20.3.c. The model is, in this case, also split into three modules (subnets generated by places starting with a or b for the first module, c for the second module, and d for the third one).

By means of T-P duality in nets, general cuts through places can also be defined. Nevertheless they need a more careful definition concerning the initial marking of the subsystems. In any case, they are usually less intuitive than the cuts through transitions.

20.1.4 Obtaining Components from Modules: Implicit Places

The above cuts decompose a net producing modules. In this section we present, in an informal way, the main ideas behind a technique for transforming modules into components. The main problem to solve is that "decomposition" means removing behavioural constraints, thus the state space of modules considered in isolation may be much larger (even unbounded) than when considered inside the full model (i.e., what corresponds to the projection of the full behaviour on the preserved nodes). Therefore, the natural idea is to restrict the behaviour of the modules, considered in isolation, with *synchronic constraints* inherited from the rest of the model. An alternative way of understanding the above complementation process of modules is to see that the added nodes implement a *reduction* of the behaviour of the rest of the model.

Using the above complementation process, the original system may be covered by components. In each component, *only one of the different modules of the original system is kept while the internal structure of the others is reduced as much as possible.*

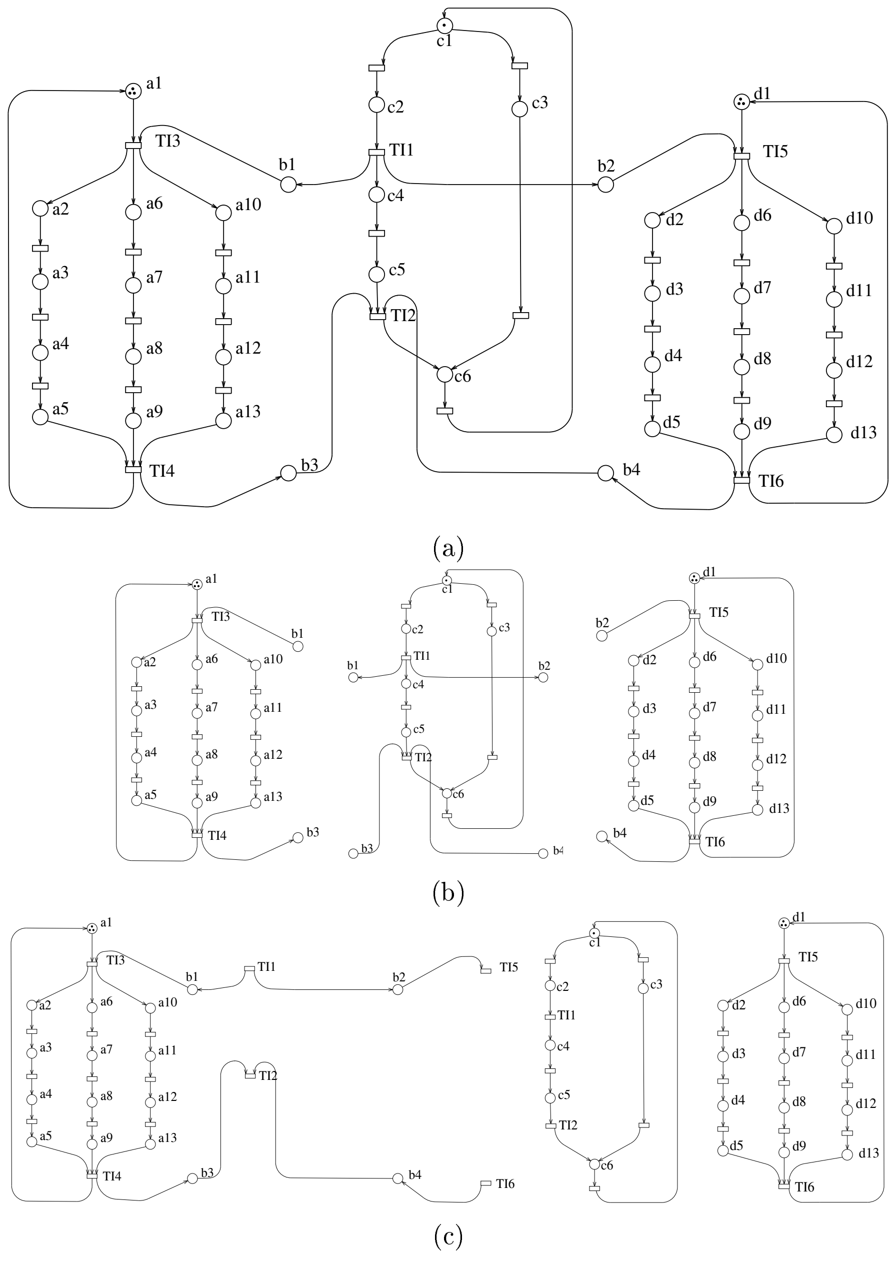


Figure 20.3: A SPN (a) and two possible decompositions obtained by cutting through (b) places b1, b2, b3, b4 and (c) transitions TI1, TI2, TI5, TI6.

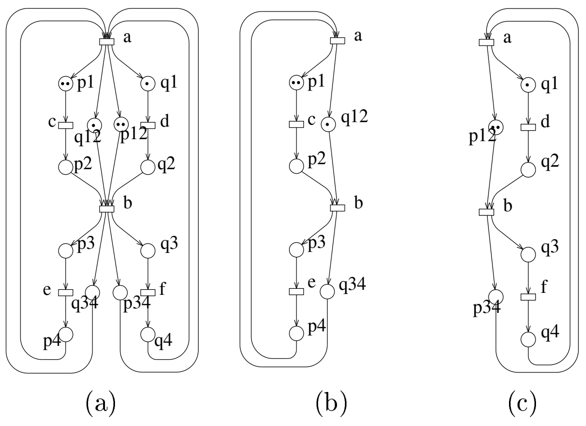


Figure 20.4: Deriving components from the modules in Figure 20.2.b.

