Computations of Uniform Recurrence Equations Using Minimal Memory Size*

Bruno Gaujal†, Alain Jean-Marie‡ and Jean Mairesse§
(gaujal@loria.fr,ajm@sophia.inria.fr,mairesse@liafa.jussieu.fr)

January 6, 2000

Abstract

We consider a system of uniform recurrence equations (URE) of dimension one. We show how its computation can be carried out using minimal memory size with several synchronous processors. This result is then applied to register minimization for digital circuits and parallel computation of task graphs.

Keywords uniform recurrence equation, register minimization, circuit design, task graph, (max,+) linear system.

*Partially supported by the European Grant BRA-QMIPS of CEC DG XIII.
†LORIA/INRIA Lorraine, 615, rue du jardin botanique, B.P.101, 54602 Villers-les-Nancy Cedex, France
‡INRIA Sophia-Antipolis, B.P. 93, 06902 Sophia Antipolis Cedex, France
§LIAFA, CNRS-Université Paris 7, Case 7014, 2 place Jussieu, 75251 Paris Cedex 05, France.

1 Introduction

Definition 1.1 (URE). *We consider $\mathcal{Q}$-valued variables $X_i(n), i \in V, n \in K$, where $\mathcal{Q}$ is an arbitrary set, V a finite set, and $K \subset \mathbb{Z}^p$ for some $p \in \mathbb{N}$. These variables satisfy the equations*

$$X_i(n) = F_i\Big(X_j(n-\gamma), (j,\gamma) \in \Delta_i\Big), \forall n \in K \, . \tag{1}$$

The sets Δ_i, called dependence sets, *are finite subsets of $V \times \mathbb{Z}^p$. The collection of Equations* (1) *is called a set of* Uniform Recurrence Equations.

There is no restriction on the generality of the functions F_i except the fact that they are computable. The system $\mathcal{S}$ defined by Equation (1) is said to be uniform because the dependence sets Δ_i do not depend on n. The integers γ are called the *delays*. It is possible to have two delays $\gamma, \gamma' \in \mathbb{Z}^p, \gamma \neq \gamma'$ such that $(j,\gamma) \in \Delta_i$ and $(j,\gamma') \in \Delta_i$.

There are various motivations to study URE. They appear in the description of differential equations using finite difference methods or in the study of discrete event systems. The case $p > 1$ and $K = \mathbb{Z}^p$ has often been studied in the literature, see [16]. In this case, some of the major issues are the constructivity [16] and loop parallelization [8]. These problems and others appearing in this framework will be discussed in § A.

In this paper, we consider only the simple case where $K = \mathbb{Z}$ (systems of dimension one). The computational model considered is that of parallel processors with a shared memory (CREW-PRAM model: *Concurrent Read Exclusive Write-Parallel Random Access Memory*). More precisely, a computation is performed by a processor, using data stored in the memory. For example, to compute $X_i(n)$, it is necessary to have at least $|\Delta_i|$ memory locations, each location containing one of the data $\{X_j(n-\gamma), (j,\gamma) \in \Delta_i\}$. In a model of parallel processors with shared memory, there are several processors which can make computations simultaneously and also access the same memory locations simultaneously.
The problem investigated consists in minimizing the "memory size":

> *What is the minimal number of memory locations that is needed to compute all the variables $X_i(n)$ of Equation (1) using a CREW-PRAM computational model?*

We solve this problem in the recycled case (see Section 2.2 for the definition) by proving that it is equivalent to the search for minimal cuts in the dependence graph associated with the system of URE. This provides polynomial algorithms to compute the minimal memory requirements.
We show that the solution of this problem has many applications. Indeed, URE appear in the modeling of logical circuits, systolic arrays or program loops. Our result can be used practically for the optimization of circuit design. Given a digital circuit, we show how to find another circuit with the same functional behavior and using a minimal number of registers. This application will be discussed in § 7.
Our results can also be used in another context, namely, in order to obtain the most efficient representation of task graph systems for parallel computation purposes. The evolution of a task graph can be represented as a linear system over the (max,+) algebra of the form $x(n+1) = A(n)x(n)$, where $x(.) \in \mathbb{R}^k_{\max}$ and $A(n) \in \mathbb{R}^{k\times k}_{\max}$. Our results enable us to obtain a

linear representation of a task graph with a minimal dimension k for the matrices $A(n)$. This application will be treated in § 8.

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. In particular, we restrict our attention to recycled systems of URE. Sections 4 and 5 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 § 6 presents the interpretation of the above quantities in the reduced graph. Finally in Sections 7 and 8, two applications are described, for digital circuits and (max,+) linear systems respectively. In Appendix § A and § B, we consider the related problems of scheduling and sequential executions.

2 Basic Models

From now on, we consider URE of dimension 1. More precisely, we consider the set of variables $X_i(n), i \in V, n \in \mathbb{Z}$ and the equations

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

where the sets Δ_i are finite subsets of $V \times \mathbb{Z}$.

A system of URE is *constructive* if given the values of the "negative" variables $X_i(n), n \leqslant 0$ (*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 $X_i(n)$ can be computed before $X_i(n)$. Equivalently, the constructivity assumption can be written as follows:
For each cycle $(i_1,\gamma_1),\ldots,(i_p,\gamma_p), i_{p+1} = i_1$ such that $(i_{j+1},\gamma_{j+1}) \in \Delta_{i_j},\ j \in \{1,\ldots,p\}$ then $\sum_{j=1}^{p} \gamma_j > 0$.

Remark 2.1. Under the constructivity assumption, Farkas Lemma states that it is possible to come back to Equation (1) with all the sets Δ_i included in $V \times \mathbb{N}$, using a simple renumbering of the variables (i.e. $X_i(n) := X_i(n + c_i)$ for some constants $c_i \in \mathbb{Z}$ independent of n). This renumbering actually amounts to a retiming of the system. This notion will be studied in details in Section 6.1.

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

Example 2.2. The illustrative examples in this section 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 2.3 (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 $(i, m-n) \in \Delta_j$ (notation: $(i,n) \to (j,m)$).*

The n-th *column* in $\mathcal{D}$ is the set of nodes $\{(i,n), i \in V\}$. The i-th *line* 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 $(i,n),\ n \geqslant 0$, as *positive* nodes.
It is immediate from the definition of an URE that $\mathcal{D}$ is 1-periodic, i.e.

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

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 2.2.

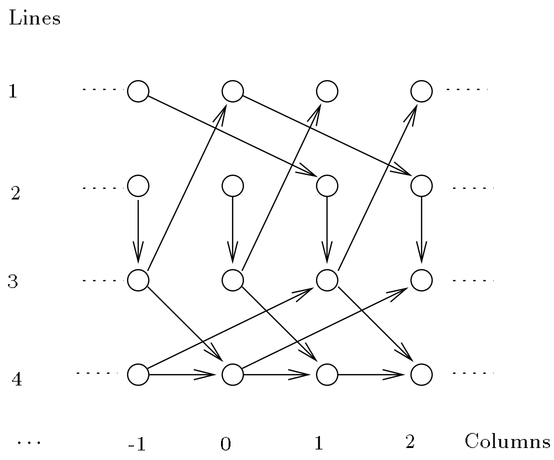


Figure 1: Dependence graph associated with the system $\mathcal{S}$ of Equation (3).

The dependence graph appears under various forms and names in the literature, for example: developed graph, PERT graph, unfolded process graph or activity network.

2.2 Reduced graph

Since the dependence graph $\mathcal{D}$ is 1-periodic, it can be folded into a *reduced graph* $\mathcal{R}$.

Definition 2.4 (Reduced graph).
The reduced graph is an arc valued graph $\mathcal{R} = (V, E, \Gamma)$. The set of nodes is V and there is an oriented arc $e \in E$ from i to j if

$$\exists \gamma \in \mathbb{Z} \;\; s.t. \;\; (i,\gamma) \in \Delta_j\,. \tag{4}$$

This arc is valued with the delay *$\Gamma(e) = \gamma$. If there exist several delays γ verifying condition (4), E contains several arcs between the nodes i and j, with corresponding values. Furthermore, we consider the functions $F_i, i \in V$, to be associated with the nodes of $\mathcal{R}$.*

There is an arc from i to j in E, if and only if there are arcs from the line $(i,.)$ to the line $(j,.)$ in the dependence graph. The system $\mathcal{S}$ is constructive if and only if the sum of the delays along any circuit in $\mathcal{R}$ is strictly positive.
The reduced graph associated with the system $\mathcal{S}$ of Example 2.2 is represented in Figure 2. The delays γ associated with the arcs are depicted in boxes.

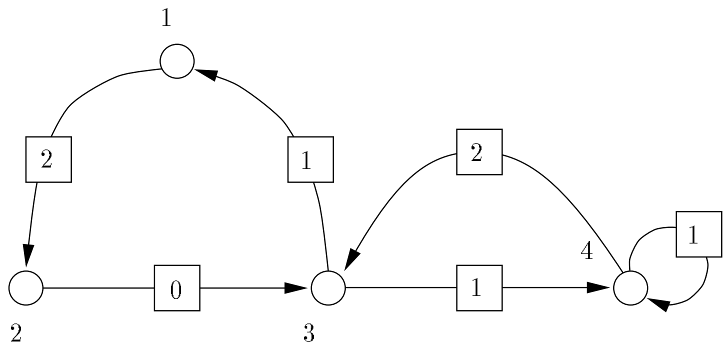


Figure 2: Reduced graph associated with the system $\mathcal{S}$ of Equation (3).

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

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.

