Minimal Representation of Uniform Recurrence Equations

Bruno Gaujal, Alain Jean-Marie, Jean Mairesse

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Minimal Representation of Uniform Recurrence Equations

Bruno Gaujal, Alain Jean-Marie and Jean Mairesse

N° 2568

June 1995

PROGRAMME 1

ISSN 0249-6399

Minimal Representation of Uniform Recurrence Equations

Bruno Gaujal, Alain Jean-Marie and Jean Mairesse*

Programme 1 — Architectures parallèles, bases de données, réseaux
et systèmes distribués
Projet Mistral

Rapport de recherche n° 2568 — June 1995 — 51 pages

Abstract: We consider a system of uniform recurrence equations of dimension one. We show how the computation can be carried using minimal memory size with several synchronous processors. This result has applications in register minimization for digital circuitry and parallel simulation of timed marked graphs.

Key-words: uniform recurrence equations, register minimization, Strahler's number, circuit design, (max,+) linear systems.

(Résumé : tsvp)

Supported by the European Grant BRA-QMIPS of CEC DG XIII.

*Research supported by the Direction des Recherches Etudes et Techniques (DRET) under contract n° 91 815.

Unité de recherche INRIA Sophia-Antipolis
2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex (France)
Téléphone : (33) 93 65 77 77 – Télécopie : (33) 93 65 77 65

Représentation Minimale des Systèmes d'Équations Récurrentes Uniformes

Résumé : On considère un système d'équations récurrentes uniformes de dimension un. On montre comment mener le calcul de façon synchrone en utilisant un nombre de cases mémoire minimal. Ce résultat a des applications pour la minimisation du nombre de registres dans les circuits digitaux ainsi que pour la simulation parallèle des graphes d'événements temporisés.

Mots-clé : équations récurrentes uniformes, minimisation de registres, nombre de Strahler, conception de circuits, systèmes (max,+) linéaires

Contents

1 Introduction

The model under study will be the Uniform Recurrence Equations (URE) [12]. We consider E-valued variables $X_i(n), i \in V = \{1, \ldots, k\}, n \in K$, where E is an arbitrary space and $K \subset \mathbb{Z}^p$ for some $p \in \mathbb{N}$. These variables satisfy the equations

$$X_i(n) = F_i(X_j(n - \gamma)), (j, \gamma) \in \mathcal{E}_i, \forall n \in K \; . \tag{1}$$

The set $\mathcal{E}_i$ is a finite non empty subset of $\{1, \ldots, k\} \times \mathbb{Z}^p$.

There are various motivations to study URE. They appear in the description of partial differential equations using finite difference methods or in the study of discrete event systems. The case $p > 1, K = \mathbb{Z}^p$ has often been studied in the literature, see [12]. In such a case, some of the major issues are the constructivity [12] and loop parallelization [6]. These problems and others appearing in this framework will be discussed in §3.1.

In this paper, we consider only the simple case where $K = \mathbb{Z}$ (systems of dimension one). We investigate the problem of minimizing the number of "memory locations": we want to determine the minimal memory size that is needed to compute all the variables $X_i(n)$ of Equation (1) using parallel processors with a shared memory.

We show that the solution of this problem has many applications. It can be used in order to obtain the most efficient representation of the system for simulation purposes. This aspect of the problem will be investigated in §7. In a quite different context, URE appear in the modeling of logical circuits, systolic arrays or program loops. The minimization for URE enables us to obtain an optimal design of such circuits (in terms of number of registers). This application will be discussed in §6.

The paper is organized as follows. In Section 2, we precise the definition of a system of URE and we present two associated graphs, the *dependence graph* and the *reduced graph*. In Section 3 we describe the problem that we are going to address. Section 4 investigates the relations that can be found between *cuts* in the dependence graph and the memory size required for an execution of the URE; section 5 presents the interpretation of the above quantities in the reduced graph. Finally in Sections 6

and 7, two applications are described, for digital circuits and (max,+) linear systems respectively.

2 Basic Models

Definition 2.1. *A system of Uniform Recurrence Equations (URE) on the set of variables* $X_i(n), i \in V = \{1, \ldots, k\}, n \in \mathbb{Z}$ *is a system of the form*

$$X_i(n) = F_i(X_j(n-\gamma), (j,\gamma) \in \mathcal{E}_i,\ n \in \mathbb{Z}\ ), \tag{2}$$

where the set $\mathcal{E}_i$ *is a finite non-empty subset of* $\{1, \ldots, k\} \times \mathbb{N}$.

The integers γ are called the *delays*. The system $\mathcal{S}$ defined by Equation (2) is said to be uniform because the dependence sets $\mathcal{E}_i$ do not depend on n. Note that it is possible to have two delays $\gamma, \gamma' \in \mathbb{N}, \gamma \neq \gamma'$ such that $(j,\gamma) \in \mathcal{E}_i$ and $(j,\gamma') \in \mathcal{E}_i$.

There is no restriction on the generality of the functions F_i considered.

A system of URE $\mathcal{S}$ is *constructive* if given the values of the "negative" variables $X_i(n), n \leqslant 0$, involved in $\mathcal{S}$ (*initial data*), there exists an ordering of the equations such that, $\forall i, \forall n > 0$, all the variables present in the right hand side of the equation defining $X_i(n)$ are either "negative" variables or can be computed before $X_i(n)$. This condition is satisfied if and only if there is no cycle in the dependences, i.e. there does not exist $i_1, \ldots, i_p, i_{p+1} = i_1$ such that $(i_2, 0) \in \mathcal{E}_{i_1}, (i_3, 0) \in \mathcal{E}_{i_2}, \ldots, (i_1, 0) \in \mathcal{E}_{i_p}$.

Remark 2.2. We could have considered an apparently more general definition of URE allowing the delays γ to be negative. In this case, the constructivity assumption becomes
For each cycle $(i_1, \gamma_1), \ldots, (i_p, \gamma_p), i_{p+1} = i_1$ such that $(i_{j+1}, \gamma_{j+1}) \in \mathcal{E}_{i_j}, j \in \{1, \ldots, p\}$ then $\sum_{j=1}^{p} \gamma_j > 0$. Under the constructivity assumption, it is possible to come back to Definition 2.1 through a simple renumbering of the variables (i.e. $X_i(n) := X_i(n + c_i)$ for some constant $c_i \in \mathbb{Z}$ independent of n).

From now on, the system $\mathcal{S}$ that we consider is always assumed to be constructive. We are going to present two equivalent ways to describe S : the dependence graph and the reduced graph.

Example 2.3. *The illustrative examples to be presented in the following correspond to the system :*

$$\left\{ \begin{array}{lcl} X_1(n) & = & F_1(X_3(n-1)) \\ X_2(n) & = & F_2(X_1(n-2)) \\ X_3(n) & = & F_3(X_2(n), X_4(n-2)) \\ X_4(n) & = & F_4(X_3(n-1), X_4(n-1)), \end{array} \right. \tag{3}$$

2.1 Dependence graph

We introduce the graph $\mathcal{D}$ of the dependences between the variables $X_i(n)$.

Definition 2.4 (dependence graph). *The dependence graph associated with a system of URE is the graph $\mathcal{D}$ with $(V \times \mathbb{Z})$ as the set of nodes. There is an arc from the node (i,n) to the node (j,m) if $X_j(m) = F_j(X_i(n), \dots)$ or equivalently if $(i, m-n) \in \mathcal{E}_j$ (notation: $(i,n) \to (j,m)$).*

The n-th *level* in $\mathcal{D}$ is the set of nodes $\{(i,n), 1 \leqslant i \leqslant k\}$. The i-th *column* in $\mathcal{D}$ is the set of nodes $\{(i,n), n \in \mathbb{Z}\}$. In the following, we will refer to nodes (i,n), $n \leqslant 0$ as *negative* nodes and nodes (i,n), $n > 0$ as *positive* nodes.

It is immediate from the definition of an URE that $\mathcal{D}$ is 1-periodic, i.e.

$$(i,n) \to (j,m) \Longrightarrow (i,n+1) \to (j,m+1) \; .$$

Note also that the constructivity assumption implies that the graph $\mathcal{D}$ is acyclic.

We have represented in Figure 1 the dependence graph corresponding to the system of Example 2.3.

The dependence graph appears under various forms and names in the literature. For example, we can mention the following names: developed graph, task graph, PERT graph, unfolded process graph or activity network.

2.2 Reduced graph

Since the dependence graph is 1-periodic, it can be folded into a more compact form. This is how we construct the *reduced graph* $\mathcal{R}$ associated with the system S.

Definition 2.5 (Reduced graph).
The reduced graph is an arc valued graph $\mathcal{R} = (V, E, \Gamma)$. The set of nodes is $V = \{1, \cdots, k\}$. There is an oriented arc in E from i to j if

$$\exists \gamma \in \mathbb{N} \;\; s.t. \;\; (i,\gamma) \in \mathcal{E}_j \; . \tag{4}$$

This arc is valued with the delay γ. If there exist several delays γ verifying condition (4), E contains several arcs between the nodes i and j, with corresponding values.

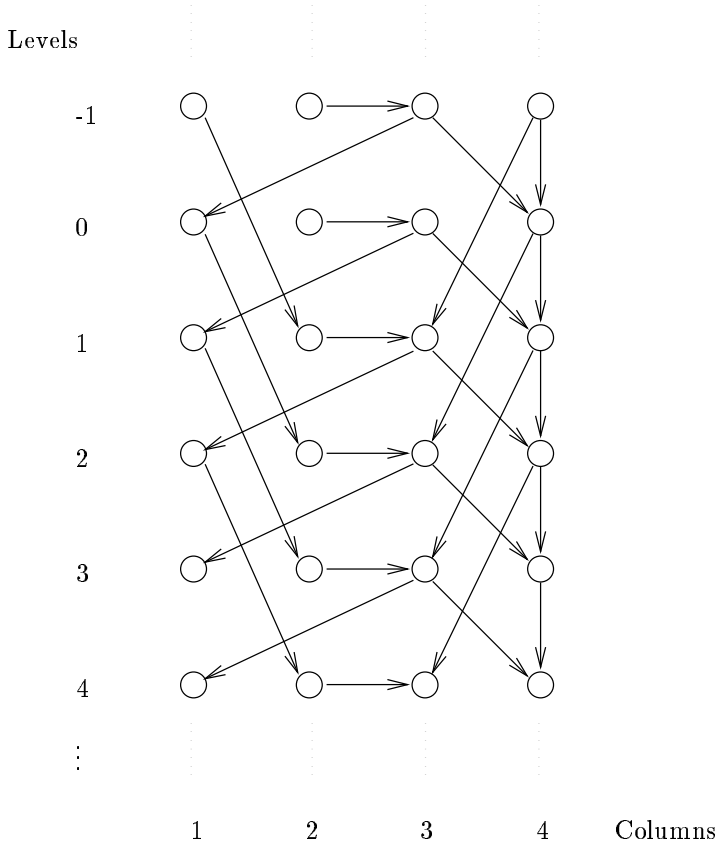


Figure 1: Dependence graph associated with the system S of Equation (3).

There is an arc from i to j in E, if and only if there are arcs from the column $(i,.)$ to the column $(j,.)$ in the dependence graph. Note that system $\mathcal{S}$ is constructive if and only if the sum of the delays along any circuit in $\mathcal{R}$ is positive.

The reduced graph associated with the system $\mathcal{S}$ of Example 2.3 is represented in Figure 2. The delays γ associated with the arcs are depicted in boxes.

Reduced graphs appear in the literature under the following names : computation graph, Synchronous Data Flow (SDF) graphs, process graphs or uniform graphs.

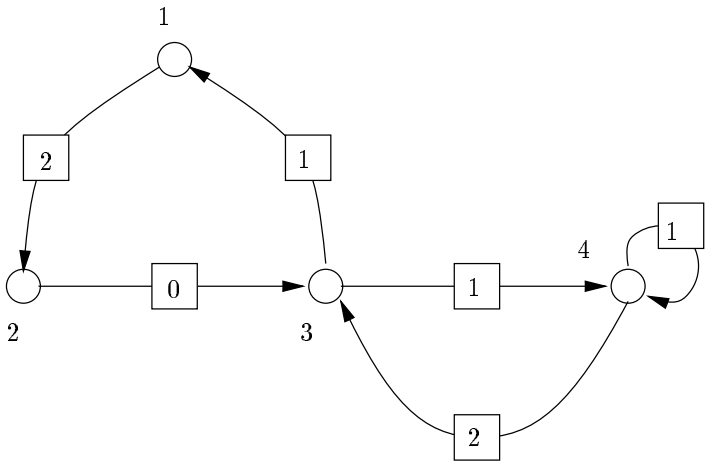


Figure 2: Reduced graph associated with the system $\mathcal{S}$ of Equation (3).

2.3 Recycled case

In the following, we will particularly study a special case of URE, where the computation of the variable $X_i(n)$ cannot be done before the computation of $X_i(n-1)$. This case appears naturally in marked graphs to impose a FIFO behavior (see section 7) and in other applications. This constraint can be modeled by imposing a dependence between $X_i(n-1)$ and $X_i(n)$, for all i and n. Formally, it results in having $(i,1) \in \mathcal{E}_i, \forall i$, for the system of URE. Equivalently, it results in having a self loop with delay one (hence the name recycled) in each node of $\mathcal{R}$, or in having arcs between the nodes (i,n) and $(i,n+1)$ in $\mathcal{D}$.