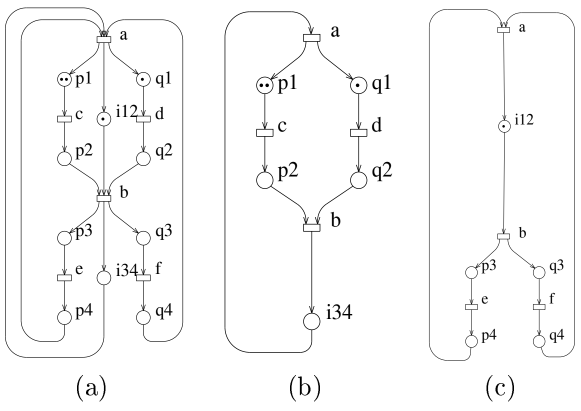


Figure 20.5: Deriving components from the modules in Figure 20.2.c.

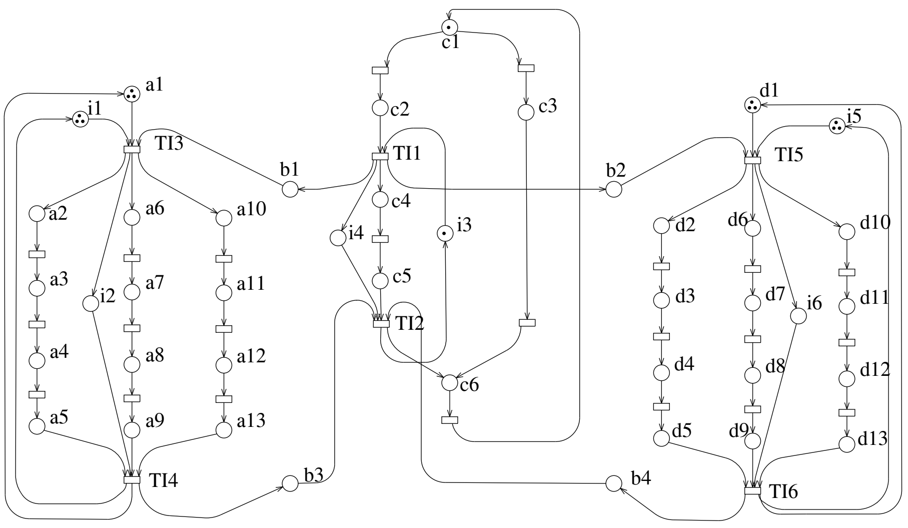


Figure 20.6: Extended system for the model in Figure 20.3.a.

Let us come back to the example in Figure 20.2. In order to complement the modules obtained in the first (synchronous) decomposition, depicted in Figure 20.2.b, an *extended system*, $\mathcal{ES}$, is derived from the original like that depicted in Figure 20.4.a. It consists of the original system plus the addition of some *implicit places*: q12 and q34 will be added to the module on the left (Figure 20.4.b) summarizing information from that on the right (respectively, p12 and p34 will be added to the module on the right as shown in Figure 20.4.c). An implicit place [17] (see Chapter 6) is one whose removal does not affect the behaviour of the system (therefore, behaviour of the original and of the extended systems is the same). Here, by behaviour we understand the *interleaving semantics*, i.e., the sequential observations or language [17], although the notion of implicit place can be directly extended to cope with a *step semantics* [15]. If a Markovian interpretation of PN's and single-server semantics of transitions is considered, the embedded CTMC of a system is preserved if implicit places are added or removed.

The extended system in Figure 20.4.a can be synchronously decomposed into the components in Figures 20.4.b and 20.4.c. The reader should notice that, except q12, the places computed as implicit in the original system (Figure 20.4.a) are also implicit in the components, so they could be deleted without changing their behaviour.

Consider now the second decomposition proposed in Figure 20.2.c for the same system in Figure 20.2.a. The modules can be complemented using the places i12 and i34 depicted in Figure 20.5.a. In this case, the addition of these places to the modules is crucial since it leads to bounded components (Figures 20.5.b and 20.5.c) while the original modules were unbounded.

After these introductory examples, let us concentrate in the decomposition process in a more general case, always staying semi-formal/illustrative. We adopt the so called SAM technique or view [11] (see Chapter 12), where modules are connected asynchronously through buffers (SAM stands for "System of Asynchronously communicating Modules"). Let us consider again the example in Figure 20.3.a distinguishing the three modules $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$ depicted in Figure 20.3.b, where places b1, b2, b3, b4 are the buffers. The extended system is depicted in Figure 20.6, where the labels of the added implicit places start with i. The way the implicit places to be added are computed is the following. First, an *equivalence relation* R is introduced in the set of places P_i of each module, partitioning P_i into equivalence classes P_i^j. Two places of a module are related by R iff there exists a non-directed path including only nodes from that module that does not include interface transitions. In the example, places in module $\mathcal{N}_1$ are partitioned into four equivalence classes, $P_1^1 = \{\mathsf{a1}\}$, $P_1^2 = \{\mathsf{a2}, \ldots, \mathsf{a5}\}$, $P_1^3 = \{\mathsf{a6}, \ldots, \mathsf{a9}\}$, and $P_1^4 = \{\mathsf{a10}, \ldots, \mathsf{a13}\}$, places in module $\mathcal{N}_2$ are partitioned into $P_2^1 = \{\mathsf{c1}, \mathsf{c2}, \mathsf{c3}, \mathsf{c6}\}$ and $P_2^2 = \{\mathsf{c4}, \mathsf{c5}\}$, while places in $\mathcal{N}_3$ are partitioned into $P_3^1 = \{\mathsf{d1}\}$, $P_3^2 = \{\mathsf{d2}, \ldots, \mathsf{d5}\}$, $P_3^3 = \{\mathsf{d6}, \ldots, \mathsf{d9}\}$, and $P_3^4 = \{\mathsf{d10}, \ldots, \mathsf{d13}\}$. Then, a set H_i^j (standing for "High-level places") of implicit places that include information of the behaviour on $\mathcal{N}_i$ is computed for each equivalence class P_i^j. For instance, in the example, $H_2^1 = \{\mathsf{i3}\}$ is the set (in this case only one place) of implicit places corresponding to the equivalence class $P_2^1 = \{\mathsf{c1}, \mathsf{c2}, \mathsf{c3}, \mathsf{c6}\}$. In some cases, some of the computed implicit places can be omitted. For instance, the implicit places corresponding to the equivalence classes P_1^2, P_1^3, and P_1^4 are identical (place i2), thus only one is added.