2.3 Recycled assumption

In the following (Sections § 4,5,7 and 8), we will only 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 task graphs (see § 8) 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 $(i,1) \in \Delta_i, \forall i$, for the system of URE. Equivalently, it results in having a self loop with delay one (hence the name recycled) at each node of $\mathcal{R}$, or in having arcs between the nodes (i,n) and $(i,n+1)$ in $\mathcal{D}$. Such arcs will be called *recycling arcs* in the sequel. Figure 3 depicts an example of a recycled system.

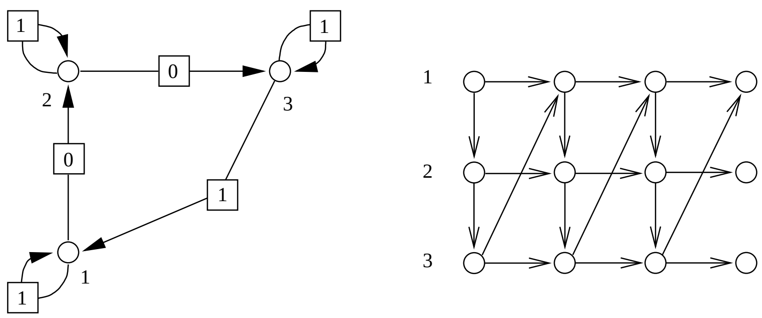


Figure 3: Recycled reduced graph and recycled dependence graph.

2.4 Connectedness

We say that a system of URE is (strongly) connected if the graph $\mathcal{R}$ is (strongly) connected. In the remainder of the paper, we will always consider systems of connected (but not necessarily strongly connected) URE.

In fact, we will see that, for a recycled connected URE and for games $\mathcal{M}_1, \mathcal{M}_2$ and $\mathcal{M}_3$ (to be defined below), a valid computation with a minimal number of memory locations requires all its memories at each instant. It implies that the minimal number of memories necessary to compute a non-connected recycled URE is the sum of the minimal numbers of memories necessary to compute the different connected components independently.

3 Synchronous Executions

We want to minimize the memory size required in the synchronous computation of a system $\mathcal{S}$. Among the related problems that have been studied in the field of URE, we can mention the *basic scheduling* problem and the *sequential* computations. These questions and their relation with the one considered in the paper are discussed in the Appendices § A and § B.

3.1 Pebble games

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

What is 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*. A pebble game is played on a graph. At each step, a finite number of pebbles are located on the nodes of the graph, with at most one pebble per node. The position of the pebbles evolves by adding or removing pebbles according to some rules.

Different variants of pebble games have been used in the literature to model memory allocation problems, see for example [21, 25]. A pebble corresponds to a memory location and puting a pebble on a node corresponds to the computation of the variable associated with the node and its storage into the memory. Removing a pebble from a node corresponds to erasing this data from the memory.

Now, we give a more formal definition of the pebble game in our framework. Here, the graph considered is the dependence graph $\mathcal{D}$ of an URE.

Definition 3.1 (configuration). *A configuration is a finite subset of $V \times \mathbb{Z}$, the set of nodes of $\mathcal{D}$. A configuration represents the position of the pebbles at some stage of the game. There is at most one pebble per node.*

Definition 3.2 (execution, successful execution, step). *An* execution *of the pebble game is a sequence of configurations $e = \{\mathcal{A}(t), t \in \mathbb{N}\}$, such that for all t, the configuration $\mathcal{A}(t+1)$ can be obtained from $\mathcal{A}(t)$ through the rules of the pebble game. The passage from $\mathcal{A}(t-1)$ to*

$\mathcal{A}(t)$ *is called the* step t *of the game. An execution of the game is* successful *if all positive nodes receive a pebble during the execution, i.e. for all* $(i,n) \in V \times \mathbb{N}$, $\exists\, t \in \mathbb{N}$, *s.t.* $(i,n) \in \mathcal{A}(t)$.

In the following, we will always consider successful executions and refer to them simply as executions. An execution corresponds to a computation of all the variables $\{X_i(n), i \in V, n \in \mathbb{N}\}$. The number of pebbles used by an execution $e = \{\mathcal{A}(t), t \in \mathbb{N}\}$ is:

$$\mathcal{P}(e) \stackrel{\text{def}}{=} \sup_{t \in \mathbb{N}} |\mathcal{A}(t)|\,, \tag{5}$$

where $|\mathcal{A}(t)|$ represents the cardinal of $\mathcal{A}(t)$. Our general objective is redefined below. It will be referred to as the Problem *MinPeb*.

Problem 1 (MinPeb). *Determine* $\min_e \mathcal{P}(e)$ *and an execution* e_o *such that* $\mathcal{P}(e_o) = \min_e \mathcal{P}(e)$.

In the following, we define several sets of rules, each of them defining a different pebble game. The different sets of rules, called $\mathcal{M}_1, \mathcal{M}_2$ and $\mathcal{M}_3$, correspond to different computation models for the URE and are related to different notions of cuts and delays (see § 4 and § 5). Also, their relevance will be justified by the applications given in § 7 and § 8. We use the expressions 'set of rules $\mathcal{M}_i$' or 'game $\mathcal{M}_i$' indifferently.

$\mathcal{M}_1$ **: Synchronous Execution.** The set of rules $\mathcal{M}_1$ is:

- (R1) *(starting rule)* Initially, a finite number of pebbles are put on negative nodes only, with at least one pebble on column 0: $\mathcal{A}(0) \subset V \times \mathbb{Z}^-$, $\quad \mathcal{A}(0) \cap (V \times \{0\}) \neq \emptyset$;
- (R2) *(playing rule)* one step of the game consists in any number of moves of type (R3), followed by any number of moves of type (R4);
- (R3) *(adding pebbles)* put a pebble on an empty node (i,n). At step t, this is possible if and only if each infinite oriented path (see Definition 4.1) ending in (i,n) intersects $\mathcal{A}(t-1)$;
- (R4) *(removing pebbles)* remove a pebble from a node.

Remark 3.3. Comments of rule (R2).
Note that our definition of $\mathcal{P}(e)$ considers only the number of pebbles at the end of the step and not in intermediate stages (after (R3) and before (R4) for example). It corresponds to the assumption that all the moves done in one step can be performed simultaneously. This is why this is called a synchronous execution. This remark also applies to games $\mathcal{M}_2$ and $\mathcal{M}_3$.

Remark 3.4. Comments on rule (R3).
Rule (R3) may look cumbersome since one may put a pebble on a node which is very far to the right from the current position of the pebbles. Its intuitive meaning for the calculation in a system of URE is the following one: at the beginning of step t, the variables which are in memory are the ones corresponding to $\mathcal{A}(t-1)$. *A new pebble can be put on a node* (i,n) *if the corresponding variable* $X_i(n)$ *can be computed given the variables in memory.* This does not say that this computation has to be direct. It may be done using the variables in memory and the

appropriate compositions of the initial functions F_i. Since the initial functions are arbitrary, no notion of the "complexity of a function" is used here. Hence the the function obtained by composition of a finite number of initial functions can be considered just as yet another arbitrary function and its computation does not require any additional memory.
However, it seems reasonable to consider that function compositions should have a 'cost', not in terms of space as mentioned above, but in terms of time. A step of the game may have a duration which depends on the "complexity" of the compositions. The discussion of this aspect of the problem is postponed until the Appendix, see § A.2.

We illustrate rule $\mathcal{M}_1$ on the example of Figure 4. We have represented a small part of the dependence graph of the URE: $X_1(n) = F_1(X_1(n-1), X_2(n-1), X_3(n-1)), X_i(n) = F_i(X_{i-1}(n), X_1(n-1), X_2(n-1), X_3(n-1))$ for $i = 2, 3$.

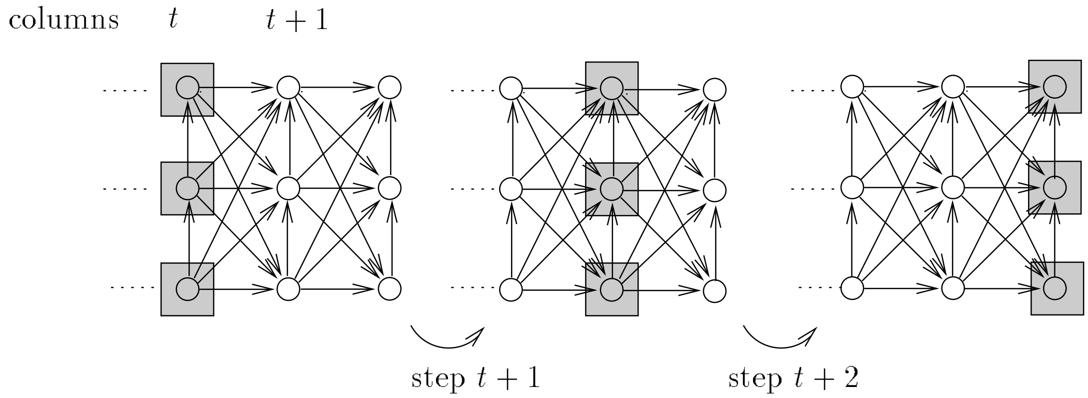


Figure 4: Rule $\mathcal{M}_1$. Three pebbles are needed.