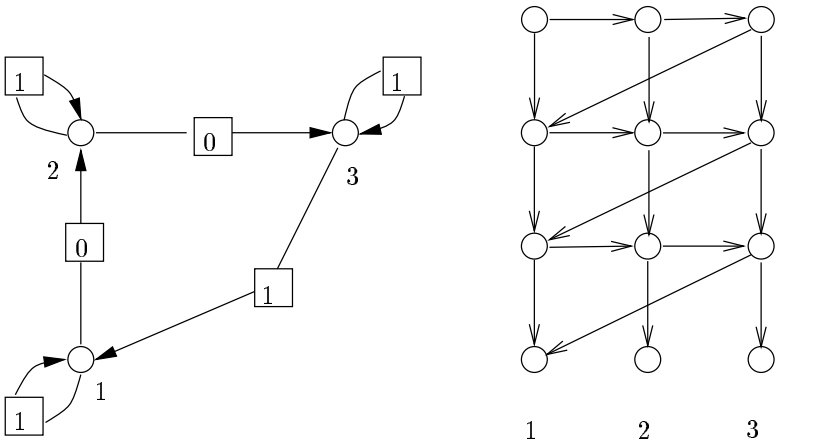
Figure 3 depicts an example of a recycled system.

2.4 Preliminary remarks

It should be clear from the definitions that there is a one to one correspondence between the three models. Indeed, a system can be given by its reduced graph as well as its dependence graph or system of equations.

If $\mathcal{R}$ is not connected, then the system of URE is made of two or more independent systems which can be studied independently. In the rest of the paper we will always assume that the graph $\mathcal{R}$ is connected.

In the following, we will study more precisely the relations between $\mathcal{D}$ and $\mathcal{R}$.



Recycled reduced graph Recycled dependence graph

Figure 3: Recycled reduced and dependence graphs.

3 Parallel Executions

In this part, we will define the problem to be investigated in the rest of the paper. Roughly speaking, we want to minimize the memory size required to compute a system $\mathcal{S}$. In §3.1, we discuss the main issues that have been studied in the field of uniform recurrence equations and we explain the difference with the aim of this paper. Section §3.2 gives a formal definition of the question we investigate in the following.

3.1 Relations with the basic scheduling problem

Organizing efficient computations for uniform recurrence equations on parallel computers has now taken a considerable importance in the literature. In the past, the investigations have often been oriented towards speeding up the execution with no or little consideration for memory requirements.

Assume that initially the negative variables $X_i(n), n \leqslant 0$ are known. The time required to compute a variable $X_i(n), n > 0$ in a system of URE is necessarily larger than the length of a critical path in $\mathcal{D}$ from level 0 to (i, n). A computation of a system of URE is *as soon as possible* if the time it takes to compute the variable

$X_i(n)$ is exactly given by the critical path from level 0 to (i,n). A first question that can be addressed is :

What is the number of processors required to carry out
a computation as soon as possible ?

This number is often called the *degree of parallelism* of the URE. In general, the solution is given by the size of the maximal anti-clique in the dependence graph.

Once this question is settled and provided a sufficient number of processors are available (*i.e.* larger than the degree of parallelism of the system), a problem is to characterize the as soon as possible schedule. This problem is often called the basic scheduling problem. It has been proved that for the as soon as possible schedule, the time at which the variable $X_i(n)$ is computed is of the form $\lambda n + d_i$, where λ is called the cycle time of the system (see [4]). Such a schedule is said to be linear. For systems of higher dimension (i.e. when $K = \mathbb{Z}^p$, $p > 1$ in Equation (1)), partial results on the optimality of linear schedules can be found in [5, 6].

When the number of available processors is fixed and less than the degree of parallelism of the URE, finding the optimal schedule becomes NP-hard, see [11].

All the results mentioned above are related with the problem of minimizing the number of processors used. This paper is concerned with the following dual problem : how much memory is necessary to carry out the computations of an URE, the number of processors being unlimited.

First, we should say that, in general, a computation as soon as possible requires a lot of memory which may not even be bounded in n. This makes the alternative to find a computation using a smaller memory size attractive. Second, the usual time-space trade-off tells us that some interesting results can be expected to arise when minimizing the memory size.

As one can expect, the schedule we will propose will not be as soon as possible in general. The time required to carry out the computation of variable $X_i(n)$ will be of the form $\lambda' n + d'_i$, where $\lambda' \geqslant \lambda$. Furthermore, the number of processors needed to carry out the computation will be greater than the degree of parallelism, but smaller than the memory size involved in the computation. In spite of these two drawbacks, the schedule we propose provides a new insight on the best ways to compute a system of URE and has interesting practical applications, see §6 and §7.

As shown in the following, the minimal size of the memory is related with *cuts* in the dependence graph (instead of anti-cliques for the minimal number of processors).

3.2 Pebble game

Let us work with an URE, S and its associated dependence graph $\mathcal{D}$ as defined in §2.1. We want to compute iteratively all the variables $X_i(n)$. At each step, the variables which are necessary to carry out the computations have to be stored in some memory locations.

We want to determine the minimal number of memory locations needed to compute all the variables $X_i(n)$.

We give a description of this problem in terms of a pebble game (see [15]).

Game 1. *We consider a dependence graph $\mathcal{D}$. At step 0, one puts a finite number of pebbles on negative nodes, i.e. on nodes $(i,n), n \leqslant 0$. By convention, we assume that at least one of these pebbles is on level 0. At each step of the game, the following moves are allowed.*

Move 1 *: Put a pebble on a node (i,n) if on each infinite oriented path (see Definition 4.1) ending in (i,n) there is at least one node with a pebble.*

Move 2 *: Remove a pebble from a node.*

An *execution* of the game is *successful* if all positive nodes receive a pebble during the execution. In the following, we will always refer to successful executions simply as executions. The set of (successful) executions will be denoted $\mathcal{E}$. A step of the game may have a duration different from 1 unit of time, see the discussion in Remark 3.2.

Definition 3.1 (configuration). *In an execution, the position of the pebbles at step $t \in \mathbb{N}$, is called the t-th configuration and is denoted $\mathcal{A}(t)$. In particular, $\mathcal{A}(0)$ corresponds to the set of initial pebbles.*

Pebbles correspond to memory locations. A pebble put on a node corresponds to the computation of a new data and its storage in the memory. The removal of a pebble corresponds to the erasing of a data from the memory. An execution corresponds to a computation of all the nodes in the dependence graph. Our objective will be to find an execution of the game using a number of pebbles which is as small as possible. The total number of pebbles used by an execution $e \in \mathcal{E}$ is

$$\mathcal{P}(e) \stackrel{\text{def}}{=} \max_{t \in \mathbb{N}} \mathcal{A}(t).$$

Our objective can be redefined as follows:

Problem 1. *We want to find an execution* $e_o \in \mathcal{E}$ *such that* $\mathcal{P}(e_o) = \min_{e \in \mathcal{E}} \mathcal{P}(e)$.

The definition of Move 1 implies that we are allowed to perform function compositions during one step of the execution (see for example Figure 6, rule $\mathcal{M}_3$).
We have to take care of the fact that function composition has a cost. In order to do so, we assume that step t lasts l time units where l is the length of the longest path in $\mathcal{D}$ joining a node marked at step $t-1$ to a node marked at step t. Note that l is also the longest chain of function compositions performed during step t.

Remark 3.2. Here is a possible execution of the game. The initial pebbles $\mathcal{A}(0)$ remain unchanged along the execution. An additional pebble is used to mark successively all the nodes in the graph. In this case, marking all the nodes up to level n takes $\Omega(n^2)$ units of time. On the other hand, we say that an execution has a *linear* time complexity if it puts a pebble on node (i,n) within $O(n)$ time units for all n. The set of linear executions is not empty. For example, if we mark the nodes as soon as possible, then node (i,n) is marked at time $\lambda n + d_i$.
The executions that we are going to propose to solve Problem 1 will not be optimal in terms of time complexity (i.e. will not be asap). However, they will always be linear, which means that the loss in terms of time efficiency is kept under control (see also the discussion in §3.1).

In order for the pebble game to be rigorously defined, we need to have some additional rules. We are going to define four different set of rules $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$ and $\mathcal{M}_4$.

$\mathcal{M}_1$: Asynchronous execution We consider two additional rules. In particular, we further constrain the rule of Move 1.

- **Move 1'** : Put a pebble on a node if all the predecessors (for the precedence relation) of this node have a pebble.
- It is possible to perform one move of type 1' and several moves of type 2 during one step of the game. On the other hand, it is not possible to perform several moves of type 1'.

Let us consider the example of Figure 4. We have represented a small part of the dependence graph of the URE $X_i(n) = F_i(X_1(n-1), X_2(n-1), X_3(n-1)), \forall i = 1,2,3$.

At step t, there are pebbles on the nodes $(1,n),(2,n)$ and $(3,n)$. At step $t+1$, we can put a pebble on node $(1,n+1)$ as all the predecessors (i,n) have a pebble.

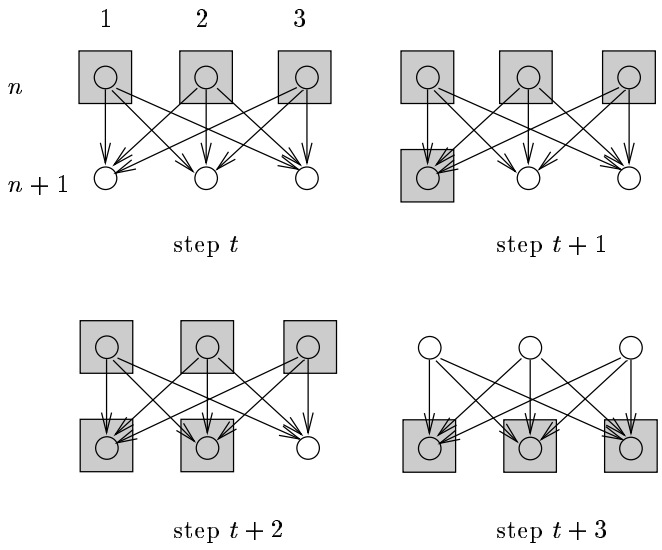


Figure 4: Asynchronous rule, $\mathcal{M}_1$, five pebbles are needed.

At step $t+2$, and for the same reason, we can put a pebble on node $(2, n+1)$. At step $t+3$, we put a pebble on node $(3, n+1)$ and we remove the pebbles on nodes (i, n) (they are not needed anymore). The minimal number of pebbles needed to describe the dependence graph of Figure 4 is 5.

This rule corresponds to the necessity of performing asynchronous computations. It is relevant if we use a sequential computer to perform the calculations. In this case, the maximal number of pebbles used during the game corresponds to the minimal number of memory locations needed to carry out the computation.

Remark 3.3. When Game 1, rule $\mathcal{M}_1$, is performed on a binary tree, the minimal number of pebbles is known as the Strahler's number. This Strahler's number appears in various fields ranging from hydrology or combinatorics to molecular biology. For a nice review paper, the reader is referred to Viennot [18].

In the forthcoming set of rules, we switch back to the original definition of Move 1, see Game 1.

$\mathcal{M}_2$: Synchronous execution We consider Game 1 with the following additional rule.

- Several moves of type 1 and several moves of type 2 can be performed at the same step of the game.

We consider, in Figure 5, the same example as previously under the new set of rules.

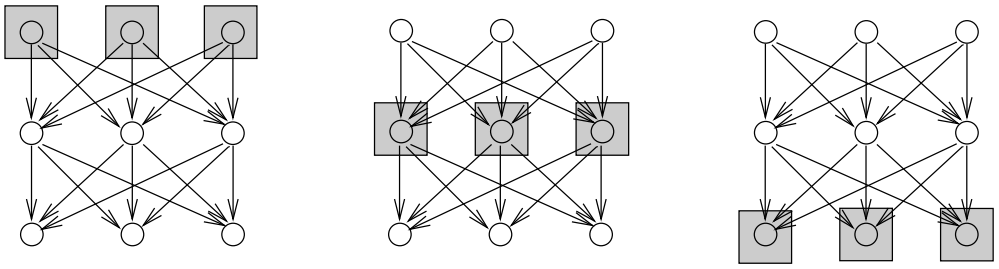


Figure 5: Synchronous rule, $\mathcal{M}_2$. Three pebbles are needed.

At step n, we have three pebbles on nodes (i,n), $i=1,2,3$. At step $n+1$, we put simultaneously three pebbles on nodes $(i,n+1)$ and we remove the initial pebbles. Hence, the number of pebbles needed by this execution is three.

This rule corresponds to the case where several synchronous processors are used during the computations. It will be adapted if we use parallel synchronous processors with shared memory to carry out the calculations.

$\mathcal{M}_3$: Synchronous regular execution We consider Game 1 with two additional rules

- Several moves of type 1 and several moves of type 2 can be performed at the same step of the game.
- If at step t the configuration is $\mathcal{A}(t)$, then at step $t+1$, the configuration is $(\mathcal{A}(t)+1)$ defined by :

$$(i,n) \in \mathcal{A}(t)+1 \iff (i,n-1) \in \mathcal{A}(t)\,.$$

The example of Figure 5 was also verifying the set of rules $\mathcal{M}_3$. To see that $\mathcal{M}_2$ and $\mathcal{M}_3$ are different, let us consider the example of Figure 6. It corresponds to the URE $X_1(n) = F_1(X_2(n-1)), X_2(n) = F_2(X_1(n))$.