The components (low level systems, $\mathcal{LS}_i$, $i = 1, \ldots, K$) are derived by reducing all the modules $\mathcal{N}_j$, $j \neq i$, to their interface transitions and to the implicit places that were added in the extended system, while $\mathcal{N}_i$ is fully preserved. If needed, another component, the *basic skeleton*, $\mathcal{BS}$, can be derived by reducing *all* the modules in the same way as for $\mathcal{LS}_i$. The basic skeleton defines a more abstract view of the original system. Figure 20.7 shows the low level systems, $\mathcal{LS}_1$ $\mathcal{LS}_2$, and $\mathcal{LS}_3$ and the basic skeleton, $\mathcal{BS}$ for the running example. Notice that if $\mathcal{LS}_1$ $\mathcal{LS}_2$ and $\mathcal{LS}_3$ were synchronized by merging common transitions (interface transitions) and identifying common places (buffers and implicit places), the extended system (with equivalent behaviour to the original system) would be obtained. Notice also that the basic skeleton is the common abstraction among the three components.

20.2 A Taxonomy of Techniques

There exist a significant number of techniques proposed in the literature in order to compute performance indices (either as *bounds*, *approximations* or *exact* values), within divide and conquer strategies. In order to provide a framework for their consideration, we shall introduce some criteria inducing a certain taxonomy for the existing techniques. Representative examples of techniques will be briefly overviewed, providing some insight into the stochastic solution.

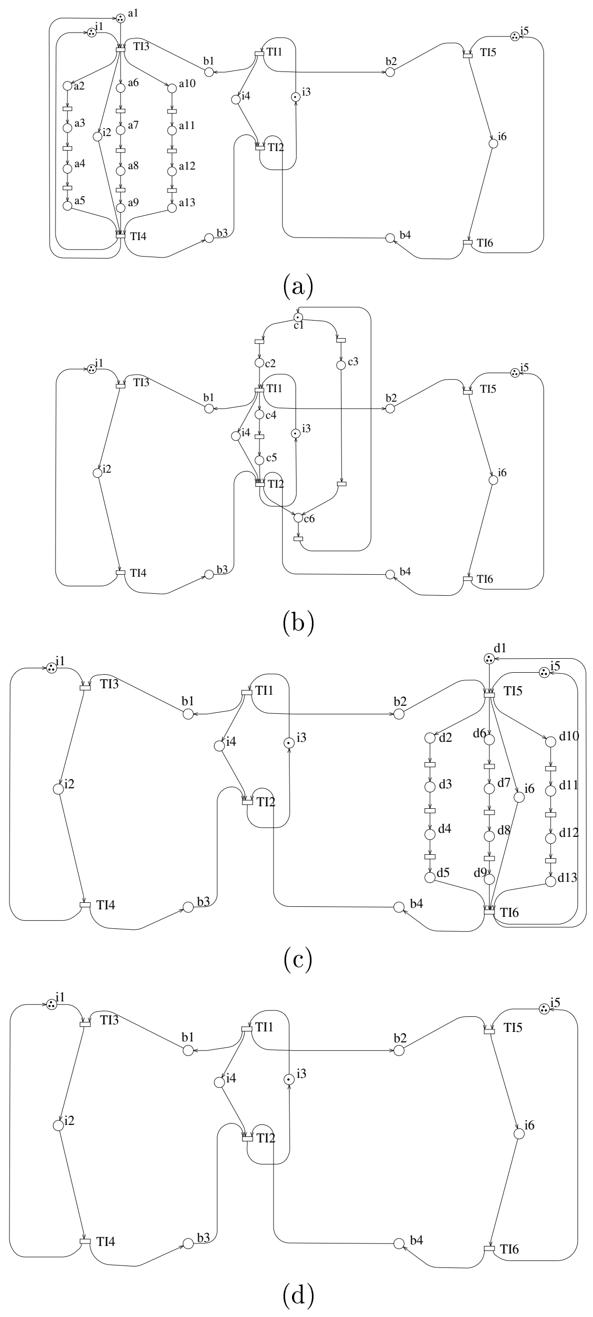


Figure 20.7: Low level systems (a,b,c) and basic skeleton (d) for the model in Figure 20.3.a.

20.2.1 The Criteria for Classification

A first criterium for the taxonomy used later concerns *the information that components have about their environment*. In some cases, the components are directly the modules obtained from the partition, thus they do not have any information of the rest of the system. Examples of this approach are: the exact analysis of SGSPN [19] (see Chapter 12), the flow equivalent aggregation for the approximate analysis of SPN's [20], or the linear programming based computation of performance bounds presented in Section 20.1.2 [7]. In other cases, modules are complemented in order to summarize the possible behaviour of the environment. A particular technique for complementing modules (for the case of the SAM view) was introduced in the previous Section, that has been used in [11] for exact analysis or in [28] for approximate analysis. Other examples where components are complemented modules appeared in [13] for the improvement of bounds or in [26] for approximate analysis.

In divide and conquer strategies there always exists an integration of partial solutions. Therefore, a second criterium for classification may be *the existence or not of an explicit global abstract view of the system* usable in the integration phase. In our case, the existence of a net model providing a global abstract view makes the difference. Examples of techniques in which no high level view of the system is used are: the bounds presented in Section 20.1.2 [7] and their improvement presented in [13], or the approximate technique proposed in [26]. On the other hand, two-level approaches (i.e., techniques that use both low and high level views of the system) for the analysis of SPN's are, for instance, the flow equivalent aggregation for the approximate analysis of SPN's [20], the product-form approximation techniques [1] presented in Chapter 22, the exact analysis of SAM models [11] (Chapter 12), and the approximate analysis of DSSP (Deterministically Synchronized Sequential Processes) [29, 30] or SAM models [28].

Since the structure of the SPN model is defined by a bipartite graph, an additional classification criterium could be *the selection of either places or transitions for the partition of the model into modules*. However, as we stressed in Section 20.1.3, both approaches can be translated one into the other, thus the selection of places or transitions for defining the cut plays a non-substantial (essentially syntactical) rule in the decomposition process.

In addition to the above criteria, all the analysis techniques for performance evaluation of models (and in particular those based on a net-driven decomposition) can be classified according to *the quality of results* into: exact, approximate and bounding techniques. The general cost/accuracy trade-off must be taken into account in the selection of the desired approach.

20.2.2 Isolated Modules and Single-Level Techniques

An example of solution method within a divide and conquer strategy with isolated modules and a single level of abstraction is the technique for computing performance bounds using P-semiflows presented in Section 20.1.2 [7], where thanks to the LPP expression the decomposition is implicit, transparent to the analyzer.