At step 0, we have three pebbles on nodes $(i, 0)$, $i = 1, 2, 3$ (rule (R1)). At step 1, it follows from rule (R3), that a pebble can be put on any positive node. For instance, let us consider node $(2, 1)$. The associated variable can be computed as follows:

$$X_2(1) = F_2\big(F_1\left(X_1(0), X_2(0), X_3(0)\right), X_1(0), X_2(0), X_3(0)\big).$$

By keeping the original pebbles untouched, we can use one additional pebble to mark all the nodes one by one. In this way, we obtain a succesful execution using four pebbles. It is however possible to do better.
Consider the following execution, illustrated in Figure 4. After step $t-1$, assume there are three pebbles on nodes $(i, t-1)$, $i = 1, 2, 3$. At step t, we can put simultaneously three pebbles on nodes (i, t) and we remove the initial pebbles (rule (R3) used three times followed by rule (R4) applied three times). At step $t+1$, we put the three pebbles on nodes $(i, t+1)$ and so on. The number of pebbles needed by this execution is three.

Game $\mathcal{M}_1$ can be seen as a model of computation of an URE where several synchronous processors are used in parallel during the computations. These processors can access the same memory locations at the same time. More precisely, this is a model of a CREW-PRAM (Concurrent Read Exclusive Write-Parallel Random Access Memory, see for instance Reif [22]) computation of the URE. The number of processors needed at one step is equal to the number of moves of type (R3) (i.e. the number of computations realized). For more details, see § A.2

$\mathcal{M}_2$ **: Synchronous Regular Execution.** The rules of $\mathcal{M}_2$ are obtained by restricting $\mathcal{M}_1$ as follows.

- (R1) Unchanged;
- (R2b) *(playing rule)* same as before with the additional restriction that configurations must be 1-periodic, i.e. $\mathcal{A}(t+1) = \mathcal{A}(t) + 1$, where

$$(i, n) \in \mathcal{A}(t) + 1 \iff (i, n-1) \in \mathcal{A}(t) ; \qquad (6)$$

- (R3) unchanged;
- (R4) unchanged.

The example of Figure 4 was also verifying the set of rules $\mathcal{M}_2$. To see that $\mathcal{M}_1$ and $\mathcal{M}_2$ are indeed different, let us consider the example of Figure 5. It corresponds to the URE $X_1(n) = F_1(X_2(n-1)), X_2(n) = F_2(X_1(n))$.

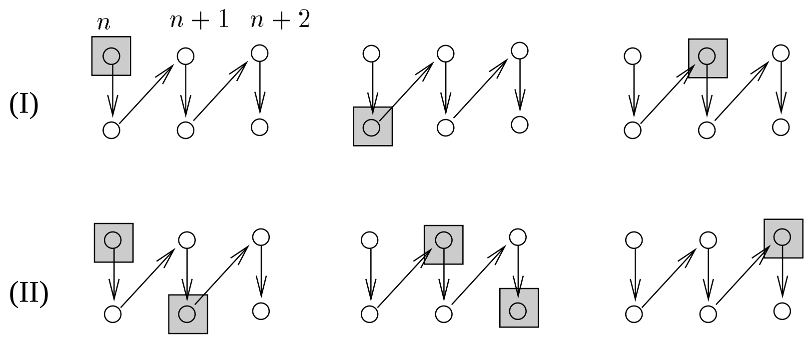


Figure 5: Rule $\mathcal{M}_1$ (I) and rule $\mathcal{M}_2$ (II).

In Figure 5 (I), only one pebble is needed under rule $\mathcal{M}_1$. The corresponding execution verifies rule (R2) (game $\mathcal{M}_1$) but not rule (R2b) (game $\mathcal{M}_2$). In Figure 5 (II), two pebbles are needed. The corresponding execution verifies rule (R2b). The computations are performed according to the following patterns :

a. Rule $\mathcal{M}_1$ (Figure 5 (I)).

- step t : $X_2(n) = F_2(X_1(n))$;
- step $t+1$: $X_1(n+1) = F_1(X_2(n))$;
- step $t+2$: $X_2(n+1) = F_2(X_1(n+1)) \ldots$

b. Rule $\mathcal{M}_2$ (Figure 5 (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 $\mathcal{M}_2$, we have to perform the function compositions $F_2 \circ F_1$ and $F_1 \circ F_2$. In § A.2, we discuss the 'cost' of function compositions.

Game $\mathcal{M}_2$ corresponds to the same computational model as game $\mathcal{M}_1$, which is the CREW-PRAM model. The difference is that in an execution of $\mathcal{M}_2$, the variables in $\mathcal{A}(t)$ are obtained from the ones in $\mathcal{A}(t-1)$ by always applying the same operator. This is interesting for implementation purposes. A non-regular execution of $\mathcal{M}_1$ would be practically very intricate to implement since each step would be essentially different. Another advantage of an execution of $\mathcal{M}_2$ is that the number of memory locations needed to carry out the calculations is easy to compute: it is equal to $|\mathcal{A}(t)|$ (independent of t).

$\mathcal{M}_3$ **: Synchronous One-Pass Execution.** The rules of $\mathcal{M}_3$ are obtained by restricting the ones of $\mathcal{M}_2$ as follows.

- (R1) Unchanged;
- (R2c) (*playing rule*) same as (R2b) with the following additional restriction. Each node in $\mathcal{D}$ must be computed only once during the whole execution.
- (R3) unchanged;
- (R4) unchanged.

In rule (R2c), the important point is the difference that exists between computing a node and keeping the result into the memory.

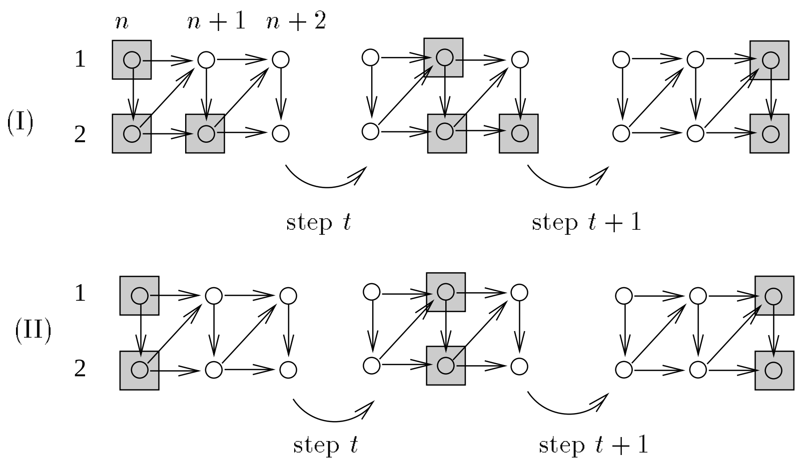


Figure 6: Rule $\mathcal{M}_2$ (I) and rule $\mathcal{M}_3$ (II).

Let us consider the example of Figure 6 (I). Each node on line 1 is computed twice, whereas each node on line 2 is computed only once. Let us detail this. Node $(1, n+2)$ is computed at step t (it is needed as an auxiliary for the computation of node $(2, n+2)$) but it is not kept into memory. Node $(1, n+2)$ is then computed a second time at step $t+1$. On the other hand, node $(2, n+2)$ is computed at step t and is kept into the memory. It does not have to be computed a second time at step $t+1$, as the computed value is just moved from one register to another.

In Figure 6 (I), we have an example of an execution satisfying rule $\mathcal{M}_2$ but not $\mathcal{M}_3$. On the other hand, in Figure 6 (II), we have an execution which verifies rule $\mathcal{M}_3$. The corresponding computation pattern is :

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

The computational model corresponding to game $\mathcal{M}_3$ is still the CREW-PRAM model. Rule (R2c) may look cumbersome but it actually corresponds to a natural notion for the applications to be detailed later on.

Notations In the following, we use the notations :

- $\mathcal{E}$: the set of all possible (synchronous) executions under rule $\mathcal{M}_1$.
- $\mathcal{RE}$: the set of all possible executions under rule $\mathcal{M}_2$. Elements of $\mathcal{RE}$ will be called regular executions.
- $\mathcal{ORE}$: the set of all possible executions under rule $\mathcal{M}_3$. Elements of $\mathcal{ORE}$ will be called one-pass regular executions.

Since the rules are increasingly restrictive, we have $\mathcal{ORE} \subset \mathcal{RE} \subset \mathcal{E}$.

4 Cuts in the Dependence Graph and their Relation with $\mathcal{M}_1, \mathcal{M}_2$

From now on, it is always implicitly assumed that the system under study is *recycled*, see § 2.3. In this section, we concentrate on games $\mathcal{M}_1$ and $\mathcal{M}_2$.

We introduce the notions of cuts and consecutive cuts in a dependence graph. We show that cuts (resp. consecutive cuts) are closely related to executions of the pebble game under game $\mathcal{M}_1$ (resp. game $\mathcal{M}_2$).
We show that there always exists a minimal cut which is consecutive (Lemma 4.7). It will allow us to prove Theorem 4.11, the main result of the section:

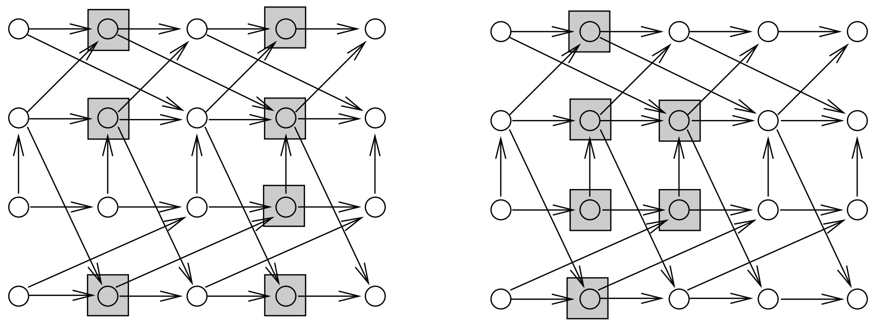
$$\min_{e \in \mathcal{RE}} \mathcal{P}(e) = \min_{e \in \mathcal{E}} \mathcal{P}(e) = \min_{C \text{ cut of } \mathcal{D}} |C|\,,$$

where the notations are defined in § 3.1. As a direct consequence, we show in § 4.3 that Problem *MinPeb* can be solved with a polynomial algorithm for games $\mathcal{M}_1$ and $\mathcal{M}_2$.