In Figure 6 (I), only one pebble is needed. The corresponding execution verifies rule $\mathcal{M}_2$ but not rule $\mathcal{M}_3$. In Figure 6 (II), two pebbles are needed. The corresponding execution verifies rule $\mathcal{M}_3$. The computations are performed according to the following patterns :

- Rule $\mathcal{M}_2$ (Figure 6 (I)).
  - step t : $X_2(n) = F_2(X_1(n))$.
  - step $t+1$: $X_1(n+1) = F_1(X_2(n))$.
  - step $t+2$: $X_2(n+1) = F_2(X_1(n+1)) \ldots$
- Rule $\mathcal{M}_3$ (Figure 6 (II)).
  - step t : $(X_1(n+1), X_2(n+2)) = (F_1 \circ F_2(X_1(n)), F_2 \circ F_1(X_2(n+1)))$.
  - step $t+1$: $(X_1(n+2), X_2(n+3)) = (F_1 \circ F_2(X_1(n+1)), F_2 \circ F_1(X_2(n+2))) \ldots$

Note that in the execution under rule $\mathcal{M}_3$, we perform function compositions, $F_2 \circ F_1$ and $F_1 \circ F_2$. Hence each step lasts two time units.

Rule $\mathcal{M}_3$ has several advantages. First, the number of pebbles needed to carry out the calculations, is easy to compute, it is equal to $|\mathcal{A}(t)|$ (independent of t). Second, a regular execution is interesting for implementation purposes. It provides an easy computation of the variables in the new configuration from the ones in memory by always applying the same operator. A non regular execution could be practically very intricate to implement.

$\mathcal{M}_4$: Synchronous regular non-anticipative execution

- Several moves of type 1 and several moves of type 2 can be performed at the same step of the game.
- If at step t the configuration is $\mathcal{A}(t)$, then at step $t+1$, the configuration is $(\mathcal{A}(t)+1)$ defined by $(i,n) \in \mathcal{A}(t)+1 \iff (i,n-1) \in \mathcal{A}(t)$.
- A path (see Definition 4.1) from a node in $\mathcal{A}(t)$ to a node in $\mathcal{A}(t+1)$ contains only nodes belonging either to $\mathcal{A}(t)$ or to $\mathcal{A}(t+1)$.

Let us consider the example of Figure 6. In Figure 6 (II), we have an example of an execution which verifies rule $\mathcal{M}_3$ but not $\mathcal{M}_4$. For example (see above), the node $(1, n+2)$ is used at step t but is computed only at step $t+1$. On the other hand, in Figure 6 (III), we have an execution which verifies rule $\mathcal{M}_4$. In the example of Figure 6. The corresponding computation pattern is :

- Rule $\mathcal{M}_4$ (Figure 6 (III)).
  - step t : $(X_1(n+1), X_2(n+1)) = (F_1(X_2(n)), F_2 \circ F_1(X_2(n)))$.
  - step $t+1$: $(X_1(n+2), X_2(n+2)) = (F_1(X_2(n+1)), F_2 \circ F_1(X_2(n+1)))\ldots$

Remark 3.4. In Figure 6, the number of pebbles is the same for the two set of rules $\mathcal{M}_3$ and $\mathcal{M}_4$. It is not always the case, see Figure 10.

3.3 Summary

In the following, we will use the notations :

- $\mathcal{E}$: the set of all possible (synchronous) executions under rule $\mathcal{M}_2$.
- $\mathcal{RE}$: the set of all possible executions under rule $\mathcal{M}_3$. Elements of $R\mathcal{E}$ will be called regular executions.
- $\mathcal{NRE}$: the set of all possible executions under rule $\mathcal{M}_4$. Elements of $\mathcal{NRE}$ will be called non-anticipative regular executions.

Note that

$$\mathcal{NRE} \subset \mathcal{RE} \subset \mathcal{E}\,.$$

Complexity Results Under rule $\mathcal{M}_1$, the problem of determining the minimal number of pebbles to compute a general directed acyclic graph with one final node has been considered by Sethi [16]. In that paper, it is proved that this problem is NP-complete. Here, we can easily embed any directed acyclic graph on each level of a recycled dependence graph. We also embed the same acyclic graph between two levels of $\mathcal{D}$ (see figure 7).

Now, it is not difficult to see that the minimal number of pebbles needed under rule $\mathcal{M}_1$ in this dependence graph is the minimal number of pebbles necessary to carry out the computation on the original acyclic graph plus the number of columns in $\mathcal{D}$. Therefore, our problem under rule $\mathcal{M}_1$ is also NP-complete.

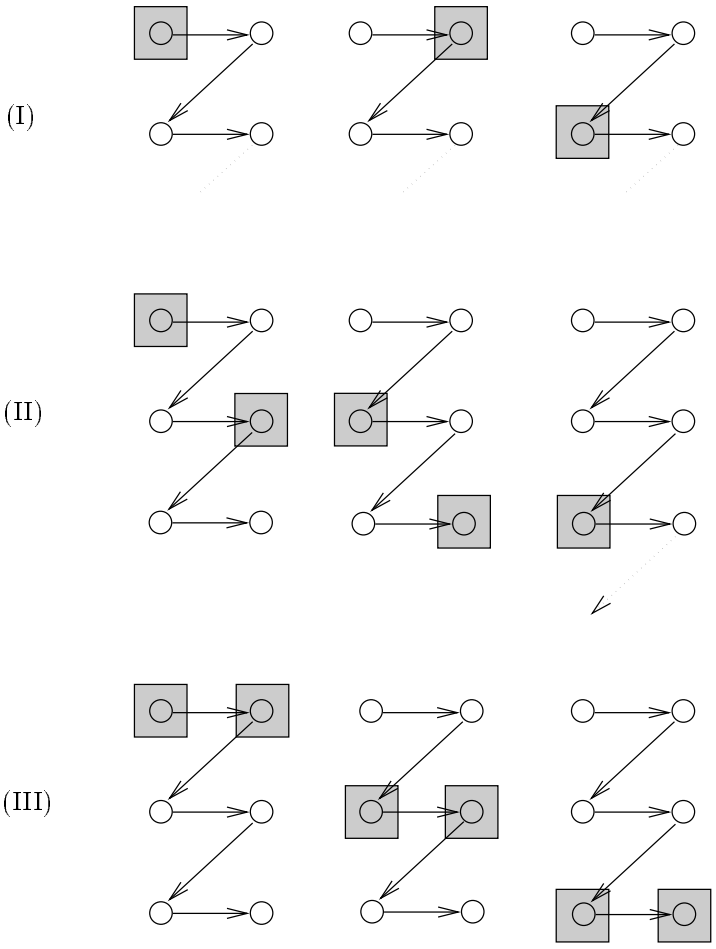


Figure 6: Rule $\mathcal{M}_2$ (I), rule $\mathcal{M}_3$ (II) and rule $\mathcal{M}_4$ (III).

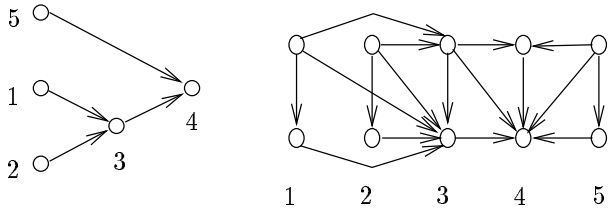


Figure 7: Embedding of an arbitrary acyclic graph in a dependence graph.

In the following we will only consider synchronous executions, i.e. the set of rules $\mathcal{M}_2$, $\mathcal{M}_3$ and $\mathcal{M}_4$. In particular, we will characterize executions using a minimal

number of pebbles under rules $\mathcal{M}_2$, $\mathcal{M}_3$ and $\mathcal{M}_4$ and we are going to prove that the minimal number of pebbles can be found in polynomial time for the recycled case.

4 Cuts and Pebbles

From now on, we consider the recycled case, see §2.3. It is always implicitly assume (unless otherwise specified) that the system under study is recycled.

4.1 Cuts in $\mathcal{D}$

Let us recall some classical definitions of graph theory, all defined in the dependence graph, $\mathcal{D}$. For further references, see [7] or [10] for example.

Definition 4.1 (path). *A path is a sequence of nodes and arcs in $\mathcal{D}$ of the form $\cdots \to (i_0, n_0) \to (i_1, n_1) \to (i_2, n_2) \to \cdots \to (i_k, n_k) \to \cdots$. A path is bi-infinite if it contains an infinite number of negative nodes and an infinite number of positive nodes.*

Definition 4.2 (cut). *A cut C is a set of nodes in $\mathcal{D}$ such that any bi-infinite path contains at least one node of C. A cut of minimal size is called a minimal cut.*

Definition 4.3 (flow). *A flow is a set of bi-infinite paths such that any two paths do not share any node. A flow containing a maximal number of paths is called a maximal flow.*

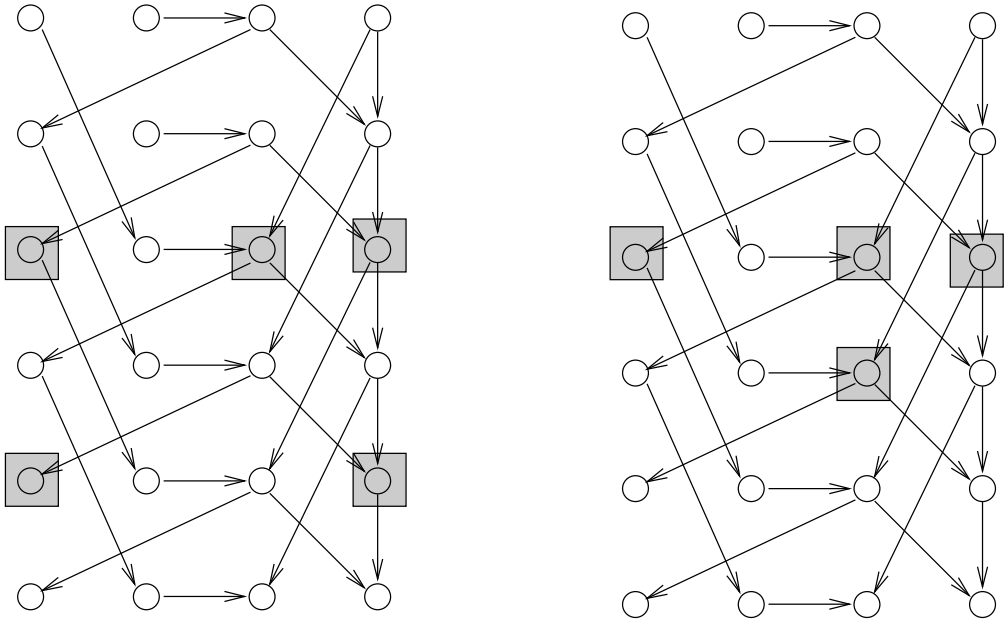
The most classical notion of cut involves arcs rather than nodes and a flow is a set of paths which do not share arcs rather than nodes. However a small transformation of each node into two nodes connected by an arc would allow us to go back to the original notions.

Definition 4.4 (consecutive cut). *A cut C in $\mathcal{D}$ is consecutive if on each column of $\mathcal{D}$, C contains only consecutive nodes, i.e.*
For all $i \in V$, $(i, n) \in C$ and $(i, n+1) \notin C \Rightarrow (i, n+k) \notin C, \forall k > 0$.

An example of consecutive and non-consecutive cuts is displayed in Figure 8.

Lemma 4.5. *There exists a minimal cut of $\mathcal{D}$ which is a minimal consecutive cut.*

Proof. Let C be a minimal consecutive cut. We will prove that C is a minimal cut. First, note that there are no arcs from C_u to $C_l + k, k \geqslant 2$, otherwise C would not be a cut.



A non consecutive cut

A consecutive cut

Figure 8: Consecutive and non consecutive cuts (on a non-recycled example).

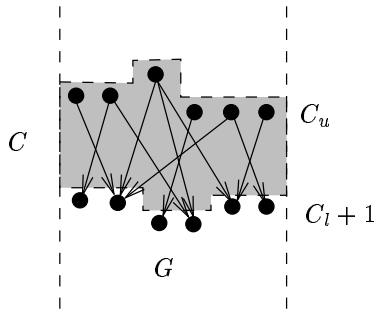


Figure 9: Graph G made from the lower and upper sections of C

Now, consider the sub-graph G of $\mathcal{D}$ made of the nodes $C_u \cup (C_l + 1)$ and the arcs between C_u and $C_l + 1$ in $\mathcal{D}$. See figure 9.

A cut in a finite graph G is a set of nodes such that, when removed from G, there is no arc remaining in G. Let Δ be a cut in G of minimal size. If $|\Delta| < |C_u|$ then $C \backslash C_u \cup \Delta$ would be a consecutive cut in $\mathcal{D}$ strictly smaller than C, which contradicts the fact that C is a minimal consecutive cut. Therefore, we have $|\Delta| = |C_u|$.

An adapted version of a famous "minimax" theorem first proved by Ford-Fulkerson (see [7]) states that we can find $|\Delta|$ node-disjoint arcs in G. Since $|\Delta| = |C_u| = |C_l + 1|$, these arcs define a one to one mapping ϕ from C_u to $C_l + 1$. From ϕ we construct a flow in C in the following way. Select all the arcs of the form $((i,n) + k) \to (\phi(i,n) + k)$ for all $(i,n) \in C_u$ and all $k \in \mathbb{Z}$. These arcs form a flow $\mathcal{F}$ in $\mathcal{D}$ of size $|C|$.

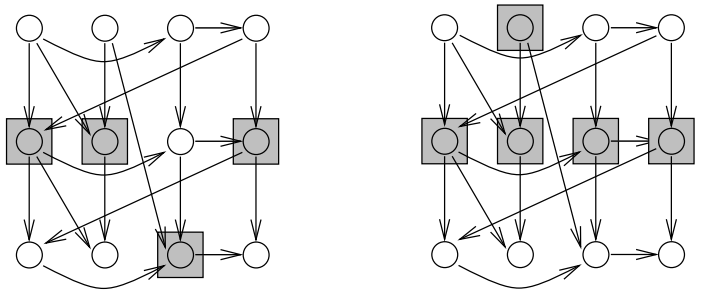
Let C_m be a minimal cut in $\mathcal{D}$. Since $\mathcal{F}$ is formed by node-disjoint paths, C_m must contain at least $|\mathcal{F}|$ nodes, $|C_m| \geqslant |\mathcal{F}| = |C|$. We conclude that $|C_m| = |C|$. □

This lemma is interesting by its own. In particular, it gives a proof of the minimax theorem (which exists in many versions) in an infinite graph $\mathcal{D}$.