Another example that makes use of isolated modules and a single level of abstraction is the exact analysis of SGSPN (Superposed Generalized Stochastic Petri Nets) [19] presented in Chapter 12. Here the decomposition must be made explicit. The basic idea behind the SGSPN technique can be explained using the model of Figure 20.2.a, that shows a GSPN $\mathcal{S}$ that can be considered as the composition of two GSPN's $\mathcal{S}_1$ and $\mathcal{S}_2$ (depicted in Figure 20.2.b) over two common transitions a and b. $\mathcal{S}_1$ has therefore a number of states equal to the number of ways in which 2 balls can be distributed into four boxes: 10 states. Analogously for $\mathcal{S}_2$ (4 boxes, 1 ball): 4 states. A product state space PS can then be defined as

$$\mathrm{PS} = \mathrm{RS}_1 \times \mathrm{RS}_2$$

and it is straightforward to observe that $\mathrm{RS} \subseteq \mathrm{PS}$. Note that PS has a number of states equal to the product of the above numbers (40 states), but the reachability set of $\mathcal{S}$ has only 14 states.

According to the techniques presented in [19] the following matrix $\mathbf{G}$ of size $|\mathrm{PS}| \times |\mathrm{PS}|$ can be constructed:

$$\mathbf{G} = \mathbf{Q}'_1 \oplus \mathbf{Q}'_2 \quad \begin{array}{l} - \sum_{t\in\{\mathsf{a},\mathsf{b}\}} w(t)[\mathbf{K}_1(t) \otimes \mathbf{K}_2(t)] \\ + \sum_{t\in\{\mathsf{a},\mathsf{b}\}} w(t)[\mathbf{K}'_1(t) \otimes \mathbf{K}'_2(t)] \end{array} \tag{20.8}$$

where the $\mathbf{Q}'_i$, $\mathbf{K}_i(t)$, and $\mathbf{K}'_i(t)$ (for $i \in \{1,2\}$) are $|\mathrm{RS}_i| \times |\mathrm{RS}_i|$ matrices, that can all be derived from the infinitesimal generator $\mathbf{Q}_i$ of $\mathcal{S}_i$.

The idea behind this formula is to split the behaviour of each component into *local behaviour* (related to transitions local to a single component), and *dependent behaviour* (related to “synchronizing transitions” a and b). The local behaviour of each GSPN is represented by $\mathbf{Q}'_i$, and since the local behaviour is independent, the global behaviour due to local transitions can be obtained as the tensor sum of the $\mathbf{Q}'_i$ matrices. The behaviour related to synchronization requires that, for a synchronization transition to fire, both $\mathcal{S}_i$ must be in a state that enables the transition. $\mathbf{K}_i(t)$, the correcting matrix for transition t, has a 1 in each entry of the matrix that corresponds to a change of state due to t in $\mathcal{S}_i$. The tensor product will indeed realize the required condition that *a synchronization transition fires only in global states whose corresponding local states in the $\mathcal{S}_i$ enable t.* The term with the $\mathbf{K}'_i(t)$ matrices is used to compute the portion of the diagonal elements expression that accounts for synchronization transitions.

By definition of the tensor sum and product, $\mathbf{G}$ is a $|\mathrm{PS}| \times |\mathrm{PS}|$ matrix, and it is shown in [19] how the non null entries of the vector $\boldsymbol{\pi}$, solution of the equation $\boldsymbol{\pi} \cdot \mathbf{G} = \mathbf{0}$, are the steady-state solution of $\mathcal{S}$. Moreover a solution process may be devised that does not require the explicit computation and storing of $\mathbf{G}$, so that the biggest memory requirement is that of the vector $\boldsymbol{\pi}$. The technique is extended to transient analysis in [24]. The computational cost, under full matrix implementation assumption, is smaller than the classical vector to matrix multiplication [31], while recent results have shown [5] that the viceversa is true for matrices with a mean number of elements per rows less than $K^{\frac{1}{K-1}}$ (K being the number of components) under sparse matrix implementation.

The above technique produces a significant storage saving whenever the size of PS, and therefore that of $\boldsymbol{\pi}$, is inferior to the number of non null elements of $\mathbf{Q}$, since, otherwise, it would be better to store $\mathbf{Q}$ explicitly.

The solution procedure outlined before is the basic idea behind a number of works that have appeared in the literature. The work in [31] defines the basics of the method and applies it to networks of stochastic automata, while classes of SPN's for which a single level structured solution has been applied are Superposed Stochastic Automata [18] and Superposed GSPN (SGSPN) [19] (SGSPN nets that can be interpreted as the superposition over a subset of timed transitions of equal label and rates of a set of GSPN's).

The distance between RS and PS can limit the applicability of the technique, but if a bit vector of size |PS| can be allocated in memory, then a state space exploration can be performed [23], which, with an additional tree-like data structure of size $O(|\mathrm{RS}| \cdot \log |\mathrm{RS}_i|)$ (i is the index of the component whose state space is represented in the tree leaves), allows the definition of multiplication algorithms that consider only reachable states. The computational overhead is at most logarithmic in |RS| [5].

20.2.3 Complemented Modules and Single-Level Techniques

As we remarked in Section 20.1.4, the main idea behind the complementation of modules is to obtain components with a certain abstract view of its environment, the rest of the system. The goal is to find a process that leads to better (or just possible) quantitative evaluation techniques using the composition of partial results computed for the components.

A simple example of complementation of modules was proposed in [13] for the improvement of the linear programming based bounds presented in Section 20.1.2. In that section, bounds for the mean interfiring time of transitions of the modules in complete isolation is computed. A more realistic computation of the mean interfiring time can be considered using the concept of *liveness bound* of transitions. The liveness bound of a transition is the *maximum reentrance* (or maximum self-concurrency) that the net structure and the marking allow for the transition in steady-state. In other words, it gives the number of servers for transition t in steady state. Formally, $L(t) = \max\{k \mid \forall \mathbf{m}' : \mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}', \exists \mathbf{m} : \mathbf{m}' \xrightarrow{\sigma'} \mathbf{m} \wedge \mathbf{m} \geq k\ \mathbf{Pre}[t]\}$. In the case that the above quantity cannot be computed efficiently, an structural counterpart (that can always be computed in an efficient manner) can be considered: the *structural enabling bound* [13].