4.1 Definitions

Let us recall some classical definitions of graph theory, all defined on the dependence graph $\mathcal{D}$. For further references, see [9, 14] 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.*



A non consecutive cut A consecutive cut

Figure 7: Consecutive and non consecutive cuts.

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 with a minimal number of nodes 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. A flow $\mathcal{F}$ is 1-periodic if we have: the arc $(i,n) \to (j,m)$ belongs to $\mathcal{F}$ if and only if $(i,n+1) \to (j,m+1)$ belongs to $\mathcal{F}$.*

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 simple transformation, each node being replaced by two nodes connected by an arc, would allow us to go back to the original definitions.

Definition 4.4 (section). *A section S in $\mathcal{D}$ is a set of nodes with exactly one node per line, $S = \{(i, n_i), i \in V\}$.*

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

Definition 4.5 (left, right section). *The left (resp. right) section C_w, with w for* west*, (resp. C_e, with e for* east*) of a finite 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 4.6 (consecutive cut). *A cut C in $\mathcal{D}$ is consecutive if on each line of $\mathcal{D}$, C contains only consecutive nodes, i.e.:*

$$\text{For all } i \in V,\ (i,n) \in C \text{ and } (i,n+1) \notin C \Rightarrow (i,n+k) \notin C, \forall k > 0.$$

Examples of consecutive and non-consecutive cuts are displayed in Figure 7.

Lemma 4.7. *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, we prove that there are no arcs from C_w to $C_e + k, k \geqslant 2$ (where $(i,n) \in C_e + k$ iff $(i, n-k) \in C_e$).

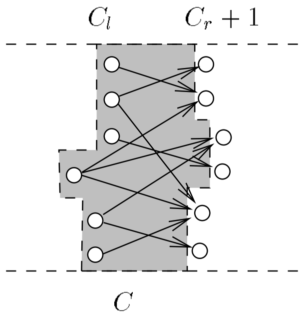


Figure 8: Graph G made from the right and left sections of C

Let us assume that there exists such an arc, that we denote by $(i, n) \to (j, m)$. By 1-periodicity, there is an arc between nodes $(i, n-1)$ and $(j, m-1)$. Now consider the bi-infinite path

$$\cdots \to (i, n-3) \to (i, n-2) \to (i, n-1) \to (j, m-1) \to (j, m) \to (j, m+1) \to \cdots$$

It does not intersect C which is a contradiction.
We consider the sub-graph G of $\mathcal{D}$ made of the nodes $C_w \cup (C_e + 1)$ and the arcs between C_w and $C_e + 1$ in $\mathcal{D}$, see Figure 8. We recall that a cut in a *finite* graph G is a set of nodes such that, when removed from G, there is no arc remaining. The set C_w is a cut in G. Let Δ be a cut in G of minimal size. We have $|\Delta| \leqslant |C_w|$. If $|\Delta| < |C_w|$ then $(C \backslash C_w) \cup \Delta$ would be a consecutive cut in $\mathcal{D}$ strictly smaller than C, which would contradict the fact that C is a minimal consecutive cut. Therefore, we have $|\Delta| = |C_w|$.
An adapted version of a famous "minimax" theorem first proved by König (1931) states that we can find $|\Delta|$ node-disjoint arcs in G. Since $|\Delta| = |C_w| = |C_e + 1|$, these arcs define a one to one mapping ϕ from C_w to $C_e + 1$. 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 $(i, n) \in C_w$ and all $k \in \mathbb{Z}$. These arcs form a 1-periodic 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 $|C_m| = |C|$. $\square$

This lemma is interesting by its own. In particular, it gives a proof of the minimax theorem (which exists in many versions) for an infinite 1-periodic and recycled graph.

Corollary 4.8. *The size of the minimal cut is equal to the size of the maximal flow in $\mathcal{D}$. Furthermore, there exists a maximal flow $\mathcal{F}$ in $\mathcal{D}$ which is 1-periodic.*

4.2 Cuts and pebbles

Lemma 4.9. *Let C be a finite consecutive cut. There is a regular execution $e \in \mathcal{RE}$ such that C is a configuration of e.*

Proof. Let C be a consecutive cut in $\mathcal{D}$. We want to prove that it is possible to have $\mathcal{A}(t) = C$ and $\mathcal{A}(t+1) = C + 1$ (note that $C + 1 = (C \backslash C_w) \cup (C_e + 1))$. It is enough to prove that for each node (i, n) in $C_e + 1$, 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. It would contradict the fact that C is a cut. $\square$

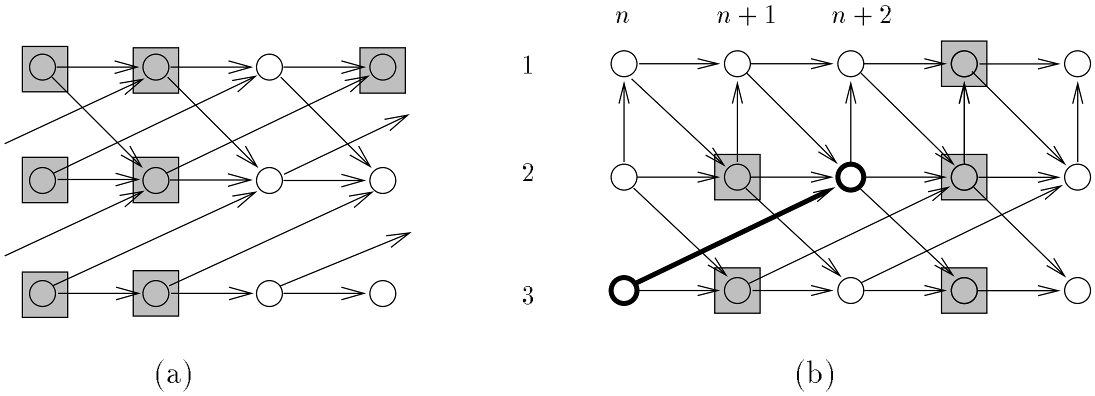


Figure 9: Two counter-examples.

The converse of Lemma 4.9 is not true: a configuration of a regular execution need not be a consecutive cut. This is illustrated on Figure 9-(a). Also note that there are non-consecutive cuts which are not configurations of a regular execution, as illustrated in the example of Figure 9-(b). In this 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 can not belong to a regular execution.

Lemma 4.10. *A configuration of any execution $e \in \mathcal{E}$ is a finite cut in $\mathcal{D}$. Conversely, let C be a finite cut in $\mathcal{D}$. There is an execution $e \in \mathcal{E}$ such that C is a configuration of e.*

Proof. Let $\mathcal{A}(t)$ be the t-th configuration of some execution e belonging to $\mathcal{E}$. All the configurations of e are finite by definition. Therefore the total number of nodes that received a pebble up to step t is finite.
Now, 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 rule (R3) of game $\mathcal{M}_1$, no node on P will receive a pebble during the execution, after step t. Combining this and the fact that the total number of nodes that received a pebble up to step t is finite, only a finite number of positive nodes on P receive a pebble during the execution e. This contradicts the fact that e has to put pebbles on all nodes.
Let us prove the converse result. Let C be a finite cut. Let $e = \{\mathcal{A}(t), t \in \mathbb{N}\}$ be any regular execution of game $\mathcal{M}_1$. Such executions exist (see Lemma 4.9). Let $\mathcal{N}$ be the set of positive nodes (i, n) such that there exists an infinite path ending in (i, n) and which does not intersect C. As C is a cut, $\mathcal{N}$ is finite. Let $T = \sup\{t \mid (\mathcal{N} \cup C) \cap \mathcal{A}(t) \neq \emptyset\}$. Note that T is finite since $\{\mathcal{A}(t)\}$ is regular and $\mathcal{N}$ is finite. We define $\tilde{e} = \{\tilde{\mathcal{A}}(t)\}$ as follows:

$$\tilde{\mathcal{A}}(t) = \begin{cases} \bigcup_{n=0}^{t} \mathcal{A}(n) & \text{if } t \leqslant T, \\ C & \text{if } t = T+1, \\ \mathcal{A}(t-1) & \text{if } t > T+1\,. \end{cases}$$

Let us show that $\tilde{e}$ is an execution of $\mathcal{M}_1$. We have $C \subset \bigcup_{n=0}^{T} \mathcal{A}(n)$, therefore, it is possible to set $\tilde{\mathcal{A}}(T+1) = C$. By definition of T, $\mathcal{A}(T+1)$ does not intersect $\mathcal{N}$, therefore, we can set $\tilde{\mathcal{A}}(T+2) = \mathcal{A}(T+1)$. Finally, $\tilde{e}$ contains all nodes in $\{\mathcal{A}(t), t \in \mathbb{N}\}$ and therefore all positive nodes. $\square$

We are now ready to state the main result of this section which states that, within all executions in $\mathcal{E}$, regular executions are dominant for Problem *MinPeb*.