Corollary 4.6. *The size of the minimal cut is equal to the size of the maximal flow in $\mathcal{D}$.*

Another immediate corollary of Lemma 4.5 is that there exists a maximal flow $\mathcal{F}$ in $\mathcal{D}$ which is 1-periodic (*i.e.* if the arc $(i,n) \to (j,m)$ belongs to $\mathcal{F}$, then $(i,n+1) \to (j,m+1)$ belongs to $\mathcal{F}$).

Definition 4.7 (section). *A section S in $\mathcal{D}$ is a set of nodes with exactly one node per column, $S = \{(i,n_i), i \in V\}$.*



Minimal cut Minimal compatible cut

Figure 10: Compatible and non compatible cuts.

Note that since $\mathcal{D}$ is recycled, a cut contains at least one node per column. Using this property, one can define the upper and lower sections of a cut.

Definition 4.8 (upper, lower section). *The upper (resp. lower) section C_u (resp. C_l) of a cut C is the set of nodes (i,n) in C such that the nodes $(i,n-h), h>0$ (resp. $(i,n+h), h>0$) do not belong to C.*

Definition 4.9. *We say that an arc crosses a section $S=\{(i,n_i), i\in V\}$ from top to bottom if it is an arc of the form $(i,n_i-h)\rightarrow(j,n_j+l)$ with $h\geqslant 0$ and $l\geqslant 1$. An arc $(i,n_i+l)\rightarrow(j,n_j-h)$ crosses S from bottom to top if $l\geqslant 1$ and $h\geqslant 0$.*

Definition 4.10 (compatible section, compatible cut). *A section in $\mathcal{D}$ is compatible if no arc crosses the section from bottom to top. A consecutive cut is said to be compatible if its lower section is compatible.*

Note that it can be that no minimal consecutive cut in $\mathcal{D}$ is compatible. This is the case in Figure 10 where the minimal compatible cut contains 5 nodes while a minimal consecutive cut of size 4 can be found.

4.2 Relations with executions of Game 1

Lemma 4.11. *A configuration of any execution $e\in\mathcal{E}$ is a cut in $\mathcal{D}$.*

Proof. Let $\mathcal{A}(t)$ be the t-th configuration of some execution e belonging to $\mathcal{E}$. Assume that $\mathcal{A}(t)$ is not a cut. By definition, there exists a bi-infinite path P which does not have any node in $\mathcal{A}(t)$. According to the rule $\mathcal{M}_2$ of Game 1, no positive node

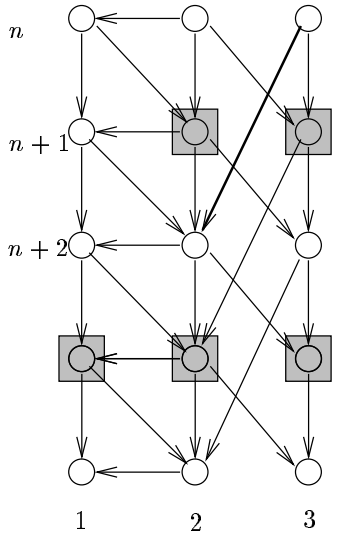


Figure 11: A non consecutive cut which is not a regular configuration.

on P can ever be marked during the execution after step t and all of them cannot have received a pebble during the t first steps. This contradicts the fact that e is an execution of $\mathcal{E}$. □

Lemma 4.12. *A consecutive cut is a configuration of a regular execution ($\mathcal{RE}$).*

Proof. Let C be a consecutive cut in $\mathcal{D}$. We have $C + 1 = (C \backslash C_u) \cup (C_l + 1)$. Therefore, C is a regular configuration if and only if for each node (i, n) in $C_l + 1$, there is no infinite path P terminating in (i, n) that does not intersect the cut. But if such a path could be found, then the bi-infinite path $P \cup \{(i, n+h), h \in \mathbb{N}\}$ would not intersect C. This contradicts the fact that C is a cut. □

Remark 4.13. Note that a non consecutive cut may not be a regular configuration, as illustrated in the example of Figure 11. In that example, the node $(2, n+2)$ belongs to $C + 1$ but cannot be computed using only variables in C (as it depends on $(3, n)$ for example). Therefore, the cut C is not a regular configuration.

Lemma 4.14. *A compatible cut is a configuration of a non-anticipative regular execution ($\mathcal{NRE}$).*

Proof. Let C be a compatible cut. Since C is consecutive by definition, Lemma 4.12 tells us that C is a configuration $\mathcal{A}(t)$ of a regular execution e. Suppose that e is anticipative. This means that there exists a node (say (i,n)) in $\mathcal{A}(t+1)$ with a predecessor (say (j,m)) not marked at steps t or $t+1$. The arc $(j,m-1) \rightarrow (i,n-1)$ crosses the lower section of C from bottom to top. This contradicts the fact that C is compatible. □

We are now ready to state the main result of this section.

Theorem 4.15. *Let us consider a recycled system of URE $\mathcal{S}$. We perform Game 1 on its associated dependence graph $\mathcal{D}$. We have*

$$\min_{e \in \mathcal{RE}} \mathcal{P}(e) = \min_{e \in \mathcal{E}} \mathcal{P}(e). \tag{5}$$

In other words, there exists a regular execution which requires a minimal number of pebbles.

Proof. The proof of the theorem is a direct consequence of Lemmas 4.11, 4.12 and 4.5. First, note that all configurations are cuts, Lemma 4.11. Let C be a consecutive cut of minimal size, which exists by Lemma 4.5. By Lemma 4.12, C is a regular configuration. □

Theorem 4.15 has several interesting corollaries. First, it allows one to focus on regular executions since no fancy irregular execution of the URE can be done with fewer pebbles. Then, it provides a polynomial method to find an optimal execution as shown in §4.3. As for non-anticipative executions, polynomial algorithms will be given in §6.4.

4.3 Complexity results

We are going to compute a maximal flow in $\mathcal{D}$ and then apply Corollary 4.6. If we want to use the algorithm of Ford and Fulkerson [7] to compute a maximal flow in $\mathcal{D}$, we need first to restrict ourselves to a finite graph.

We call *span* of a cut the difference between the smallest level and the largest level containing a node of the cut.

A *slice* of $\mathcal{D}$ from level0 to level n will be sufficient to compute the maximal flow in the graph if a consecutive minimal cut spans over less than n levels. So it is important to determine, or at least to bound, the span of a consecutive minimal cut.

Lemma 4.16. *The span of a minimal consecutive cut is smaller than the total sum of the delays in $\mathcal{R}$.*

Proof. Let C be a minimal consecutive cut. The associated maximal 1-periodic flow $\mathcal{F}$ is a set of paths in $\mathcal{D}$. First, note that these paths cover all the nodes in $\mathcal{D}$. Indeed, by the 1-periodicity of $\mathcal{F}$, if a node (i,n) is not in $\mathcal{F}$, then the whole column $(i,.)$ is not in $\mathcal{F}$, but this means that the bi-infinite path $\{(i,n), n \in \mathbb{Z}\}$ can be added to flow $\mathcal{F}$ and this would contradict the maximality of $\mathcal{F}$.

Let P_1 be any path in $\mathcal{F}$. Since P_1 is bi-infinite, P_1 intersects at least one column in $\mathcal{D}$ infinitely often. We denote by $i_0, i_1, \cdots, i_{l_1}$ the successive columns visited by the path P_1. Now, using the 1-periodicity of $\mathcal{F}$, the total number of paths intersecting columns $i_0, i_1, \cdots, i_{l_1}$ in $\mathcal{F}$ is k_1. By definition of $\mathcal{R}$ there exists a circuit (L_1) in $\mathcal{R}$ containing the nodes $i_0, i_1, \cdots, i_{l_1}$ and of total delay k_1. The span of C on columns $i_0, i_1, \cdots, i_{l_1}$ is smaller than k_1.

A new path P_2 in $\mathcal{F}$ not intersecting columns $i_0, i_1, \cdots, i_{l_1}$ ranges over different columns $i_{l_1+1}, i_{l_1+2}, \cdots, \cdots, i_{l_2}$ and defines a circuit (L_2) in $\mathcal{R}$ similarly. The span of C on columns $i_{l_1+1}, \cdots, i_{l_2}$ is smaller than k_2. We apply the same arguments to all the paths in $\mathcal{F}$ until all columns in $\mathcal{D}$ are covered. This defines a set of circuits H in $\mathcal{R}$ covering $\mathcal{R}$.

We build a new graph $\mathcal{G}$ starting with $\mathcal{R}$ where each circuit in H is aggregated into one node. $\mathcal{G}$ has $|H|$ nodes and all the arcs in $\mathcal{G}$ do not belong to any circuit in H.

The span of C is smaller than the sum of the span on all circuits in H plus the sum of the delays on all the arcs in $\mathcal{G}$. Note that no delay is counted twice in this upper bound. Therefore, the total span (M) of C is smaller than the total sum of the delays in $\mathcal{R}$. □

Remark 4.17. This bound is tight since it is not difficult to exhibit examples in which the span of the minimal cut is the sum of all the delays in $\mathcal{R}$. However, in most cases, the span of a minimal consecutive cut is significantly smaller.

Let M be the sum of the delays in $\mathcal{R}$. A slice of $\mathcal{D}$ with M levels has the same cut size as $\mathcal{D}$ itself. The computation of the minimal cut in a finite slice of $\mathcal{D}$ can be done using the augmenting path algorithm, see [7] [10]. Starting with a 1-periodic flow (the recycled columns) and maintaining the 1-periodicity throughout the construction yields a maximal 1-periodic flow. The complexity of the construction of the maximal flow is $O(M^2k^2)$. By Corollary 4.6, it provides the size of a minimal (consecutive) cut in $\mathcal{D}$.

5 Cuts and Delays

In this section, we will exhibit the relations that can be found between cuts in $\mathcal{D}$ and values of the delays in $\mathcal{R}$.

5.1 Retiming

A *retiming* of $\mathcal{R}$ is a transformation of the graph resulting in a decrease or increase of the values of the delays but with no transformation of the graph topology. This notion has been described in digital circuits to move registers (see §6) and in Petri nets, where a retiming corresponds to the firing of transitions (see §7).

Definition 5.1 (retiming). *A retiming of $\mathcal{R}$ is a node function $r : V \rightarrow \mathbb{N}$ which specifies a new value of the delays. After retiming r, the value of the delay on an arc (i,j) in the new graph $\mathcal{R}_r$ is $\gamma_r = \gamma + r(i) - r(j)$.*

In the example of Figure 12, the new values of the delays correspond to a retiming r such that $r(1) = 1, r(2) = 1$ and $r(3) = 0$.

Note that after a retiming r of $\mathcal{R}$ the delay on one arc can be negative as in Figure 12.

Lemma 5.2. *Two retimings r and r' yield the same value of the delays in a connected graph $\mathcal{R}$ if and only if there exists a constant $h \in \mathbb{Z}$ such that $\forall i \in V, r(i) = r'(i) + h$.*

Proof. First, if $r(i) = r'(i) + h$ for all $i \in V$, then on any arc (i,j), $\gamma_r = \gamma + r(i) - r(j) = \gamma + r'(i) - r'(j) = \gamma_{r'}$. Conversely, if $\gamma_{r'} = \gamma_r$, then $r(i) = r'(i) + h$ and $r(j) = r'(j) + h$ for some $h \in \mathbb{Z}$. The fact that $\mathcal{R}$ is connected implies that the constant h is the same for all the nodes in V. □

The question that arises now is what is the corresponding notion in the graph $\mathcal{D}$? To answer this question, let us consider the graph $\mathcal{D}_r$ associated with the retimed reduced graph $\mathcal{R}_r$. This dependence graph can be constructed directly from $\mathcal{D}$ by shifting the columns as described in Lemma 5.3.

Lemma 5.3. *A retiming r in $\mathcal{R}$ corresponds to an isomorphism f_r between $\mathcal{D}$ and $\mathcal{D}_r$ defined by:*

$$\begin{array}{rccc} f_r : & \mathcal{D} & \rightarrow & \mathcal{D}_r \\ & (i,n) & \rightarrow & (i, n - r(i)) \end{array}$$

The function f_r will also be called a retiming of $\mathcal{D}$.