The technique we are going to briefly present by means of an example is based on the consideration of *embedded product-form closed monoclass queueing networks* of the SPN. From a topological point of view, these embedded networks are *P-components*: strongly connected state machines generated by P-semiflows. An improvement of the lower bound for the mean interfiring time

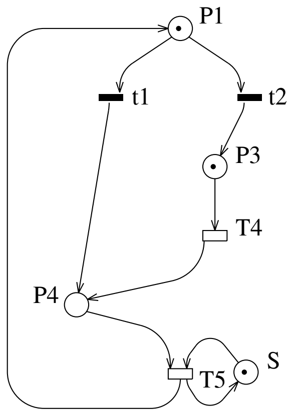


Figure 20.8: Complementation of the module in Figure 20.1.b.

of a transition t_i given by LPP (20.5) can be eventually obtained computing the exact mean interfiring time of that transition in the P-component generated by a minimal P-semiflow $\mathbf{y}$, with $L(t)$–server semantics for each involved transition t (in fact, it is not necessary that t_i belongs to the P-component; the bound for other transition can be computed and then weighted according to the visit ratios in order to compute a bound for t_i). The P-semiflow $\mathbf{y}$ can be selected among the optimal solutions of (20.5) or it can be just a feasible *near-optimal* solution.

Let us consider once again the net system depicted in Figure 20.1.a. The bound computed in (20.7) does not take into account the queueing time at places P4 and P5 due to synchronization (T5). That bound can be improved if the P-component generated by $\mathbf{y_1} = (1, 0, 1, 1, 0)$ (depicted in Figure 20.1.b) is considered with liveness bound of transition T5 reduced to 1 (which is the liveness bound of this transition in the whole net). That information may be introduced in the module of Figure 20.1.b by the addition of a place S that limits the number of servers of T5 using the information of the rest of the system, leading to the component shown in Figure 20.8. Notice that if place S were added in the original system it would be implicit. The system in Figure 20.8 is isomorphous to a product-form queueing network and it can be efficiently solved. The average interfiring time of T5 in this system gives the following improvement of the bound (20.7) for the average interfiring time of the same transition in the original system (Figure 20.1.a):

$$\Gamma[\mathsf{T5}] \geq 4.562502 \tag{20.9}$$

The exact average interfiring time of T5 is 4.978493, therefore by adding S and considering the exact performance of the component in isolation the relative error has been reduced from 19.65% to 8.36%.

Another interesting example of complemented modules using information of the rest of the system may be described by the tensor algebra approach presented in the previous section for the exact solution of SGSPN models. If the components depicted in Figures 20.4.b and 20.4.c are used in equation (20.8) instead of modules depicted in Figure 20.2.b, the size of matrices $\mathbf{Q}'_1$, $\mathbf{K}_1(t)$, and $\mathbf{K}'_1(t)$ is reduced from 10 to 7, therefore, the difference between PS and RS is reduced. Notice that places q34, p12, and p34 are implicit what means, in particular, that the environment of $\mathcal{S}_2$ (subsystem $\mathcal{S}_1$) does not constraint its behaviour. If the other decomposition depicted in Figure 20.2.c is considered, the tensor algebra approach cannot be directly applied since the state spaces of the modules (thus the size of matrices involved in the tensor expression) are unbounded. Using implicit places to complement the modules, as explained in Section 20.1.4 and depicted in Figures 20.5.b and 20.5.c, the state spaces of the components are limited (finite) to 8 and 5, respectively, and the technique can be applied (with $|\text{PS}| = 40$, while $\text{RS} = 14$).

20.2.4 Isolated Modules and Two-Level Techniques

A classical technique for the analysis of queueing network models that can be also applied to SPN's is *flow equivalent aggregation* (FEA) [14, 25, 20, 22]. The basic approach is to replace a general server or subnetwork of queues by a Flow Equivalent Service Center (FESC). An arriving customer sees the FESC as a black box whose behaviour is completely characterized by a listing of the residence time (the inverse of the throughput) as a function of the possible customer population. In order to determine the state dependent service rates, the subsystem is studied off-line, i.e., without any interaction with the environment. In general, a FESC matches only the first moment of the probability distribution, as the higher moments are too cumbersome to obtain.

Two steps for constructing an FESC can be determined, once a *subsystem* to be analysed by FESC is defined:

1. Analyse the (low level) subsystem by maintaining the number of customers constant. This is done by shorting out all other parts of the network and varying the number of customers up to the maximum allowed in the subsystem.

2. The Aggregated Network (AN) or high level system is constructed as a server with a queue-dependent service rate. Let $\chi_{\text{SW}}(k)$ be the conditional throughput of the subnetwork (SW) when k customers are present, and $\mu_{\text{AN}}(k)$ the (state dependent) service rate of AN. The approximation is done through the following equality, under the assumption of exponential service time: $\chi_{\text{SW}}(k) = \mu_{\text{AN}}(k)$.

As an example, consider the SPN depicted in Figure 20.9. Service times of transitions are the following (single server semantics is assumed, and race policy at conflicts): $\mathsf{s}[\mathsf{Ti}] = 1.0$ for $i \neq 6, 7, 13$; $\mathsf{s}[\mathsf{T6}] = \mathsf{s}[\mathsf{T7}] = 2.0$; $\mathsf{s}[\mathsf{T13}] = 0.5$. Once the subsystem to be aggregated has been selected, the systems depicted in Figure 20.10.a and 20.10.b are derived. The first of them is the isolated subsystem or low level subsystem while the second one is the aggregated network or high level