Theorem 4.11. *Let us consider a recycled system of URE. We play the pebble game on its associated dependence graph $\mathcal{D}$ under rules $\mathcal{M}_1$ and $\mathcal{M}_2$. We have*

$$\min_{e \in \mathcal{RE}} \mathcal{P}(e) = \min_{e \in \mathcal{E}} \mathcal{P}(e) = \min_{C \text{ cut of } \mathcal{D}} |C| .$$

In words, there exists a regular execution which requires a minimal number of pebbles, this number being equal to the size of a minimal cut.

Proof. It is a direct consequence of Lemmas 4.10, 4.9 and 4.7. First, note that all configurations are cuts, according to Lemma 4.10. Let C be a consecutive cut of minimal size, which exists by Lemma 4.7. By Lemma 4.9, C is a configuration of a regular execution. $\square$

Theorem 4.11 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.

4.3 Complexity results for $\mathcal{M}_1$ and $\mathcal{M}_2$

Proposition 4.12. *Let $\mathcal{R} = (V, E, \Gamma)$ be the reduced graph associated with a recycled system of URE with non-negative delays. We set $\Gamma_A = \sum_{e \in E} \Gamma(e)$. For games $\mathcal{M}_1$ and $\mathcal{M}_2$, Problem* MinPeb *can be solved using an algorithm having a complexity $O\left(\Gamma_A^2 |V|^2\right)$.*

If the system of URE has negative delays, it is possible to go back to non-negative delays (see Remark 2.1) and apply Proposition 4.12 to the new system.

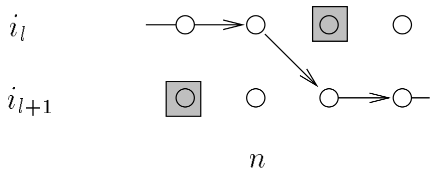
In order to prove Proposition 4.12, we are going to compute a maximal flow in $\mathcal{D}$ and then apply Corollary 4.8. If we want to use the algorithm of Ford and Fulkerson [9] to compute a maximal flow, we need first to restrict ourselves to a finite graph.
We call *span* of a cut the difference between the largest and the smallest of the numberings of columns containing a node of the cut.
A *slice* of $\mathcal{D}$ of dimension n is defined as the subgraph of $\mathcal{D}$ having nodes $\{(i,k), i \in V, 0 \leqslant k \leqslant n\} \cup \{T, B\}$ where T (Top) and B (Bottom) are two special nodes. There is an arc $T \to (i,k)$, $0 \leqslant k \leqslant n$, if $\exists (j,l), l < 0$, such that there is an arc $(j,l) \to (i,k)$ in $\mathcal{D}$. There is an arc $(i,k) \to B$, $0 \leqslant k \leqslant n$, if $\exists (j,l), l > n$, such that there is an arc $(i,k) \to (j,l)$ in $\mathcal{D}$.
If a consecutive minimal cut spans over less than n columns, then $\mathcal{D}$ and a slice of dimension n have the same minimal consecutive cut (the special nodes T and B are not allowed to belong to the cut). So it is important to determine, or at least to bound, the span of a consecutive minimal cut.

Lemma 4.13. *If all the delays are non-negative, the span of a minimal consecutive cut is smaller than the total sum of the delays in $\mathcal{R}$, i.e. smaller than $\Gamma_A = \sum_{e \in E} \Gamma(e)$.*

Proof. Let C be a minimal consecutive cut and let $\mathcal{F}$ be a maximal 1-periodic flow, see Corollary 4.8.

The associated maximal 1-periodic flow, see Corollary 4.8, $\mathcal{F}$ is a set of paths in $\mathcal{D}$. First, 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 line $(i, .)$ is not in $\mathcal{F}$, but this means that the bi-infinite path $\{(i, n), n \in \mathbb{Z}\}$ can be added to the flow $\mathcal{F}$ and this contradicts the maximality of $\mathcal{F}$.

Let P_1 be any path in $\mathcal{F}$. It follows from the 1-periodicity of $\mathcal{F}$ that P_1 is periodic. Let $i_0, i_1, \cdots, i_{l_1}, i_0, i_1, \ldots$ be the successive lines visited by the path P_1. Let (i_0, n) and $(i_0, n + k_1)$ be the consecutive nodes visited by the path P_1 on line $(i_0, .)$. Using the 1-periodicity of $\mathcal{F}$, the total number of paths intersecting lines $i_0, i_1, \cdots, i_{l_1}$ in $\mathcal{F}$ is k_1. It implies that the cardinal of C over the lines $i_0, i_1, \cdots, i_{l_1}$ is exactly k_1 (Corollary 4.8). Assume that the span of C on lines $i_0, i_1, \cdots, i_{l_1}$ is strictly greater than k_1. Then there exists a column, say n, not intersecting C and such that on some of the lines $\{i_0, i_1, \cdots, i_{l_1}\}$, C is on the 'right' of column n and on some others C is on the 'left' of column n. Let l be such that C is on the 'right' of n at line i_l and on the 'left' at line i_{l+1}. There exists an arc of the type $(i_l, n) \to (i_{l+1}, n + h), h \geqslant 0$ (the delays



are non-negative) in the flow $\mathcal{F}$. Then the path $\{(i_l, n - u), (i_{l+1}, n + h + v), u, v \in \mathbb{N}\}$ does not intersect the cut C, see the Figure 4.3. This is a contradiction. We conclude that the span of C over the lines $i_0, i_1, \cdots, i_{l_1}$ is smaller than 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 path P_1 and all its shifts are in the flow $\mathcal{F}$ and contain all the nodes in the lines $i_0, i_1, \cdots, i_{l_1}$. If $i_0, i_1, \cdots, i_{l_1}$ do not cover all the lines, a new path P_2 in $\mathcal{F}$ not intersecting the lines $i_0, i_1, \cdots, i_{l_1}$ ranges over different lines, say $i_{l_1+1}, i_{l_1+2}, \cdots, \cdots, i_{l_2}$, and defines a circuit L_2 in $\mathcal{R}$ similarly. The span of C on the lines $i_{l_1+1}, \cdots, i_{l_2}$ is smaller than k_2, the total delay of circuit L_2. We apply the same argument until all lines in $\mathcal{D}$ are covered. This defines a set H of circuits partitioning the nodes of $\mathcal{R}$.

We build a new multi-graph $\mathcal{G}$ starting with $\mathcal{R}$ and where each circuit in H is merged into one single node. The graph $\mathcal{G}$ has $|H|$ nodes and the arcs of $\mathcal{G}$ correspond to the arcs of $\mathcal{R}$ which do not belong to any circuit in H. Considering two nodes in $\mathcal{G}$, say L_1 and L_2, the span of C over lines $i_0, \cdots, i_{l_2}$ can be choosen to be smaller than $k_1 + k_2 + d$, where d is the maximum delay on all arcs between the nodes L_1 and L_2 in $\mathcal{G}$. Overall, the cut C can be choosen to have a span which is smaller than the sum of the delays on all the circuits in H plus the sum of the delays on all the arcs in $\mathcal{G}$. No delay is counted twice in this upper bound. Therefore, the total span of C is smaller than the total sum of the delays in $\mathcal{R}$. $\square$

Proof of Proposition 4.12. A slice of $\mathcal{D}$ of size Γ_A has the same minimal consecutive cut as $\mathcal{D}$ itself. The computation of the maximal flow in a finite slice can be done using the augmenting path algorithm, see [9, 14]. Starting with a 1-periodic flow (the recycled lines) and maintaining the 1-periodicity throughout the construction yields a maximal 1-periodic flow. The complexity of this construction of the maximal flow is $O(\Gamma_A^2 |V|^2)$. By Corollary 4.8, it provides the size of a minimal cut in $\mathcal{D}$. Furthermore, a standard procedure provides a minimal consecutive cut starting from a maximal flow (with a complexity $O(\Gamma_A|V|)$). Using Lemma 4.9, an execution of game $\mathcal{M}_2$ (or $\mathcal{M}_1$) using a minimal number of pebbles, is obtained from the minimal consecutive cut.

For the game $\mathcal{M}_3$, the problem *MinPeb* is solved by working on the reduced graph. A polynomial algorithm is given in § 6.5.

5 Compatible Cuts and their Relation with $\mathcal{M}_3$

We introduce the notion of compatible cuts. It enables us to show Theorem 5.5, the main result of the section, which is the analog of Theorem 4.11:

$$\min_{e\in\mathcal{ORE}} \mathcal{P}(e) = \min\left\{|C|,\ \ C \text{ compatible cut of } \mathcal{D}\right\}\ .$$

Let us introduce some new definitions.

Definition 5.1 (crossings). *We say that an arc crosses a section* $S = \{(i, n_i), i \in V\}$ from left to right *if it is an arc of the form* $(i, n_i - h) \to (j, n_j + l)$ *with* $h \geqslant 0$ *and* $l \geqslant 1$. *An arc* $(i, n_i + l) \to (j, n_j - h)$ *crosses* S from right to left *if* $l \geqslant 1$ *and* $h \geqslant 0$.

Definition 5.2 (compatible section, compatible cut). *A section in* $\mathcal{D}$ *is* compatible *if no arc crosses the section from right to left. A consecutive cut is said to be* compatible *if its right section is compatible.*

Roughly speaking a compatible cut is a consecutive cut which agrees with the dependence relations in the system of URE. Compatible cuts are connected to one-pass executions through the next two lemmas.

Lemma 5.3. *Let* C *be a compatible cut. There is a one-pass regular execution* $e \in \mathcal{ORE}$ *such that* C *is a configuration of* e.