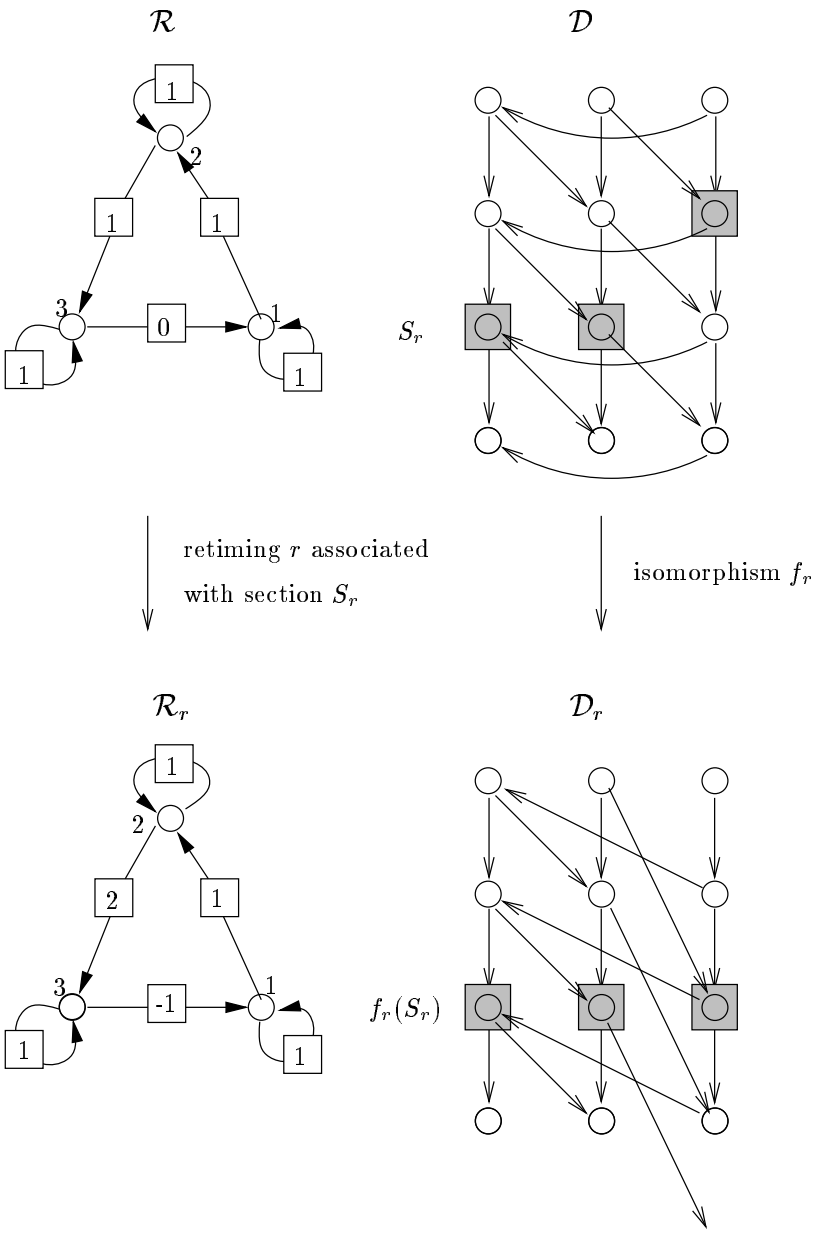


Figure 12: Retimed reduced graph and dependence graph.

Proof. By definition of $\mathcal{D}_r$, there is an arc from (i,n) to (j,m) in $\mathcal{D}_r$ if the delay in $\mathcal{R}_r$ on arc (i,j) is $\gamma_r = m-n$. We have $\gamma_r = \gamma + r(i) - r(j) = (m-r(j)) - (n-r(i))$. Therefore, f_r is an isomorphism between $\mathcal{D}$ and $\mathcal{D}_r$. □

We recall that the notion of *section* was defined in 4.7. A retiming r in $\mathcal{R}$ can be associated with the section $S_r = \{(i, r(i)), i \in V\}$ in $\mathcal{D}$.

Lemmas 5.2 and 5.3 tell us that two retimings r and r' are similar (in the sense that they yield the same value of the delays) if and only if they are associated with two sections S_r and S'_r with $S_r = S_{r'} + h$, for some $h \in \mathbb{Z}$. This relation enables us to define a parallelism relation between sections in $\mathcal{D}$ as well as between retimings in $\mathcal{R}$. We say that section S_r (resp. retiming r) is equivalent to section $S_{r'}$ (resp. retiming r') if $S_r = S_{r'} + h$, for some $h \in \mathbb{Z}$. In the following, we will always consider one arbitrary section among the equivalent class and call it the section associated with retiming r.

5.2 Counting the delays

Given a graph $\mathcal{R}$, there are different possible ways to count the number of delays involved in the graph. We are going to propose two different ways of counting, mode A and mode B.

Mode A : The number of delays in $\mathcal{R}$ is

$$\Gamma_A = \sum_{i \in V} \sum_{(j,\gamma) \in \mathcal{E}_i} \gamma \,. \tag{6}$$

Mode A corresponds to the exact number of registers appearing in the graphical representation of the reduced graph $\mathcal{R}$ as defined in §2.2. See for example, Figure 13 (A).

Mode B : The number of delays in $\mathcal{R}$ is

$$\Gamma_B = \sum_{j \in V} \max\{\gamma \mid \exists i \text{ s.t. } (j,\gamma) \in \mathcal{E}_i\} \,. \tag{7}$$

Let us explain this mode of counting. Assume that node $j \in V$ has several output arcs with respective delays $\gamma_1, \gamma_2, \cdots, \gamma_l$. If we allow the possibility to share the delays between these l arcs, the number of delays will be counted as $\max_k \gamma_k$ instead of $\sum_k \gamma_k$ as in Mode A.

Graphically, Γ_B corresponds to the number of delays in a modified reduced graph where we have performed a forward splitting of the nodes. An example is provided in Figure 13 (B). We have added a "dummy" node, represented by a black dot, with function F =Identity. The other nodes remain unchanged. This reduced graph describes exactly the same system of URE. More precisely, the variables computed at the white nodes in Figure 13 (B) are the same as the variables computed in Figure 13 (A).

There is another way to interpret mode B. Let us assume for a moment that we modify the definition of a reduced graph. We consider a reduced graph $\tilde{\mathcal{R}}$ where delays are put on nodes instead of arcs. The total number of delays in graph $\tilde{\mathcal{R}}$ is equal to Γ_B. This is illustrated in Figure 13 $(\tilde{B})$.

In the following, we will say, with some abuse of language, that counting mode A corresponds to delays on arcs and counting mode B to delays on nodes.

Other ways of enumerating delays are conceivable. We will not consider them as they appear to be less interesting, mathematically speaking as well as from a practical point of view.

Deciding which counting mode of the delays to choose is very important. Different modes will yield different optimal graphs, after minimization of the number of delays.

5.3 Cuts and delays

We recall that the system under study is assumed to be recycled.

Consider a section $S = \{(i, n_i), i \in E\}$ in $\mathcal{D}$. We define a consecutive cut $\mathcal{C}(S)$ of the graph in the following way. On column i, $\mathcal{C}(S)$ contains all the nodes (i, n) such that there is an arc from a node (i, n) to a node $(j, m), m > n_j$.

$$\mathcal{C}(S) \stackrel{\text{def}}{=} \{(i,n), i \in V, n \leqslant n_i \mid \exists j \in V, m > n_j, (i,n) \to (j,m)\}\,.$$

Note that $\mathcal{C}(S)$ is a cut with lower section S. Furthermore, if any node is removed from the upper section of $\mathcal{C}(S)$, then it is not a cut anymore.

In a cut C, a node $(i, n) \in C$ is *redundant* if $C \backslash \{(i, n)\}$ is a cut. Note that any consecutive cut C with no redundant node on its upper section is characterized by its lower section S_l only. More precisely, we have $C = \mathcal{C}(S_l)$.

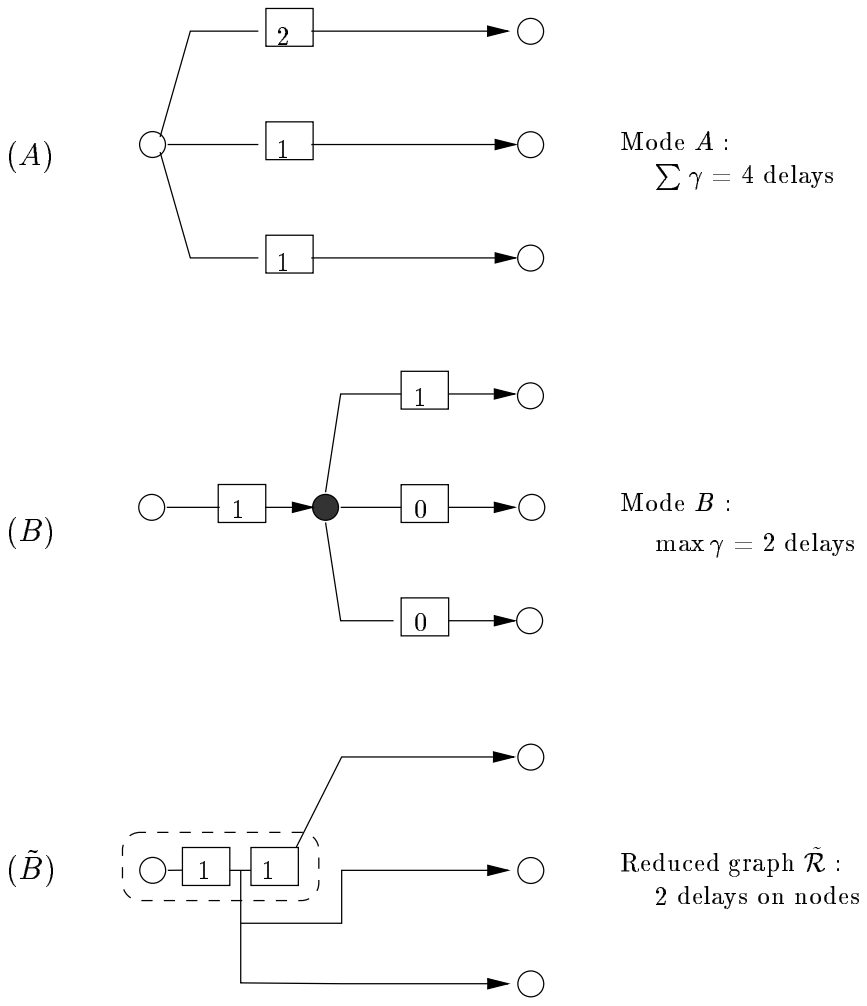


Figure 13: Different ways to enumerate the delays.

We are now ready to state the relations between delays in $\mathcal{R}$ and consecutive cuts in $\mathcal{D}$.

Lemma 5.4. *Let r be a retiming of $\mathcal{R}$ and S_r an associated section in $\mathcal{D}$. Then the number of delays in $\mathcal{R}_r$ under mode B (Γ_B) is equal to the cardinal of the cut $\mathcal{C}(S_r)$.*

Proof. We recall that the section associated with r is $S_r = \{(i, r(i)), i \in V\}$. First, let us prove that the size of $\mathcal{C}(S_r)$ (cut in $\mathcal{D}$) is the size of $\mathcal{C}(f_r(S_r))$ (in $\mathcal{D}_r$). Note that by definition, $f_r(S_r)$ is on a single level, $f_r(S_r) = \{(i, 0), i \in V\}$. From the definition of f_r, it should also be clear that $f_r(\mathcal{C}(S_r)) = \mathcal{C}(f_r(S_r))$ and *a fortiori* $f_r(\mathcal{C}(S_r))$ and $\mathcal{C}(f_r(S_r))$ have the same size.

It remains to be shown that the size of $\mathcal{C}(f_r(S_r))$ is the number of delays counted according to mode B. The delay on node i of $\mathcal{R}$ is the maximum of all γ_r for all (j, γ_r) in $\mathcal{E}_{i,r}$. This maximum (m) induces an arc in $\mathcal{D}_r$ from node $(i, -m+1)$ to node $(j, 1)$. By construction, $\mathcal{C}(f_r(S_r))$ contains exactly m nodes on column i: nodes $(i, 0), (i, -1), \cdots, (i, -m+1)$.

The same argument repeated on each column of $\mathcal{D}_r$ finishes the proof. □

We recall that given a section S, we defined the arcs crossing S from bottom to top or top to bottom in Definition 4.9.

Lemma 5.5. *Let r be a retiming of $\mathcal{R}$ and S_r an associated section in $\mathcal{D}$. The number of delays in $\mathcal{R}_r$ under mode A (Γ_A) is equal to the number of arcs in $\mathcal{D}$ crossing section S_r from top to bottom minus the number of arcs crossing S_r from bottom to top.*

Proof. By definition of f_r, all the arcs crossing $f_r(S_r)$ in $\mathcal{D}_r$ are the transform by f_r of the arcs crossing S_r in $\mathcal{D}$. We will rather count the arcs in $\mathcal{D}_r$. Pick one arc in $\mathcal{R}_r$ (i, j) with delay $\gamma \geqslant 0$. In $\mathcal{D}_r$, this arc induces exactly γ arcs crossing $f_r(S_r)$ from top to bottom, the arcs:

$$\begin{array}{rcl} (i,0) & \to & (j,\gamma) \\ (i,-1) & \to & (j,\gamma-1) \\ & \vdots & \\ (i,-\gamma+1) & \to & (j,1). \end{array}$$

Similarly, an arc in $\mathcal{R}_r$, (i, j) with delay $\gamma < 0$ induces exactly $-\gamma$ arcs crossing $f_r(S_r)$ from bottom to top:

$$\begin{array}{rcl} (i,-\gamma) & \to & (j,0) \\ (i,-\gamma-1) & \to & (j,-1) \\ & \vdots & \\ (i,1) & \to & (j,\gamma+1). \end{array}$$

The same argument applied to all the columns finishes the proof. □

Remark 5.6. Lemma 5.4 shows that one can describe the number of delays Γ_B as the cardinal of a set of nodes of $\mathcal{D}$. On the other hand, we deduce from Lemma 5.5 that the number of delays Γ_A is computed as the cardinal of a set of arcs in $\mathcal{D}$. This is natural as we have seen that (roughly speaking) mode A corresponds to delays on arcs and mode B to delays on nodes.

The notion of compatible cut introduced in Definition 4.10 has a very natural interpretation in terms of delays.