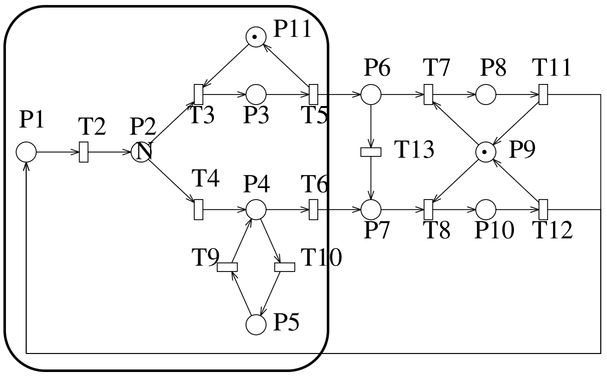
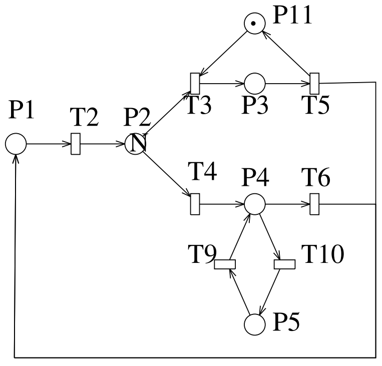
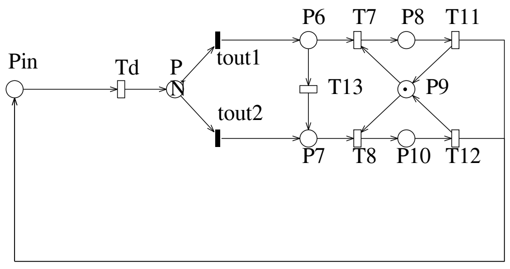


Figure 20.9: An example of stochastic Petri net system with a subsystem to be aggregated.



(a)



(b)

Figure 20.10: (a) Isolated subsystem and (b) aggregated system for the model in Figure 20.9.

N	r_5	r_6	$\chi(\mathsf{T2})$
1	0.500	0.500	0.400
2	0.431	0.569	0.640
3	0.403	0.597	0.780
4	0.389	0.611	0.863
5	0.382	0.618	0.914

Table 20.1: Solution of the isolated subsystem in Figure 20.10.a for different values of N.

N	# states		throughput		% error
	aggreg.	orig.	aggreg.	orig.	
1	5	9	0.232	0.232	0.00
2	12	41	0.381	0.384	0.78
3	22	131	0.470	0.474	0.84
4	35	336	0.521	0.523	0.38
5	51	742	0.548	0.547	< 0.10

Table 20.2: Solution of the aggregated system in Figure 20.10.b and of the original system (Figure 20.9) for different values of N.

system. The subsystem is evaluated in isolation with constant number of tokens varying from 1 to N (Figure 20.10.a). The outcome of the analysis is: (1) the relative throughput (visit ratios) between T5 and T6, and (2) the throughput of the subsystem (more precisely, the throughput of T2), for different values of the initial marking of P2. The obtained results are shown in Table 20.1.

When the subsystem is aggregated (Figure 20.10.b), routing at tout1 and tout2, and delay at Td are state dependent (they depend on $\mathbf{m}[\mathsf{Pin}]$). Table 20.2 shows the results obtained with flow equivalent aggregation for the system in Figure 20.9 for different values of the initial marking at P1 ($N = 1, \ldots, 5$). The exact results obtained by solving the CTMC of the original system and the relative error of the FEA are also included (together with the size of the state spaces of the original system and of the aggregated system).

In FEA, the behaviour of the subnet is assumed to be *independent* of the arrival process and to depend only on the number of customers in the system, i.e., the behaviour is completely independent of the environment. This condition is frequently violated. In [21], it is shown that even in very simple cases the mean completion (or traversing) time of a subnet (i.e., the average time spent by a token traversing the subnet) may depend on the token's interarrival process. This fact happens, for instance, if there exist internal loops in the subnet or if there are trapped tokens in a fork-join inside the subnet; when such situations arise, it becomes necessary to implement a less efficient (usually involving a fixed-point search iterative process) but more accurate approximation technique, like, for instance, response time approximation [22, 21, 8, 26, 29, 30, 28].

There are some cases where FEA leads to exact results. This happens for the particular case of product-form queueing networks (Norton's theorem) [14], where the mean completion time of the subnet is strictly independent of the token's interarrival process.

20.2.5 Complemented Modules and Two-Level Techniques

In Section 20.2.3 we have illustrated how the use of complemented modules can improve the results obtained using isolated modules. In the particular case of the tensor algebra approach for the exact solution of models, we have seen that the distance between RS and PS can be reduced, including in each module some information (abstract view) of the other. It may be possible that PS includes a number of *spurious states* that are non-reachable in the original system (i.e., they do not belong to RS). To limit the number of spurious states, an abstract description of the system can be used to appropriately pre-select the subsets of the states that should enter in the Cartesian product.

For instance, for both decompositions of Figure 20.2.a (the one in Figures 20.4.b and 20.4.c, and that in Figures 20.5.b and 20.5.c), the basic skeleton $\mathcal{BS}$ (or high level view of the model) is just the cycle a-i12-b-i34 with one token initially in i12. The states of the low level systems $\mathcal{LS}_1$ and $\mathcal{LS}_2$ can be partitioned according to the states of $\mathcal{BS}$: those states with equal high level representation $\mathbf{z}$ (i.e., those whose projection on i12 and i34 is the same) belong to a single class $\mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_1)$ ($\mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_2)$) in the state space of $\mathcal{S}_1$ (respectively, $\mathcal{S}_2$). The restricted product state space RPS can then be built as:

$$\mathrm{RPS}(\mathcal{S}) = \biguplus_{\mathbf{z}\in\mathrm{RS}(\mathcal{BS})} \mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_1) \times \mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_2)$$

where $\biguplus$ is the *disjoint* set union. Note that for the example in Figure 20.4 we obtain a precise characterization of the state space (i.e., there exist no spurious states; a property that holds in general for live and bounded marked graphs), but the union of Cartesian products can in general produce a superset of the reachable state space, depending on how precise is the abstract representation.