Proof. Let C be a compatible cut. Since C is consecutive by definition, Lemma 4.9 tells us that C is a configuration of a regular execution which can be written as $e = \{C + t, t \in \mathbb{N}\}$. Suppose that e is not one-pass. This means that there exists a node, say (j, m) which is computed twice in e.
Let us assume that node (j, m) receives a pebble at step t_0 and that this pebble is removed at step t_1, $t_1 > t_0$. By regularity of the execution e, node (j, m) will not be used at step $t > t_1 + 1$. Indeed, a node in $C + t$ only depends on variables in $C + t - 1$ or 'below'.
Now assume that node (j, m) is used at step $t < t_0$ to compute another node, say (i, n). The path $(j, m) \to (i, n)$ crosses the right section of $C + t$ from right to left. Therefore it contains an arc that crosses the right section of $C + t$ from right to left. This contradicts the fact that C is compatible. □

Lemma 5.4. *The configuration of a one-pass regular execution is a compatible cut.*

Proof. Let $e = \{C + t, t \in \mathbb{N}\}$ be a one-pass regular execution. First, C is a cut by Lemma 4.10. Next, C is consecutive. Indeed, if C is not consecutive on line i, then each variable $X_i(n)$ receives a pebble at least twice in e, and *a fortiori* this means it is computed at least twice.

It remains to show that C is compatible. Assume that C is not compatible. Then, there exists a node (i, n) belonging to the right section of $C + t$, a positive integer k and a node $(j, m) \in (C + t + k)\backslash(C + t)$ such that the arc $(j, m) \to (i, n)$ belongs to $\mathcal{D}$. But this implies that in the execution e, the computation of $X_j(m)$ is performed twice: once as an auxiliary computation at step t to compute $X_i(n)$ and once at step $t + k$. This contradicts the fact that e is one-pass. □

Theorem 5.5. *Let us consider a recycled system of URE. We play the pebble game on its associated dependence graph $\mathcal{D}$ under rules $\mathcal{M}_3$. We have*

$$\min_{e \in \mathcal{ORE}} \mathcal{P}(e) = \min \{|C|, \ C \textit{ compatible cut of } \mathcal{D}\} \ .$$

Proof. It is an immediate corollary of Lemmas 5.3 and 5.4. □

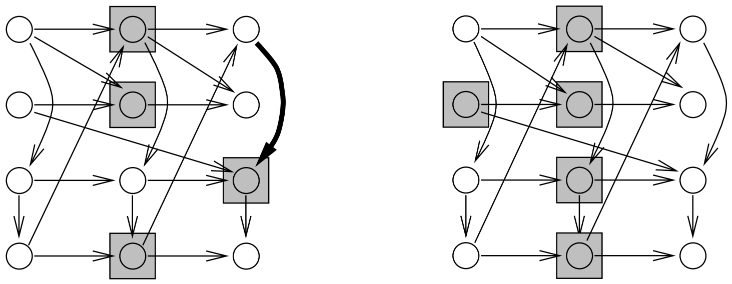


Figure 10: Non compatible and compatible cuts.

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 there is a minimal consecutive cut of size 4.
Therefore, rule $\mathcal{M}_3$ requires more memory in general than rule $\mathcal{M}_2$.

6 Delays in the Reduced Graph

In the previous sections, we have investigated the relations between executions of a system of URE and cuts in the dependence graph. In this section, we connect these two notions with values of the delays in the reduced graph.
Let us consider a system of URE $\mathcal{S}$ with variables $\{X_i(n), i \in V, n \in \mathbb{Z}\}$ and a regular execution $e = \{\mathcal{A}(t), t \in \mathbb{N}\}$ of the system. We introduce the modified system $\tilde{\mathcal{S}}$ with variables $\{\tilde{X}_i(n), i \in V, n \in \mathbb{Z}\}$ and the execution $\tilde{e} = \{\tilde{\mathcal{A}}(t), t \in \mathbb{N}\}$ defined as follows:

$$\begin{aligned} c_i &= \max\{n \in \mathbb{Z}^- \mid (i, n) \in \mathcal{A}(0)\} \\ \tilde{X}_i(n) &= X_i(n + c_i) \\ \tilde{\mathcal{A}}(t) &= \{(i, n) \text{ s.t. } (i, n + c_i) \in \mathcal{A}(t)\} \end{aligned}$$

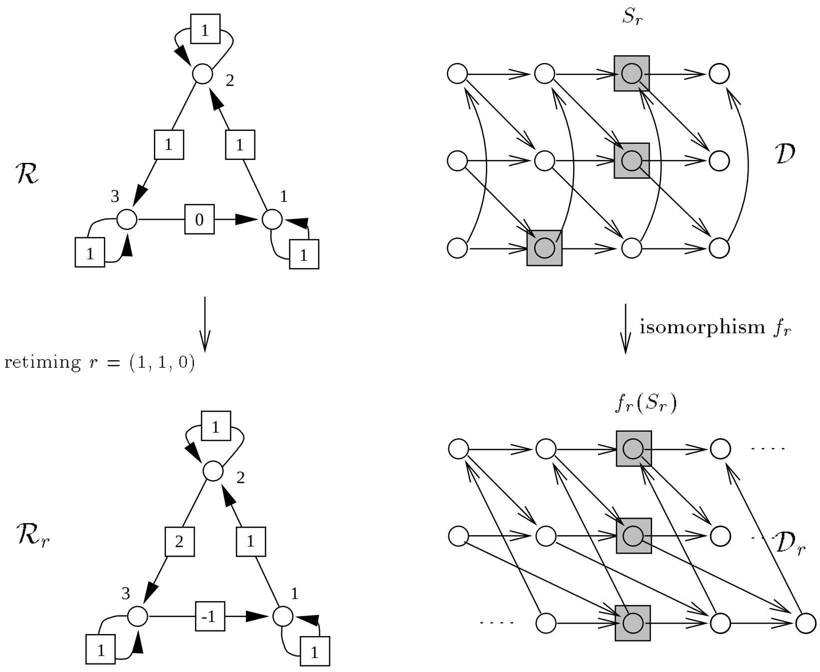


Figure 11: Retimed reduced graph and dependence graph.

The above definition is such that the right section of $\tilde{\mathcal{A}}(0)$ is $S_0 = \{(i,0), i \in V\}$. Viewed on the dependence graphs, the passage from $\mathcal{S}$ to $\tilde{\mathcal{S}}$ corresponds to a shift of the lines. Viewed on the reduced graphs, it corresponds to a *retiming*, i.e. a modification of the value of the delays, while preserving the graph topology.

We show (Lemma 6.6 and 6.7) that the total number of delays in $\tilde{\mathcal{R}}$ is closely related to regular executions. As a consequence, we obtain a polynomial algorithm to solve Problem *MinPeb* under rule $\mathcal{M}_3$, see § 6.5.

6.1 Retiming

Definition 6.1 (retiming). *Let $\mathcal{R}$ be the reduced graph of a system of URE. A* retiming *of $\mathcal{R}$ is a node function $r : V \to \mathbb{Z}$ which specifies a new graphe $\mathcal{R}_r$, with the same nodes and arcs as $\mathcal{R}$. The value of the delay on an arc $e = (i,j)$ in $\mathcal{R}_r$ is equal to $\Gamma_r(e) = \Gamma(e) + r(i) - r(j)$.*

The notion of retiming is classical in digital circuits (see [17] and § 7, where we provide a detailled discussion of its usefulness in this context) and in Petri nets, where it corresponds to the firing of transitions (see § 8).

In the example of Figure 11, the new values of the delays correspond to a retiming r such that $r(1) = 1, r(2) = 1$ and $r(3) = 0$.

Retiming may create negative delays as in Figure 11.

Lemma 6.2. *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 $e = (i,j)$, $\Gamma_r(e) = \Gamma(e) + r(i) - r(j) =$

$\Gamma(e)+r'(i)-r'(j) = \Gamma_{r'}(e)$. Conversely, if $\Gamma_{r'}(e) = \Gamma_r(e)$, then $r(i) = r'(i)+h$ and $r(j) = r'(j)+h$ for some $h \in \mathbb{Z}$. As $\mathcal{R}$ is connected, the constant h is the same for all the nodes in V. $\square$

The question that arises now is what is the notion corresponding to retiming in the system of URE $\mathcal{S}$ and in the dependence 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 lines as described in Lemma 6.3.

Lemma 6.3. *A retiming r in $\mathcal{R}$ corresponds to a transformation f_r between $\mathcal{D}$ and $\mathcal{D}_r$ defined by:*

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

The transformation f_r is an isomorphism of graphs, meaning that there is an arc between u and v in $\mathcal{D}$ iff there is one between $f_r(u)$ and $f_r(v)$ in $\mathcal{D}_r$. It will also be called a retiming of $\mathcal{D}$.

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

We recall that the notion of *section* was introduced in Definition 4.4. We associate with a retiming r in $\mathcal{R}$, the section $S_r = \{(i, r(i)), i \in V\}$ in $\mathcal{D}$.
Lemmas 6.2 and 6.3 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 equivalence class and call it the section associated with the retiming r, for instance S_0 corresponds to the equivalence class of $\{(i,0), i \in V\}$.

6.2 Counting the delays

Given a graph $\mathcal{R} = (V, E, \Gamma)$, we define the *total number of delays* of $\mathcal{R}$ as follows

$$\Gamma_A(\mathcal{R}) = \sum_{i \in V} \sum_{(j,\gamma) \in \Delta_i} \gamma\,. \tag{7}$$

It corresponds to the number of delays appearing in the graphical representation of the reduced graph $\mathcal{R}$ as defined in § 2.2. See for example the graph $\mathcal{R}$ on Figure 12.