Proposition 5.7. *Let r be a retiming of $\mathcal{R}$ and S_r an associated section in $\mathcal{D}$. The retimed reduced graph $\mathcal{R}_r$ has only non-negative delays if and only if the section S_r is compatible in $\mathcal{D}$.*

In Figure 14 (this example is the same as the one of Figure 10), we have represented the retimed reduced graphs associated with two sections (cuts). One of them is compatible, Figure 14 (I), and the other one is not compatible, Figure 14 (II).

In §4.2, we have established the relations between configurations (for Game 1) and cuts in the dependence graph. In this paragraph, we have established the relations between cuts and delays. As an immediate by-product, we obtain the relations between delays and pebble configurations.

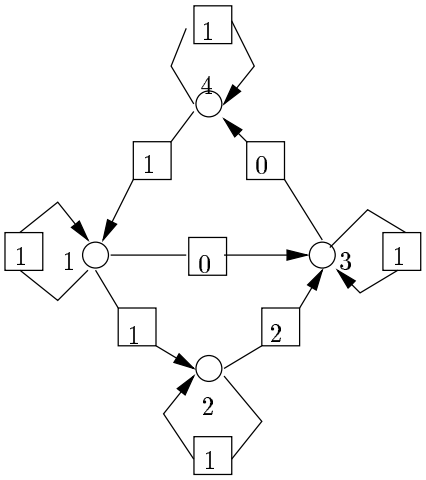
- An execution $e \in \mathcal{E}, e \notin \mathcal{RE}$ has configurations with different shapes at each step. Since any configuration can be viewed as a set of value of the delays in a retimed reduced graph, then an execution which is not regular provides a different value of the delays in $\mathcal{R}$ at each step of the computation.
- On the contrary, for an execution $e \in \mathcal{RE}$ the configurations are just shifted between two steps and this induces a fixed value of the delays in $\mathcal{R}$.
- Finally, an execution $e \in \mathcal{NRE}$ corresponds to fixed and non-negative delays.

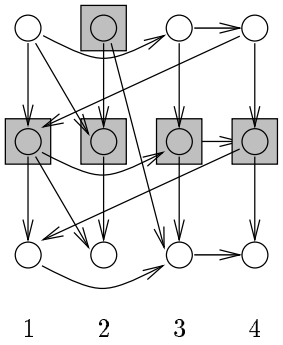
5.4 Summary and open problems

In the recycled case, the following table gives a summary of the main relations established so far between executions of a system of URE, cuts in $\mathcal{D}$ and delays in $\mathcal{R}$.

Executions of Game 1	Cuts in $\mathcal{D}$	Delays in $\mathcal{R}$
execution in $\mathcal{E}$	arbitrary cut	changing delays
regular execution, $\mathcal{RE}$	consecutive cut	fixed delays
non-anticipative reg. exec., $\mathcal{NRE}$	compatible cut	non-negative fixed delays

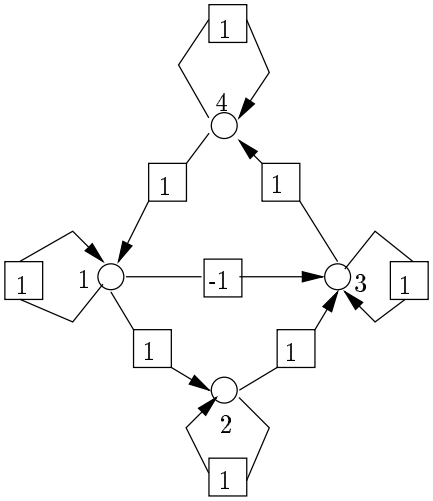
(I)



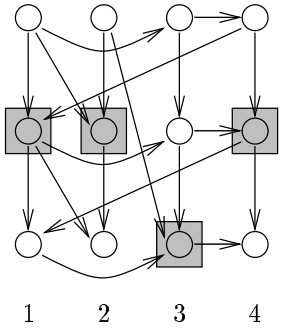


Minimal compatible cut

(II)



Corresponding delays



Minimal (non-compatible) cut

Figure 14: Compatible and non compatible cuts, non-negative and negative delays.

To complete the picture, it would be nice to extend all the results presented in this section to the non recycled case. The different definitions (cut, flow, section) extend easily to the non-recycled case. The main results which would make everything else easy to generalize are of two types. Results related with cuts in $\mathcal{D}$, §4.1, and results linking cuts and regular configurations, §4.2. For example, is it possible to find a minimal cut which is consecutive (generalization of Lemma 4.5) ? Can we find a minimal consecutive cut which is a regular configuration (generalization of Lemma 4.12)? It seems that most of these properties still hold in the non-recycled case but at this point the problem is still open.

One of the main results so far is that the size of the minimal consecutive cut is the same as the size of the minimal cut in $\mathcal{D}$ (see Lemma 4.5). However, the example displayed in Figure 14 shows that in some cases this minimal consecutive cut is not compatible and therefore, its associated execution is anticipative. Although anticipative executions seems to have no or little interest in practice, we will show in the two applications presented below (sections §6 and §7) that there are particular situations in which they can be used efficiently.

6 Application 1 : Registers in Circuit Design

In this section we will show how the previous results relate to the problem of register minimization in digital circuits. The interest of the relation will be two-fold. In a first part we show how the notions we defined so far help to prove the optimality of retiming in digital circuits. In a second part, we will use the algorithms developed in digital circuits to get optimal regular executions of a system of URE.

6.1 Definition of a circuit

A digital circuit is constituted by functional gates, wires and registers. More precisely,

- A functional element computes an output data from one or several input data. For example, in the case of a logical circuit, the functional elements will be boolean logical gates (AND, OR,...)
- A wire between element i and element j enables to transfer the output data of i which becomes an input data for j.

- A register corresponds to a storage facility. A register of size p (or equivalently p registers) between elements i and j enables to keep in memory the last p values computed by the element i.

The model of the behavior of the system is the following. There is a global clock for the system. Between two clock ticks, here are the operations taking place.

- Functional element :
  1. receive the input data from upstream registers.
  2. compute a new output data.
  3. send the output data to downstream registers.
- Register :
  1. transmit the stored data downstream (to another register or a functional element depending on the structure).
  2. remove the stored data.
  3. receive a new data from upstream (from another register or a functional element depending on the structure).

Between two clock ticks, these operations are synchronously performed at *all* functional elements and registers[1].

Let $X_i(n)$ be the n-th variable computed at element i. After n clock ticks, exactly n values have been computed at each element i, i.e. the variables $\{X_i(m), m \leqslant n\}$ have been computed. The number of registers on a wire between i and j corresponds to the number of variables $X_i(n-k)$ which need to be still in the memory in order to carry on the computation of the variables $X_j(n+k)$.
It appears from the previous description that a digital circuit can be viewed as the reduced graph $\mathcal{R}$ of some system of URE. The functional elements of the circuit correspond to the nodes of $\mathcal{R}$, the wires to the arcs and the registers to the delays. The computation operation corresponding to the functional element i is denoted by F_i to be coherent with previous notations. In the remainder of the section, we will use indifferently the terminology of digital circuits and the one of reduced graphs.

We have represented an example of a digital circuit in Figure 15. We have represented the flow of data between clock ticks. We have chosen on purpose a graphical convention coherent with the one of reduced graphs.

[1] In particular, we do not consider systems where the computations times might be different from one element to the other.

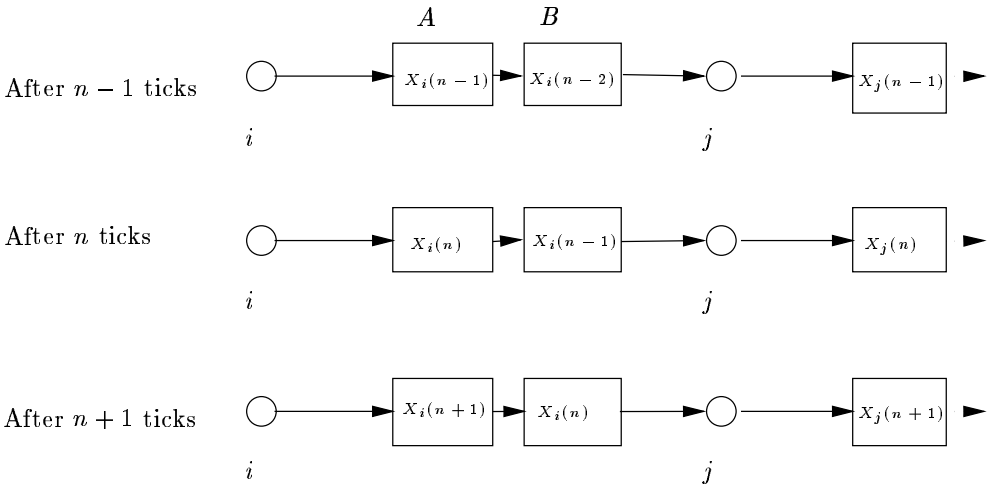


Figure 15: Digital circuit computing $X_j(n) = F_j(X_i(n-2), \dots)$.

Remark 6.1. It might be interesting to consider that the different operations described above have a duration, let us say 1 unit of time for a computation and instantaneous for a transmission-reception. According to this, the elementary operations occurring between two ticks have a total execution time. It is 1 for the system $X_j(n) = X_i(n-\gamma), \gamma > 0$ and 2 for the system $X_j(n) = X_i(n)$ (both variables are computed successively during the same clock interval. Hence the length of the time interval between two clock ticks has to be at least 1 in the first case and at least 2 in the second one. More generally the time interval between two clock ticks has to be equal (at least) to the length (i.e. the number of nodes) of the longest path without registers in the graph $\mathcal{R}$. Hence it is often a problem of practical interest to minimize the longest path without registers, see [13] [3]. We will not consider this problem in the following. We consider the problem called min-area in [13]. It consists in minimizing the number of registers.

6.2 Counting the registers

The two modes for counting delays (see §5.2) are interesting from a practical point of view, when delays are viewed as registers in digital circuits. In order to explain it, we are going to focus on the example of Figure 13. Let us compare the characteristics of the three digital circuits, A, B and $\tilde{B}$, proposed in Figure 13. In circuit (A),

there is a synchronous *write* operation (also called fanout) performed by node i when displaying its output data to downstream registers. In circuit $(\tilde{B})$, there is a synchronous *read* operation performed by the nodes j_2 and j_3 when they get the variable stored in the first register of node i. According to physical and technological constraints, it might be better to avoid either synchronous read or synchronous write, hence to prefer either circuit (A) or $(\tilde{B})$.
Even if we assume that synchronous read has to be avoided (as it is often the case for digital circuits), we might be interested in considering circuit (B) instead of circuit (A). In circuit (B), we have less registers but more functional gates. Hence depending on the compared cost of a node and a register, one shall consider one circuit or the other.

6.3 Minimizing the registers

A classical problem in circuit design is to minimize the number of registers used while preserving the functional behavior of the circuit (i.e. while computing the same variables $X_i(n)$), see the seminal paper of Leiserson and Saxe [13]. If the circuit (i.e. the corresponding reduced graph) is recycled, this problem is directly connected with the notions introduced in §4 and §5. It enables us to propose some complements to the results of [13] for the special case of recycled circuits.

6.3.1 Optimality of retiming

In [13], Leiserson and Saxe define a notion of retiming which is exactly the one of Definition 5.1. They restrict their attention to *legal* retimings.

Definition 6.2. *A retiming r is legal if $\mathcal{R}_r$ has only non-negative delays.*

This is a natural restriction as legal retimings are the only one having a physical meaning for circuits (at least apparently, see §6.3.2). They also define register sharing. This corresponds exactly to the transformation from circuit (A) to circuit (B) in Figure 13. Leiserson and Saxe prove that retiming and register sharing preserve the functional behavior of the circuit. Then they propose an algorithm to compute the optimal circuit after retiming and also after retiming and register sharing, see §6.4.

However the question whether other techniques can be used to get a circuit with even fewer registers remains to be answered. Using the results of previous sections, we show that the retiming technique combined with register sharing does in fact minimize the number of registers.

This result comes from the following argument. Let us consider a circuit. We consider the same circuit where we have positioned the registers in an arbitrary way. We assume that the functional behavior is not modified. These registers can be seen as delays in the reduced graph $\mathcal{R}$. If the functional behavior of the circuit is preserved, it means that the delays correspond to a non-anticipative regular configuration in the associated dependence graph $\mathcal{D}$. But in the recycled case, such a configuration is also a compatible cut in $\mathcal{R}$. The lower section of this cut defines a compatible section which is in turn associated with a legal retiming of the circuit. Therefore, the positions of the registers can be obtained from a retiming of the original circuit.

Corollary 6.3. *If we count the number of registers according to mode B, then we can obtain a circuit with a minimal number of registers solely by performing retiming.*

Proof. This is a direct consequence of Lemma 5.4. □

6.3.2 Further modifications of the circuit

If we allow other modifications of the circuit than just register sharing, further improvements on the number of registers can be obtained.

Let us consider the best possible retiming in the original circuit without restricting ourselves to legal retimings (Definition 6.2). It corresponds to a minimal consecutive (but not necessarily compatible) cut in the associated dependence graph $\mathcal{D}$, see §5.4.

It is possible to perform some appropriate modifications to the structure of the circuit to go back to positive delays. The procedure is as follows. For each node i following a negative delay, we track back the paths terminating at node i until the total delay on each path is non-negative. This is always possible for circuits associated with constructive URE. The nodes initiating such paths are duplicated into 2 nodes computing the same function. In the example of Figure 16, we have to track back two paths : $3 \leftarrow 1 \leftarrow 1$ and $3 \leftarrow 1 \leftarrow 4$ and node 1 is dublicated into 1 and $1'$.