Since the state space is no longer the Cartesian product of sets of local state spaces, but the union of Cartesian products, we cannot expect to have for the infinitesimal generator a tensor expression as simple as before: we can build a matrix $\mathbf{G}$, of size $|\mathrm{RPS}| \times |\mathrm{RPS}|$, and then consider it as block structured according to the states of $\mathcal{BS}$. Each block will thus refer to a set of states obtained by a Cartesian product, and a tensor expression for each block can be derived. Diagonal blocks $\mathbf{G}(\mathbf{z},\mathbf{z})$ can be expressed as

$$\mathbf{G}(\mathbf{z},\mathbf{z}) = \mathbf{Q}_1(\mathbf{z},\mathbf{z}) \oplus \mathbf{Q}_2(\mathbf{z},\mathbf{z})$$

where the $\mathbf{Q}_i(\mathbf{z},\mathbf{z})$ are the submatrices of the infinitesimal generator of $\mathcal{LS}_i$ determined by the states whose abstract representation is $\mathbf{z}$. Each $\mathbf{G}(\mathbf{z},\mathbf{z}')$ block, with $\mathbf{z} \neq \mathbf{z}'$, describes instead the behaviour that changes the high level

	$\mathbf{m}$(c1)=1	$\mathbf{m}$(c1) = 2	$\mathbf{m}$(c1) = 3
$\mathcal{BS}$	10	46	146
$\mathcal{LS}_1$	199	6.985	108.945
$\mathcal{LS}_2$	199	6.985	108.945
$\mathcal{LS}_3$	22	190	1.032
RPS	8.716	3.872.341	431.270.717
RS	8.716	3.872.341	no memory
PS(case A)	11.426.400	no memory	no memory
PS(case B)	—	198.994.880	no memory

Table 20.3: SAM technique: State spaces computed for the model of Figure 20.3.a.

state; each block $\mathbf{Q}(\mathbf{z},\mathbf{z}')$ can be written as

$$\mathbf{G}(\mathbf{z},\mathbf{z}') = \sum_{t:\mathbf{z}\xrightarrow{t}\mathbf{z}'} w(t)[\mathbf{K}'_1(t)(\mathbf{z},\mathbf{z}') \otimes \mathbf{K}'_2(t)(\mathbf{z},\mathbf{z}')]$$

where the $\mathbf{K}'_i(t)(\mathbf{z},\mathbf{z}')$ are the submatrices of the infinitesimal generator of $\mathcal{LS}_i$ whose rows (columns) are abstractly represented by $\mathbf{z}$ ($\mathbf{z}'$), projected to include only the contribution due to t. Observe that, in our example, there is a single t for each given pair $(\mathbf{z},\mathbf{z}')$.

The above approach that has been applied here to SGSPN, was developed and used in the literature in the context of the solution of "asynchronous systems", SPN that can be seen as modules that interact by exchanging tokens [4, 2, 6, 10, 11]. For the example in Figure 20.3.a, considering the SAM view depicted in Figure 20.7, Table 20.3 shows the sizes of the state space of $\mathcal{LS}_i$ and $\mathcal{BS}$ as well as the size of RPS and RS of the complete system, for an initial marking with three tokens in a1 and d1, and corresponding implicit places, and of $1, 2$, and 3 tokens in place c1.

A straightforward comparison with single-level synchronous decomposition may not be very significant, since the division of the SAM into modules may not be the best one for SGSPN. We have tried two different synchronous decompositions: three components identified by places starting with a or b for the first component, c for the second, and d for the third (case **A**) (derived from modules in Figure 20.3.c by adding some appropriated places [11]), and two components identified by places starting with a or b or d for the first component, and c for the second (case **B**).

The size of the product state space is also shown in Table 20.3 (where — means that the experiment was not performed). For the **A** case it was not possible to increase to 2 the number of tokens in c1, since after the generation of the state spaces of the three components, of size $1.701, 5.161$, and 5.161, the tool stops, which is not surprising considering that, according to the size of the components, PS has a size of about $45 \ * 10^9$.

The decomposition of case **B** is indeed more favourable, since we were able to solve the $\mathbf{m}(\texttt{c1}) = 2$ model.

In summary, if we have a construction rule for RS as a disjoint union of Cartesian products, we can limit the problem of the difference $|\mathrm{PS}| - |\mathrm{RS}|$, while still maintaining the benefits of the solution in structured form as in the SGSPN case: indeed it is only necessary to organize the classical vector by matrix multiplication $\boldsymbol{\pi} \cdot \mathbf{G}$ in terms of subvector by submatrix multiplication. Details for the computational costs of this technique under sparse and full matrix storage scheme have been reported in [3].

20.3 Concluding remarks

Divide and conquer is a classical and sometimes successful enough strategy to fight against computational complexity. Using stochastic PN models of concurrent systems we consider several model decomposition and subsequent solution techniques in order to compute performance figures.

Modules, implicit places and components are the basic elements for providing decomposed views of systems. Criteria like the addition of information summarizing the behaviour of the environment of a module or the eventual existence of a global abstract view allows to situate many sparse techniques in a certain conceptual framework. The introduced possibility for going to a two-level view of systems can be generalized by recursively applying the principles, leading to multiple-level views; these have been considered, for example, for approximation techniques [14, 21].

Finally, just to point out that a different use of structure theory (also using implicit places) in order to improve performance bounds is to add to the original net model some implicit places (in relation with some *traps* structures) in such a way that new components are found [9, 12]. If the new components are bottlenecks, the computed bound is improved. As an example, adding a place π to the net in Figure 20.1.a with input transitions $\mathsf{T3}$ and $\mathsf{T4}$ and output transition $\mathsf{T5}$ (with ordinary arcs) and initially marked with zero tokens, an additional minimal P-semiflow is generated (with support $\{\mathsf{P1}, \mathsf{P2}, \mathsf{P3}, \pi\}$), and the application of (20.6) gives now $\Gamma[\mathsf{T5}] \geq \max\{(5+3)/2, 0.5+3, (0.5+5+3)/2\} = 4.25$ thus improving the first bound ($\Gamma[\mathsf{T5}] \geq 4$) given in (20.7). Moreover, if the new component generated by this additional P-semiflow is considered using the technique presented in Section 20.2.3, limiting the liveness bound of transition $\mathsf{T5}$ to 1 (as it was performed in Figure 20.8), the bound is improved to $\Gamma[\mathsf{T5}] \geq 4.779406$ (instead of the value $\Gamma[\mathsf{T5}] \geq 4.562502$ obtained without the addition of the implicit place π, thus the relative error is reduced from 8.36% to 4%). Better quality results are obtained increasing the computational investments.

Bibliography

[1] B. Baynat and Y. Dallery. A product-form approximation method for general closed queueing networks with several classes of customers. *Performance Evaluation*, 24:165–188, 1996.

[2] P. Buchholz. A hierarchical view of GCSPN's and its impact on qualitative and quantitative analysis. *Journal of Parallel and Distributed Computing*, 15(3):207–224, July 1992.

[3] P. Buchholz. Numerical solution methods based on structured descriptions of Markovian models. In G. Balbo and G. Serazzi, editors, *Computer Performance Evaluation. Modeling Techniques and Tools*, pages 251–267. Elsevier, 1992.