Given a graph $\mathcal{R} = (V, E, \Gamma)$, another quantity of interest is the following one

$$\Gamma_B(\mathcal{R}) = \sum_{j \in V} \max\{\gamma \mid \exists i \text{ s.t. } (j,\gamma) \in \Delta_i\}\,. \tag{8}$$

When the delays are positive ($\forall e \in E, \Gamma(e) \geqslant 0$), Γ_B corresponds to the total number of delays (Γ_A) in a modified reduced graph obtained by performing a forward splitting of the nodes. In the context of digital circuits, this is also called register sharing, see § 7.2. Here is, given under the form of an algorithm, the formal construction of the `Forward Splitting` algorithm.

Algorithm 6.4 (Forward Splitting).

Input: *Reduced graph $\mathcal{R} = (V, E, \Gamma)$ with $\Gamma \geqslant 0$, functions associated with the nodes: $\{F_i, i \in V\}$.*

1. `Set` $V' = V$ `and` $E' = \emptyset$. `Associated functions` $F_i' = F_i, i \in V$.
2. `For all node` $v \in V$, `let` δ `be the maximum delay on all the output arcs of` v.
   - `Set` $v_0 = v$.
   - `If` $\delta > 0$, `create` δ `new nodes in` V', `called` $v_1, \cdots, v_\delta$. `Set` $F'_{v_i} = Id$, `the identity function, for` $i = 1, \ldots, \delta$.
   - `For each arc` $e = (v, u) \in E$ `with delay` $\Gamma(e) = \gamma$, `create an arc` $e' = (v_\gamma, u)$ `in` E' `with delay` $\Gamma'(e') = 0$.
   - `Add the arcs` (v_i, v_{i+1}), $0 \leqslant i \leqslant \delta - 1$, `in` E', `with delay` $\Gamma' = 1$.

Output: *Split reduced graph $\mathcal{R}' = (V', E', \Gamma')$ with $\Gamma' \geqslant 0$, functions associated with the nodes: $\{F_i', i \in V\}$.*

It is important to remark that the split graph $\mathcal{R}'$ is not necessarily recycled, as opposed to $\mathcal{R}$. We come back to this point in § 6.4.
The following proposition, easy to prove, justifies the `Forward Splitting` operation.

Proposition 6.5. *Let $\mathcal{R}$ be a reduced graph with positive delays and $\mathcal{R}'$ the associated split graph.*
(i)- The associated systems of URE $\mathcal{S}$ and $\mathcal{S}'$ have the same behavior. More precisely, borrowing the notations of the algorithm, we have

$$\forall v \in V, n \in \mathbb{Z},\ X_v'(n) = X_v(n) \text{ and } \forall v_i \in V' \setminus V, n \in \mathbb{Z},\ X'_{v_i}(n) = X_v(n-i)\ .$$

(ii)- Furthermore, we have

$$\Gamma_B(\mathcal{R}) = \Gamma_A(\mathcal{R}') = \Gamma_B(\mathcal{R}')\ . \tag{9}$$

Note that in order for Equation (9) to make sense, it is necessary to extend the definitions of Γ_A and Γ_B to non-recycled graphs.

In Figure 12, $\mathcal{R}'$ is the `Forward Splitting` of $\mathcal{R}$. In the split graph, there are two "dummy" nodes (associated with the identity function), represented by black dots. We have $\Gamma_A(\mathcal{R}) = 8$ and $\Gamma_B(\mathcal{R}) = \Gamma_A(\mathcal{R}') = \Gamma_B(\mathcal{R}') = 5$.

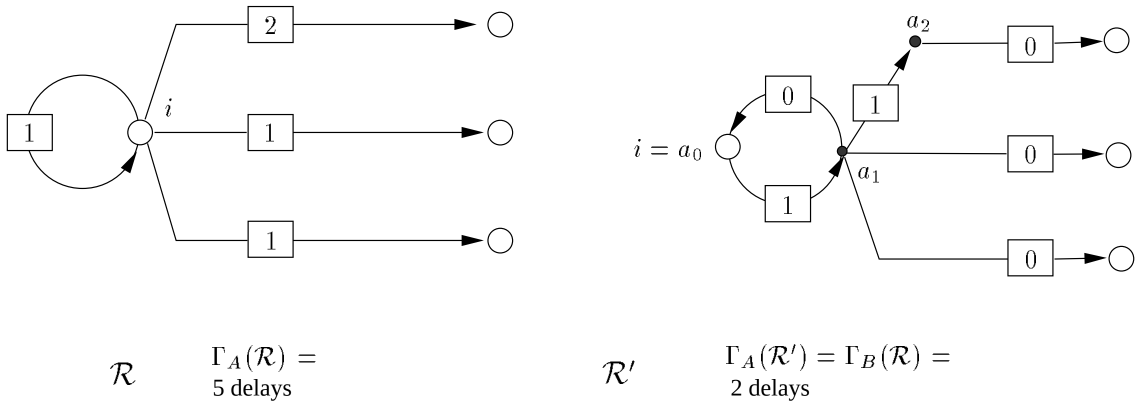


Figure 12: A reduced graph and the associated split graph.

6.3 Delays, cuts and pebbles

Given a section $S = \{(i, n_i), i \in V\}$ in $\mathcal{D}$, we define the cut $\mathcal{C}(S)$ in the following way.

$$\mathcal{C}(S) \stackrel{\text{def}}{=} \{(i,n), i \in V, n \leqslant n_i \mid \exists j \in V, m > n_j, (i,n) \to (j,m)\}\,. \tag{10}$$

The cut $\mathcal{C}(S)$ is consecutive and its right section is S. Furthermore, if any node is removed from the left 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. Any consecutive cut C with no redundant node on its left section is characterized by its right section $\underline{S}$ only. More precisely, it verifies $C = \mathcal{C}(\underline{S})$.

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

Lemma 6.6. *(i)- The number of delays $\Gamma_B(\mathcal{R})$ is equal to the cardinal of $\mathcal{C}(S_0)$. (ii)- Let r be a retiming of $\mathcal{R}$ and S_r an associated section in $\mathcal{D}$. Then the number of delays $\Gamma_B(\mathcal{R}_r)$ is equal to the cardinal of the cut $\mathcal{C}(S_r)$ in $\mathcal{D}$ (resp. $\mathcal{C}(S_0)$ in $\mathcal{D}_r$).*

Proof. The isomorphism f_r transforms the section $\mathcal{S}_r$ in $\mathcal{D}$ into the section $\mathcal{S}_0$ in $\mathcal{D}_r$. Hence (ii) is implied by (i). Let us work on graph $\mathcal{D}$. We are going to prove that $\Gamma_B(\mathcal{R}) = |\mathcal{C}(\mathcal{S}_0)|$. We consider a node i of $\mathcal{R}$. Let $m = \max\{\gamma \mid \exists j,\ (i,\gamma) \in \Delta_j\}$ (we have $m \geqslant 1$ as $(i,1) \in \Delta_i$) and let j be such that $(i,m) \in \Delta_j$. There is an arc in $\mathcal{D}$ from $(i,-m)$ to $(j,0)$ and no arc from a node on the 'left' of $(i,-m)$ (and on line i) to a node on the 'right' of S_0. Hence $\mathcal{C}(S_0)$ contains the nodes $(i,-m+1), \cdots, (i,0)$ on line i. The same argument repeated on each line finishes the proof. ☐

We recall that the arcs crossing a section *from right to left* and *from left to right* are defined in Definition 5.1.

Lemma 6.7. *(i)- The number of delays $\Gamma_A(\mathcal{R})$ is equal to the number of arcs crossing S_0 from left to right minus the number of arcs crossing it from right to left. (ii)- Let r be a retiming of $\mathcal{R}$ and S_r an associated section in $\mathcal{D}$. The number of delays $\Gamma_A(\mathcal{R}_r)$ is equal to the number of* *arcs in $\mathcal{D}$ (resp. $\mathcal{D}_r$) crossing section S_r (resp. S_0) from left to right minus the number of arcs crossing S_r (resp. S_0) from right to left.*

Proof. For the same reason as in Lemma 6.6, it is enough to prove (i). Let (i,j) be an arc of $\mathcal{R}$ with delay $\gamma \geqslant 0$. In $\mathcal{D}$, this arc induces exactly γ arcs crossing S_0 from left to right, the arcs:

$$(i, -\gamma+1) \to (j,1) \quad \cdots \quad (i,-1) \to (j, \gamma-1), \quad (i,0) \to (j,\gamma) \, .$$

Similarly, an arc (i,j) with delay $\gamma < 0$ induces exactly $-\gamma$ arcs crossing S_0 from right to left:

$$(i,1) \to (j, \gamma+1) \quad \cdots \quad (i, -\gamma-1) \to (j,-1), \quad (i,-\gamma) \to (j,0) \, .$$

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

The notion of compatible cut introduced in Definition 5.2 has a very natural interpretation in terms of delays.

Lemma 6.8. *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 cut $\mathcal{C}(S_r)$ is compatible in $\mathcal{D}$ (equiv. the cut $\mathcal{C}(S_0)$ is compatible in $\mathcal{D}_r$).*

Proof. If $\mathcal{R}_r$ has only non-negative delays, the argument used in the proof of Lemma 6.7 shows that all the arcs crossing S_r in $\mathcal{D}$, cross it from left to right. It implies that the cut $\mathcal{C}(S_r)$ is compatible. The converse result is proved by contradiction, using again the proof of Lemma 6.7. □

In Figure 13 (this example is the same as the one of Figure 10), we have represented the retimed reduced graphs associated with two sections (cuts) of $\mathcal{D}$. One of them is compatible and the other one is not compatible.