If the registers are counted according to mode B, the resulting circuit uses only 4 registers while the best possible number of registers we can get with only legal retimings is 5. Of course, on the other hand, we have to increase the number of functional nodes.

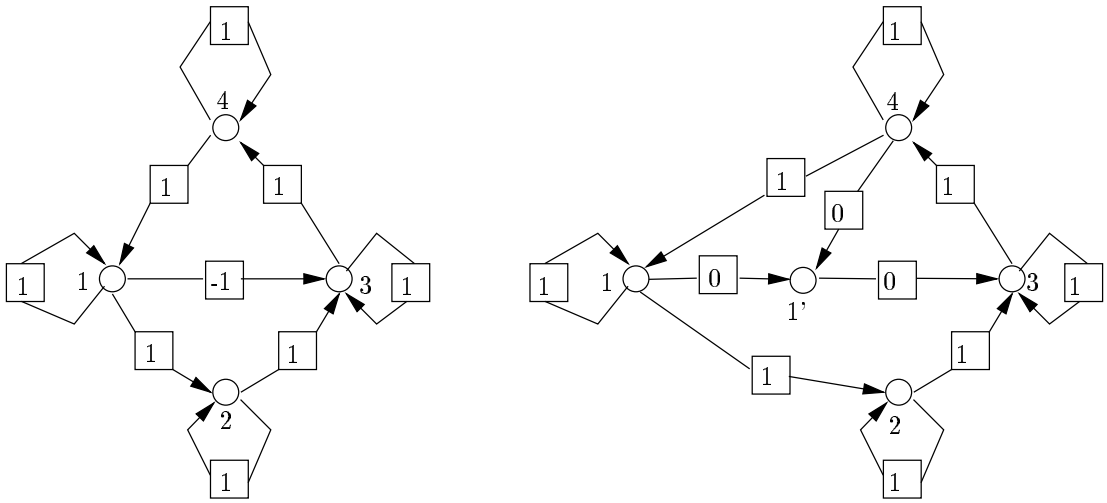


Figure 16: A transformation of a circuit with negative delay into a circuit with non-negative delay.

6.4 Complexity results

In [13], an algorithm is given to compute the best legal retiming of a circuit (with or without using the register sharing technique). The complexity of this algorithm is $O(|E|^2 log|V|)$. An efficient implementation of this algorithm can be found in [17]. In the recycled case, this algorithm can be used to compute the minimal compatible cut in $\mathcal{D}$, using corollary 6.3. And finally using the correspondence between compatible consecutive cuts and non-anticipative regular executions, see Lemma 4.14, this also gives a way to find an optimal non-anticipative regular execution of a system of URE. This is an example of results originally proved for digital circuits and applied in the context of URE.

Further results developed for circuits can be applied in the computation of URE. It is the case of the problem of the minimization of the clock period in digital circuits, see Remark 6.1. This issue is not addressed here.

Conversely, the results of §4.3 (using the Ford-Fulkerson algorithm) can be applied in the context of digital circuit to compute the optimal (non necessarily legal) retiming. This is interesting as it is not straightforward to extend the original algorithm of Leiserson and Saxe to general retimings.

7 Application 2: (max,+) Linear Systems and Parallel Simulation

In the following we will apply our results to a particular class of URE: $(\max, +)$ linear systems, and to issues arising in the distributed simulation of such systems.

Our interest for $(\max, +)$ systems comes originally from the analysis of a class of timed Petri nets: Timed Event Graphs [1]. However, these systems arise naturally in the study of general URE.

To see this, assume that, in the computation of some URE of the form (2), computing $X_i(n) = F_i(X_j(n-\gamma), \dots)$ requires σ_i units of time. Assume moreover that the computation is performed on a parallel computer with an unlimited supply of identical processors, common memory (or instantaneous communication), and no synchronization overhead. If variables are computed as soon as possible (greedy execution), then the *make-span* of the computation is given by an URE with the same structure as (2). Indeed, if $T_i(n)$ is the instant at which the computation of $X_i(n)$ starts, then T_i is given by $T_i(n) = \max(T_j(n-\gamma) + \sigma_j, \dots)$.

We shall discuss below some issues arising in the computation of $(\max, +)$ systems. It should be clear that the results will apply, or can be adapted to other linear recurrences, such as the $(+, \times)$ linear systems of classical control theory.

According to the preliminary remarks above, an application of the results of this section will therefore be an algorithm to compute the make-span of the greedy execution of some URE. Note that this algorithm itself will not be greedy.

We shall first introduce some concepts and notations. We will then present the optimization problem which arises in the parallel computation of $(\max, +)$ URE, and apply the preceding results to solve it. Finally, we shall mention some particularities of stochastic versions of $(\max, +)$ systems.

7.1 Introduction

From now on, we therefore restrict our attention to "Linear Max-Plus Recurrences" (MPR), which are URE of the form:

$$X_i(n) = \max_{(j,\gamma)\in\mathcal{E}_i} (X_j(n-\gamma) + \sigma_{i,j,\gamma}),\ \sigma_{i,j,\gamma} \in \mathbb{R}^+ . \tag{8}$$

The assumption that $\sigma_{i,j,\gamma}$ is nonnegative is not necessary but natural because of the physical interpretation we gave above.

Let us introduce some definitions and notation.

Definition 7.1. *The* (max,+) semi-ring $\mathbb{R}_{\max}$ *is the set* $\mathbb{R} \cup \{-\infty\}$*, equipped with* max*, written additively (*i.e. $a \oplus b = \max(a, b)$*) and the usual sum, written multiplicatively (*i.e. $a \otimes b = a + b$*). The zero element is noted* $\varepsilon = -\infty$*, and the unit element is noted* $e = 0$*.*

For matrices of appropriate sizes, we define $(A \oplus B)_{ij} = A_{ij} \oplus B_{ij} = \max(A_{ij}, B_{ij})$, $(A \otimes B)_{ij} = \bigoplus_k A_{ik} \otimes B_{kj} = \max_k (A_{ik} + B_{kj})$, and for a scalar a, $(a \otimes A)_{ij} = a \otimes A_{ij} = a + A_{ij}$. When no confusion is possible, we abbreviate $A \otimes B$ to AB.

We can rewrite Equation (8) with the previously defined notations. Let $X(n)$ be the column vectors of coordinates $X_i(n)$ and let $A(\gamma)$ be the matrix with coordinates $A(\gamma)_{ij} = \sigma_{i,j,\gamma}$ if $(j, \gamma) \in \mathcal{E}_i$ and $A(\gamma)_{ij} = \varepsilon$ otherwise. We have

$$X(n) = A(0) \otimes X(n) \oplus A(1) \otimes X(n-1) \oplus \cdots \oplus A(\Gamma) \otimes X(n-\Gamma)\,, \tag{9}$$

where Γ is the maximum of the delays appearing in the sets $\mathcal{E}_i$.

This algebraic formulation enables some simple transformations. Let us define

$$A(0)^\star = \bigoplus_{n=0}^{\infty} A(0)^n = \bigoplus_{n=0}^{k} A(0)^n\,,$$

where k is the size of matrix $A(0)$ and where $A(0)^0 = I$ is the identity matrix defined by $I_{ii} = e$ and $I_{ij} = \varepsilon, i \neq j$. It is easy to prove that $A(0)^\star$ is the formal inverse of $I - A(0)$, i.e. $A(0)^\star(I - A(0)) = (I - A(0))A(0)^\star = I$. Hence, Equation (9) can be transformed into:

$$X(n) = A(0)^\star A(1) X(n-1) \oplus \cdots \oplus A(0)^\star A(\Gamma) X(n-\Gamma)\,. \tag{10}$$

Equation (10) is nicer, because it involves only strictly positive delays, and is therefore obviously constructive.

Dependence graph The dependence graph has a general form as presented in §2.1. Its only characteristic is that the functions on the nodes are "max" applied to all entries.

Reduced graph A formalism naturally associated with $(\max, +)$ recurrences it that of *Petri nets*. The graphical formalism of Petri nets is close but different from the one we used for reduced graphs.

Petri nets consist of *transitions*, usually interpreted as service centers (processing units, etc.), and *places* containing *tokens*, usually interpreted as entities (programs, customers...) receiving services from transitions. Places are connected to transitions and transitions to places with directed arcs. It is therefore natural to speak of "input places", "output places" and so on. The passage from the reduced graph associated with a MPR to the corresponding Petri net consists in replacing nodes with transitions, and arcs with delay δ with a place containing δ tokens, connected to the corresponding transitions. Values of the delay therefore correspond to positions of tokens, called *markings*.

Figure 17 shows such a transformation.

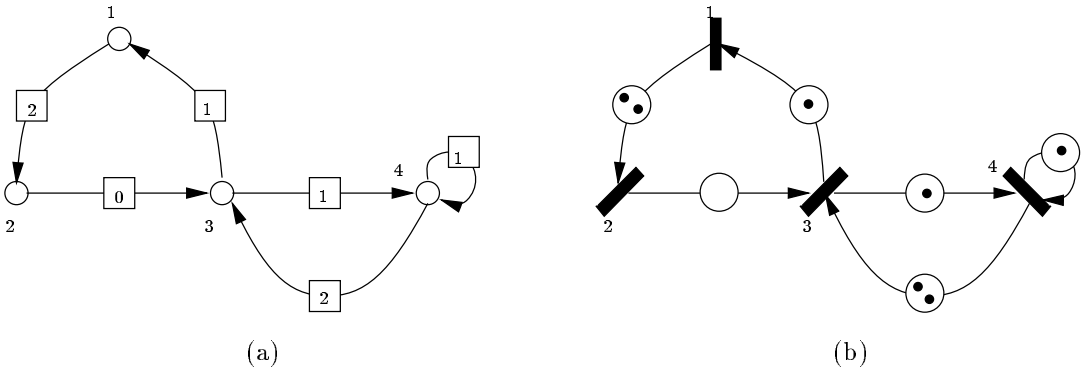


Figure 17: Transformation of a reduced graph (a) into a Petri net (b).

The Petri nets corresponding to reduced graphs have the particular property that places have exactly one input transition and one output transition. This property defines the class of *Event Graphs* (EG).

Petri nets are dynamical systems, in which tokens may move, according to the following rule. Transition may *fire*, thus removing one token from every input places and creating one in every output place. A fundamental remark is that firings in a Petri net are equivalent to retimings in reduced graphs (see §5.1). Indeed, the equations describing the transformations of the marking after a firing are precisely of the form $\gamma' = \gamma + r(i) - r(j)$.

It is important to note that the usual convention for Petri nets is that a firing may occur only when at least one token is present in every input place of the transition.

This requirement is dropped here. We therefore allow *negative markings*, which correspond to the negative delays of §5.

In *timed* event graphs, durations are associated with firings. Linear $(\max,+)$ systems of the form (8) describe the evolution of the associated event graphs in the following way. The variable $X_i(n)$ represents the instant at which the n-th firing of transition i starts, given that transitions start firing *as soon as possible*, that is, as soon as all tokens necessary are present in the input places *and* available, *i.e.* not involved in another firing. This interpretation holds under the assumption that the system is recycled, because the fact that tokens are involved in at most one firing implies that tokens go through transitions in a *first-in-first-out* (FIFO) order. Therefore, there are no overtaking of tokens, and the n-th firing of transition i requires the $(n-\gamma_{j,i})$-th token produced by transition j.

7.2 MPR of order 1

A standard step in the analysis of linear systems is the transformation of recurrences of order Γ such as (10) into an "equivalent" system of order 1. For general URE, this operation consists in introducing new variables $X_{k+1},\dots,X_\ell$ and new functions $G_i, i\in\{1,\dots,\ell\}$ such that

$$X_i(n) = G_i(X_j(n-\gamma)), 1\leqslant i\leqslant \ell,\ (j,\gamma)\in\mathcal{E}_i,\ n\in\mathbb{N}\ ,\quad \gamma\leqslant 1\ . \tag{11}$$

This is usually done by setting $X_{(i-1)\times\Gamma+\gamma}(n) = X_i(n-\gamma)$, for $1\leqslant i\leqslant k$ and $1\leqslant\gamma\leqslant\Gamma$. The new number of variables is therefore $\ell = k\Gamma$.

In the case of $(\max,+)$ linear systems, the equivalent system of order 1 is characterized by a matrix of size $\ell\times\ell$. The recurrence becomes

$$X(n+1) \ = \ A(n)\otimes X(n)\ . \tag{12}$$

In some practical applications, it may be desirable to reduce this size as much as possible. An instance of such applications is described in the following section.

7.3 Parallel simulation of time varying MPR

For the purposes of this section, we informally introduce a generalization of the URE model (2), in which the functions F_i may additionally depend on n. The particular example we have in mind is that of MPR of the form (8) in which the numbers $\sigma_{i,j,\gamma}$ are allowed to depend on n. In the analysis of discrete event systems, these sequences are commonly assumed to be random.

Consider therefore an URE defined with a sequence of (possibly random) functions :

$$\{(F_1^n, \ldots, F_k^n),\ n \in \mathbb{N}\}\ .$$

We consider the associated dependence graph $\mathcal{D}$ and the pebble game under the set of rules $\mathcal{M}_3$. Executions under this rule are regular, and therefore characterized by a finite set $\mathcal{A}(0) \subset \{1, \ldots, k\} \times \mathbb{Z}^-$ providing the position of the pebbles at step 0. Let $X(n), n \in \mathbb{N}$, be the vector whose coordinates are $X_i(p+n)$ for $(i,p) \in \mathcal{A}(0)$. By definition of rule $\mathcal{M}_3$, see §3.2, we have that there exists a sequence of functions $\{\phi^n, n \in \mathbb{N}\}$ such that:

$$X(n+1) = \phi^n(X(n)) = \phi^n \circ \phi^{n-1} \circ \cdots \circ \phi^0(X(0))\ . \tag{13}$$

The *simulation* of the system consists in computing the value of all $X(n)$. A possible algorithm for doing this with a parallel computer uses the so-called parallel prefix principle. Using the fact that the composition of functions is associative, it is possible to divide the computation of $\phi^n \circ \cdots \circ \phi^0$ in smaller products $\phi^p \circ \cdots \circ \phi^q$ which may be computed by different processors. Note that for this, it is necessary that the operators ϕ possess a numerical representation on which composition may be performed. This is typically the case for linear operators, which are represented by matrices, for which composition is equivalent to the common product. The parallel prefix algorithm therefore directly applies to the parallel simulation of MPR (12).