[4] P. Buchholz. A class of hierarchical queueing networks and their analysis. *Queueing Systems*, 15:59–80, 1994.

[5] P. Buchholz, G. Ciardo, S Donatelli, and P. Kemper. Complexity of Kronecker operations on sparse matrices with applications to the solution of Markov models. Icase report 97-66, Institute for Computer Applications in Science and Engeneering, Hampton, VA, 1997.

[6] P. Buchholz and P. Kemper. Numerical analyisis of stochastic marked graphs. In *Proc.* 6^{th} *Intern. Workshop on Petri Nets and Performance Models*, pages 32–41, Durham, NC, USA, October 1995. IEEE-CS Press.

[7] J. Campos, G. Chiola, and M. Silva. Properties and performance bounds for closed free choice synchronized monoclass queueing networks. *IEEE Transactions on Automatic Control*, 36(12):1368–1382, December 1991.

[8] J. Campos, J. M. Colom, H. Jungnitz, and M. Silva. Approximate throughput computation of stochastic marked graphs. *IEEE Transactions on Software Engineering*, 20(7):526–535, July 1994.

[9] J. Campos, J. M. Colom, and M. Silva. Improving throughput upper bounds for net based models of manufacturing systems. In J. C. Gentina and S. G. Tzafestas, editors, *Robotics and Flexible Manufacturing Systems*, pages 281–294. Elsevier Science Publishers B.V. (North-Holland), Amsterdam, The Netherlands, 1992.

[10] J. Campos, S. Donatelli, and M. Silva. Structured solution of stochastic DSSP systems. In *Proceedings of the 7th International Workshop on Petri Nets and Performance Models*, pages 91–100, Saint Malo, France, June 1997. IEEE Computer Society Press.

[11] J. Campos, S. Donatelli, and M. Silva. Structured solution of asynchronously communicating stochastic modules. Research Report RR-98-4, Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, María de Luna 3, 50015 Zaragoza (Spain), April 1998. Submitted paper.

[12] J. Campos and M. Silva. Structural techniques and performance bounds of stochastic Petri net models. In G. Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 352–391. Springer-Verlag, Berlin, 1992.

[13] J. Campos and M. Silva. Embedded product-form queueing networks and the improvement of performance bounds for Petri net systems. *Performance Evaluation*, 18(1):3–19, July 1993.

[14] K. M. Chandy, U. Herzog, and L. Woo. Parametric analysis of queueing networks. *IBM Journal of Res. Develop.*, 19:36–42, January 1975.

[15] J. M. Colom. *Análisis Estructural de Redes de Petri, Programación Lineal y Geometría Convexa.* PhD thesis, Departamento de Ingeniería Eléctrica e Informática, Universidad de Zaragoza, Spain, June 1989. Research Report. GISI-RR-89-11. In Spanish.

[16] J. M. Colom and M. Silva. Convex geometry and semiflows in P/T nets. A comparative study of algorithms for computation of minimal p-semiflows. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 79–112. Springer-Verlag, Berlin, 1991.

[17] J. M. Colom and M. Silva. Improving the linearly based characterization of P/T nets. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 113–145. Springer-Verlag, Berlin, 1991.

[18] S. Donatelli. Superposed stochastic automata: A class of stochastic Petri nets with parallel solution and distributed state space. *Performance Evaluation*, 18:21–36, 1993.

[19] S. Donatelli. Superposed generalized stochastic Petri nets: Definition and efficient solution. In R. Valette, editor, *Application and Theory of Petri Nets 1994*, volume 815 of *Lecture Notes in Computer Science*, pages 258–277. Springer-Verlag, Berlin, 1994.

[20] H. Jungnitz and A. A. Desrochers. Flow equivalent nets for the performance analysis of flexible manufacturing systems. In *Proceedings of the*

1991 IEEE International Conference on Robotics and Automation, pages 122–127, Sacramento, CA, USA, April 1991.

[21] H. Jungnitz, B. Sánchez, and M. Silva. Approximate throughput computation of stochastic marked graphs. *Journal of Parallel and Distributed Computing*, 15:282–295, 1992.

[22] H. J. Jungnitz. *Approximation Methods for Stochastic Petri Nets.* PhD thesis, Dept. of Electrical, Computer and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY, USA, May 1992.

[23] P. Kemper. Numerical analyisis of superposed GSPN. *IEEE Transactions on Software Engineering*, 22(4):615–628, September 1996.

[24] P. Kemper. Transient analysis of superposed GSPNs. In *7-th International Conference on Petri Nets and Performance Models - PNPM97*, pages 101–110. IEEE Computer Society, 1997.

[25] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Prentice-Hall, Englewood Cliffs, NJ, 1984.

[26] Y. Li and C. M. Woodside. Complete decomposition of stochastic Petri nets representing generalized service networks. *IEEE Transactions on Computers*, 44(8):1031–1046, August 1995.

[27] G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors. *Optimization*, volume 1 of *Handbooks in Operations Research and Management Science.* North-Holland, Amsterdam, 1989.

[28] C. J. Pérez-Jiménez and J. Campos. A response time approximation technique for stochastic general P/T systems. In *Proceedings of the 2nd IMACS International Multiconference on Computational Engineering in Systems Applications (CESA'98)*, Hammamet, Tunisia, April 1998. IEEE Systems, Man and Cybernetics.

[29] C. J. Pérez-Jiménez, J. Campos, and M. Silva. Approximate throughput computation of a class of cooperating sequential processes. In *Proceedings of the Rensselaer's Fifth International Conference on Computer Integrated Manufacturing and Automation Technology (CIMAT'96)*, pages 382–389, Grenoble, France, May 1996.

[30] C. J. Pérez-Jiménez, J. Campos, and M. Silva. On approximate performance evaluation of manufacturing systems modelled with weighted T-systems. In *Proceedings of the IMACS/IEEE-SMC Multiconference on Computational Engineering in Systems Applications (CESA'96)*, pages 201–207, Lille, France, July 1996.

[31] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *Proceedings of the 1985 SIGMETRICS Conference*, pages 147–154, Austin, TX, USA, August 1985. ACM.

[32] M. Silva. *Introducing Petri Nets*, chapter 1. Chapman & Hall, London, 1993.