A way to quantify the efficiency of the parallel algorithm is to evaluate its PRAM complexity. It can be shown that the number of operations required to simulate the linear system up to time N with P processors is $\mathcal{O}(\ell^3(N/P + \log(P)))$, where ℓ is, as above, the size of the matrix characterizing the linear system.

In order to minimize the complexity of this algorithm, it is therefore necessary to find a representation of the MPR of minimal size.

7.4 Optimization results

We shall show in this section that finding the minimum possible size for an order 1 representation of a MPR can be done in polynomial time with respect to k and Γ. This problem appears to be new in the context of event graphs. Some preliminary results, which correspond to our mode A for counting memories may be found in [2].

The basic idea is that, given a marking of the event graph, it is possible to transform this graph by adding new transitions and places, in such a way that the resulting event graph, restricted to the original transitions, has the same dynamic

behavior as the original one, and moreover, the marking of the places is less than one (see [9] for further discussion).

To see this, recall the discussion of §5.2 on counting the delays, and in particular Figure 13 (B). Interpreted in terms of Petri nets, this construction amounts to "factor out" tokens introducing dummy transitions in a tree-like fashion, as in Figure 18. The dummy transitions are assumed to have a firing time of 0, and are recycled (this is not shown on the figure).

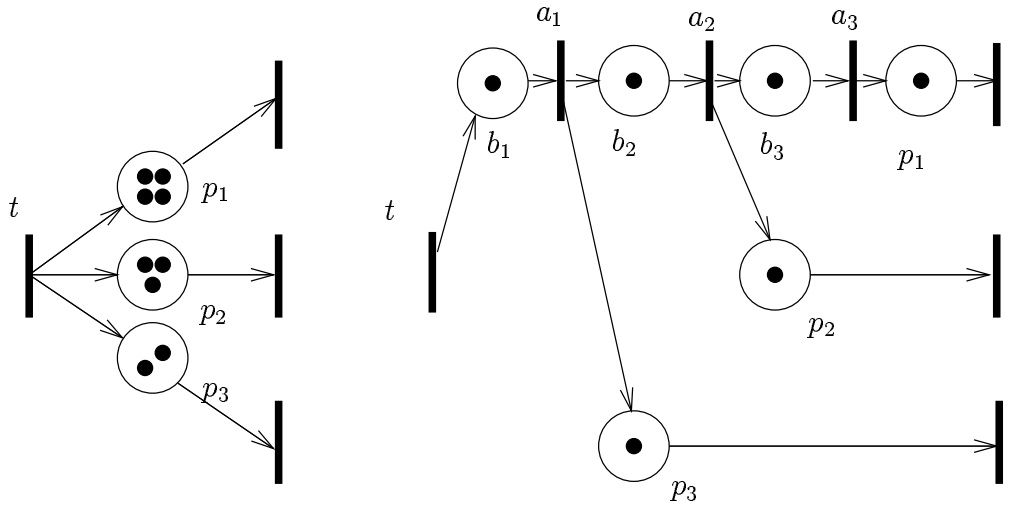


Figure 18: Forward transformation of event graphs.

The number of transitions in the resulting event graph is Γ_B. The MPR associated with the new graph has a maximum delay $\Gamma = 1$ and by (10), it has the desired order 1 form with $\ell = \Gamma_B$

This is already an improvement on the standard representation, but the results of §5 allow to improve this, by finding first an optimal marking of the net, that is, a marking such that the above transformation provides an event graph with $\ell = \min_{\mathcal{RE}} \Gamma_B$ or $\ell = \min_{\mathcal{NRE}} \Gamma_B$ transitions.

It is indeed necessary to distinguish the two cases, according to whether negative markings are desirable or not. This can be understood as follows.

Assume that the marking corresponding to $\ell = \min_{\mathcal{RE}} \Gamma_B$ is negative. The event graph can be transformed into another equivalent one in the same way as for circuits

in 6.3.2. The newly created transitions will then have firing times $\sigma_i(n)$ *equal* to some of the $\sigma_j(n+\gamma), \gamma > 0$ of the original transitions. Therefore, the construction of matrices $A(n)$ and $A(n+\gamma)$ in recurrence (12) will use the same numbers σ. When two matrices do not use the same numbers, they are called *disjoint* matrices

In the parallel computing context, where the computations using $A(n)$ and $A(n+\gamma)$ are (possibly) done by different processors, it may be acceptable to use matrices which are not disjoint. For instance if $\sigma_i(n)$ does not depend on n, or if it can be computed in a deterministic way by the different processors.

However, it is not acceptable if the variables $\sigma_i(n)$ are *independent and identically distributed* (i.i.d.). In this case, we want the random matrices $A(n)$ and $A(n+\gamma)$ to be independently generated by their respective processor. This situation is the most common one in the context of discrete event system modeling, which we have already mentioned. It requires that all the matrices are disjoint.

In both cases, the optimal marking is found in polynomial time:

- If negative markings are acceptable, use the results of §4.3 to find a minimal consecutive cut.
- If not, use the algorithm of §6.4 to find a minimum compatible cut.

Remark 7.2. The optimality of the size of representation should be understood as the best possible that can be obtained without making assumptions on the value of the numbers $\sigma_{i,j,\gamma}(n)$. When these numbers are constant and known, this knowledge may be exploited to obtain a minimal representation in the sense of linear system theory [8], which is normally better than ours. A deeper investigation of the relations between the two approaches is an interesting direction for further research.

7.5 Backward transformation

To conclude the section, we make the remark that there are actually two ways to perform graph transformations : the *forward* transformation on downstream places as above (Figure 18), or the *backward* transformation on upstream places (Figure 19).

Remark 7.3. It is important to note that the backward transformation is not possible for a general system of URE. A system of URE associated with the example of Figure 19 (before transformation) is of the form :

$$X_4(n) = F_4(X_1(n-2), X_2(n-3), X_3(n-4)) .$$

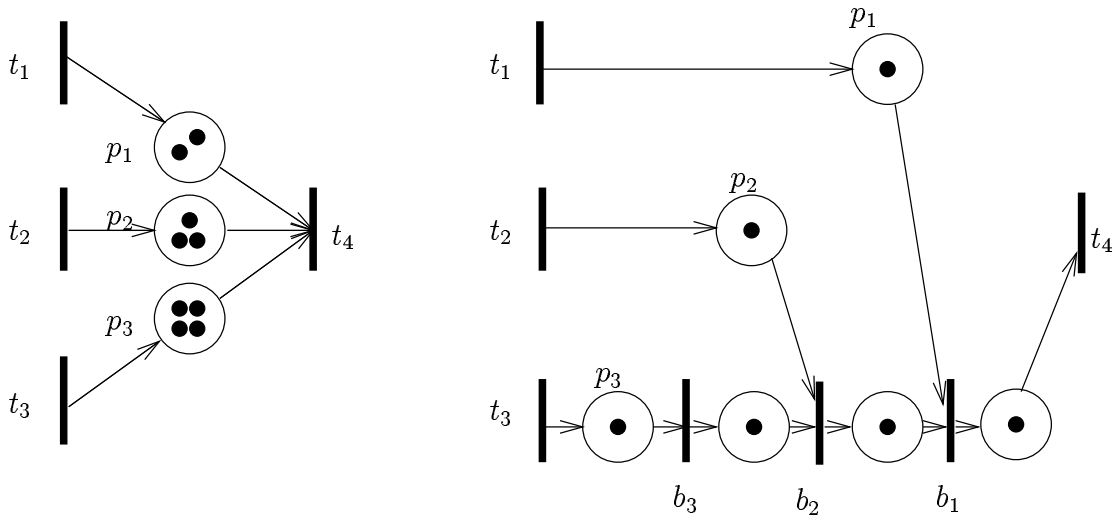


Figure 19: Backward transformation of event graphs.

If we perform the backward transformation, the form of an URE associated with the new graph has to be :

$$X_4(n) = F(\ G(\ X_1(n-2), H(\ X_2(n-3), I(\ X_3(n-3)\ )\ )\ )\ ),$$

for some functions F, G, H and I associated with nodes t_4, b_1, b_2 and b_3 respectively. In general it is not possible to perform such a factorization of the original F_4 function. It becomes possible in MPR because of the special form of the functions F_i which are involved.

In the context of event graphs, the backward transformation is interesting since it results in a graph which might be smaller than the one obtained with the forward transformation.

The example displayed in Figure 20 has the following property. If we apply a forward transformation (after optimal legal retiming) the number of transitions is 7. However, a backward transformation yields a graph with only 6 transitions. It is also interesting to note that in this example, the size of the minimal (non compatible) cut in the dependence graph is 5. Therefore, this is an example where one

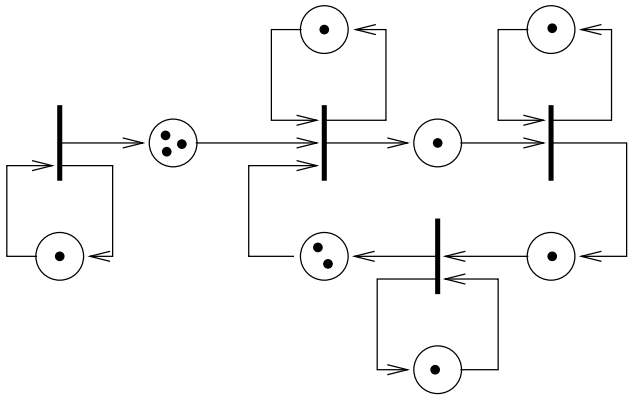


Figure 20: Optimal forward and backward transformations of this event graph yield to systems of different size.

can find an intermediate system of size 6, (strictly between the size of the smallest compatible cut and the size of the smallest cut) which allows a representation of the original system with disjoint matrices of size 6.

Determining an optimal backward transformation can be done using the previous results. In fact, it corresponds to the optimal forward transformation on the *reversed* event graph obtained by reversing the direction of all the arcs.

More generally, it is easy to come up with examples where the optimal transformation of an event graph involves both forward and backward transformations. Finding an algorithm to compute such an optimal mixed transformation is an interesting open problem.

7.6 Stochastic issues

The retimings used above for deriving optimal representations necessitate changes in the initial condition and shifts in the indices of the sequences $\{\sigma_{i,j,\gamma}(n)\}$.

When these sequences are random, these changes may be unnecessary, depending on the performances that are measured on the system.

Indeed, it is proved in [14] that *stationary* statistics of the MPR (such as asymptotic growth rate, and limit distributions for finite differences) are insensitive to the initial conditions under minimal stochastic assumptions on the sequences.

References

[1] F. Baccelli, G. Cohen, G.J. Olsder, and J.P. Quadrat. *Synchronization and Linearity.* John Wiley & Sons, New York, 1992.

[2] H. Braker. *Algorithms and Applications in Timed Discrete Event Systems.* PhD thesis, Delft Univ. of Technology, 1993.

[3] M. Canales and B. Gaujal. Marking optimization and parallelism in marked graphs. Technical Report 2049, INRIA, Sophia Antipolis, France, 1993.

[4] P. Chretienne. *Les Réseaux de Petri Temporisés.* PhD thesis, Université Paris VI, Paris, 1983.

[5] A. Darte, L. Khachiyan, and Y. Robert. Linear scheduling is nearly optimal. *Parallel Processing Letters*, 1(2):73–81, 1991.

[6] A. Darte and F. Vivien. Automatic parallelization based on multi-dimensional scheduling. Technical Report 24, LIP, ENS Lyon, September 1994.

[7] L.R. Ford and D.R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.

[8] S. Gaubert. *Théorie des systèmes linéaires dans les dioïdes.* Phd thesis, École des Mines, Paris, 1992.

[9] B. Gaujal. *Parallélisme et simulation des systèmes à événements discrets.* PhD thesis, University of Nice-Sophia Antipolis, June 1994.

[10] M. Gondran and M. Minoux. *Graphes et algorithmes.* Eyrolles, Paris, 1979. Engl. transl. *Graphs and Algorithms*, Wiley, 1986.

[11] C. Hanen and A. Munier. A study of the cyclic scheduling problem on parallel processors. *Discrete Appl. Math.*, 57:167–192, 1995.

[12] R. Karp, R. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *J. Ass. Comp. Mach.*, 14(3):563–590, 1967.

[13] C. Leiserson and J. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6:5–35, 1991.

[14] J. Mairesse. *Stabilité des systèmes à événements discrets stochastiques.* PhD thesis, Ecole Polytechnique, Paris, 1995.

[15] N. Pippenger. Pebbling. Research report RC 8258 (#35937), IBM Research Division, 1980.

[16] R. Sethi. Complete register allocation problems. *SIAM J. Comp.*, 4(3):226–248, 1975.

[17] N. Sheney and R. Rudell. Efficient implementation of retiming. 1994.

[18] G.X. Viennot. Trees. In M. Lothaire, editor, *Mots*, pages 265–297, Paris, 1990. Hermès.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399