6.4 Summary

In § 4.2, we have established the relations between executions of the pebble game and cuts in the dependence graph. In § 6.3, we have established the relations between cuts and delays. As a by-product, we obtain the relations between delays and pebble configurations.
More precisely, let $e = \{\mathcal{A}(t), t \in \mathbb{N}\}$ be an execution such that $\mathcal{A}(t)$ is a consecutive and non-redundant (see p. 22) cut for all t. Note that we did not assume that e is regular. With the configurations $\mathcal{A}(t)$, we associate a retiming $r(t)$ and a reduced graph $\mathcal{R}_{r(t)}$ as follows:

$$r(t)_i = \max\{n_i \mid (i, n_i) \in \mathcal{A}(t)\} \, .$$

We have the following situations:

- If the execution e belongs to $\mathcal{E}$, the configurations $\mathcal{A}(t)$ may have different shapes at each step. Then, the reduced graphs $\mathcal{R}_{r(t)}$ may have changing values for the delays.

- For an execution e belonging to $\mathcal{RE}$, the configurations are just shifted between two steps. It implies that the reduced graphs $\mathcal{R}_{r(t)}$ are all identical, with a fixed value of the delays.

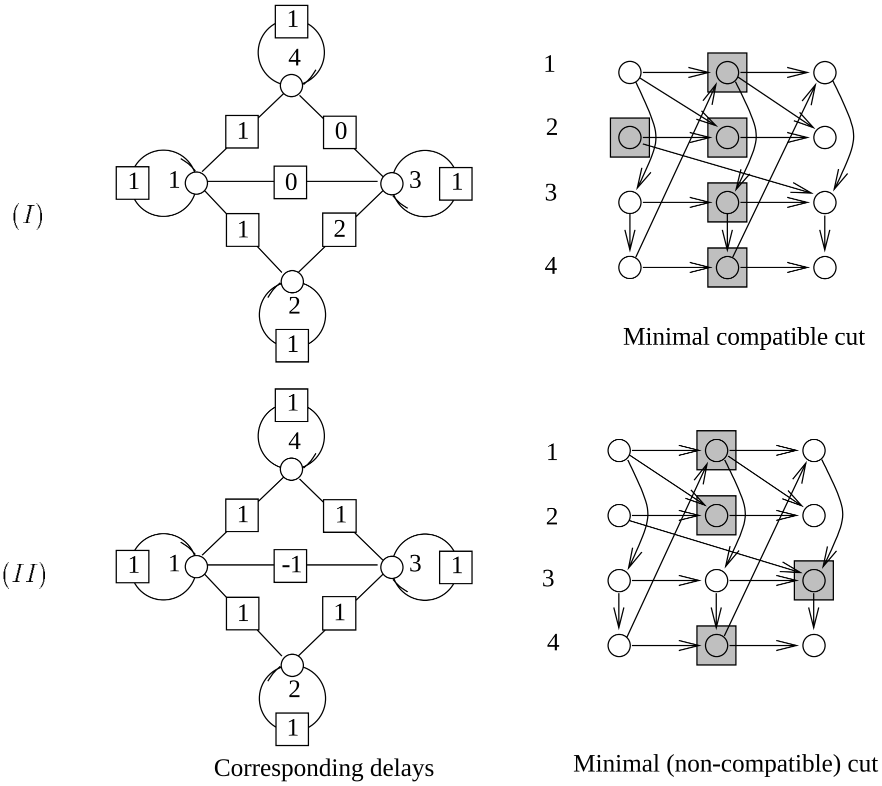


Figure 13: Compatible and non compatible cuts, non-negative and negative delays.

- Finally, an execution e belonging to $\mathcal{ORE}$ corresponds to identical reduced graphs $\mathcal{R}_{r(t)}$ with fixed and non-negative values of the delays.

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

Games	Executions	Cuts in $\mathcal{D}$	Delays in $\mathcal{R}$
$\mathcal{M}_1$	execution in $\mathcal{E}$	arbitrary cut	changing delays
$\mathcal{M}_2$	regular execution, $\mathcal{RE}$	consecutive cut	fixed delays
$\mathcal{M}_3$	one-pass reg. exec., $\mathcal{ORE}$	compatible cut	non-negative fixed delays

A general remark is that the main theoretical results, Theorems 4.11 and 5.5, apply only to recycled graphs. Using them, we have been able to solve Problem *MinPeb* for an initial graph which is recycled. However, the solution proposed involves the construction of an associated graph, which is not recycled. It is not a problem as we do not need to apply Theorems 4.11 or 5.5 on this associated graph.

6.5 Complexity results for $\mathcal{M}_3$

Proposition 6.9. *Let $\mathcal{R} = (V, E, \Gamma)$ be the reduced graph associated with a recycled system of URE. Under game $\mathcal{M}_3$, Problem* MinPeb *can be solved using an algorithm of complexity $O(|E|^2 \log |V| + |V||E| \log^2 |V|)$.*

Proof. As detailed above, there is a one-to-one correspondence between minimal one-pass regular executions and retimed reduced graphs with non-negative delays.
In Leiserson and Saxe, § 8 of [17], an algorithm is given to solve the following problem: find a retimed reduced graph $\mathcal{R}_r$ with only non-negative delays and minimizing $\Gamma_B(\mathcal{R}_r)$. It is a minimum-cost flow algorithm; it provides an explicit solution and its complexity is $O(|E|^2 \log|V| + |V||E|\log^2|V|)$. Using Lemma 5.3 and 6.6, the cut $\mathcal{C}(S_r)$ is compatible and the execution $\{\mathcal{C}(S_r)+t, t \in \mathbb{N}\}$ is one-pass regular and solves Problem *MinPeb* under rules $\mathcal{M}_3$ according to Theorem 5.5. □

The algorithm of [17] was developed in the context of digital circuits, see § 7. An efficient implementation of this algorithm can be found in Shenoy and Rudell [24].

7 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 is two-fold. First, algorithms developed for digital circuits can be used to get optimal executions of a system of URE, see the previous section. Second, we will show that the results proved so far enable to prove some new results for recycled digital circuits, see Theorem 7.4 below.

7.1 Definition of a circuit

A digital circuit is constituted by functional gates, wires and registers. More precisely , (i)- 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,...); (ii)- A wire between element i and element j enables to transfer the output data of i which becomes an input data for j; (iii)- A register corresponds to a storage facility, or a memory cell of finite size. If there are p registers between elements i and j, it 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 or elements; (2)- compute a new output data; (3)- send the output data to downstream registers or elements.
- Register : (1)- transmit the stored data downstream to another register or a functional element; (2)- remove the stored data; (3)- receive and store a new data from upstream from another register or a functional element.

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

Let $X_i(n)$ be the n-th variable computed at element i. Since registers are finite size memory cells, the variables $X_i(n)$ can only take a finite number of values. The set of all these possible values is denoted by $\mathcal{W}$.
After n clock ticks, for each element i, 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 $X_j(n+m), m > 0$.
It follows 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 consistent 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 consider only *constructive circuits*, i.e. circuits whose associated URE is constructive, and *recycled circuits*, i.e. circuits whose associated reduced graph is recycled.

The specificity of digital circuits (with respect to general reduced graphs) is that *only non-negative registers (delays) have a physical meaning.*
In Figure 14, we have represented the flow of data between clock ticks in a digital circuit. The graphical convention is consistent with the one of reduced graphs.

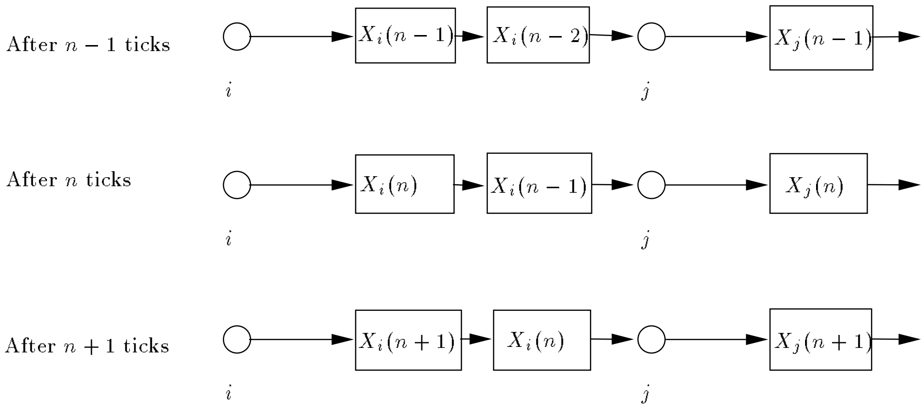


Figure 14: Digital circuit computing $X_j(n) = F_j(X_i(n-2), \ldots)$.

7.2 Minimizing the number of registers

We consider Problem *MinReg* which is classical in circuit design, see for instance Leiserson and Saxe [17], § 8.

Problem 2 (MinReg). *Given a recycled circuit, find a new circuit preserving the functional behavior and having a minimal number of registers.*

To make this statement precise, we have to define rigorously the *functional behavior* and the *number of registers* of a circuit.
Let $\{X_i(n), i \in V, n \in \mathbb{Z}\}$ and $\{\tilde{X}_i(n), i \in \tilde{V}, n \in \mathbb{Z}\}$ be the variables computed by the original and the new circuits respectively. The preservation of the functional behavior means that we have:

$$\forall i \in V,\ \exists u \in \tilde{V},\ \exists c_i \in \mathbb{Z} \text{ s.t. } \forall n \in \mathbb{Z},\ X_i(n) = \tilde{X}_u(n + c_i)\ . \tag{11}$$

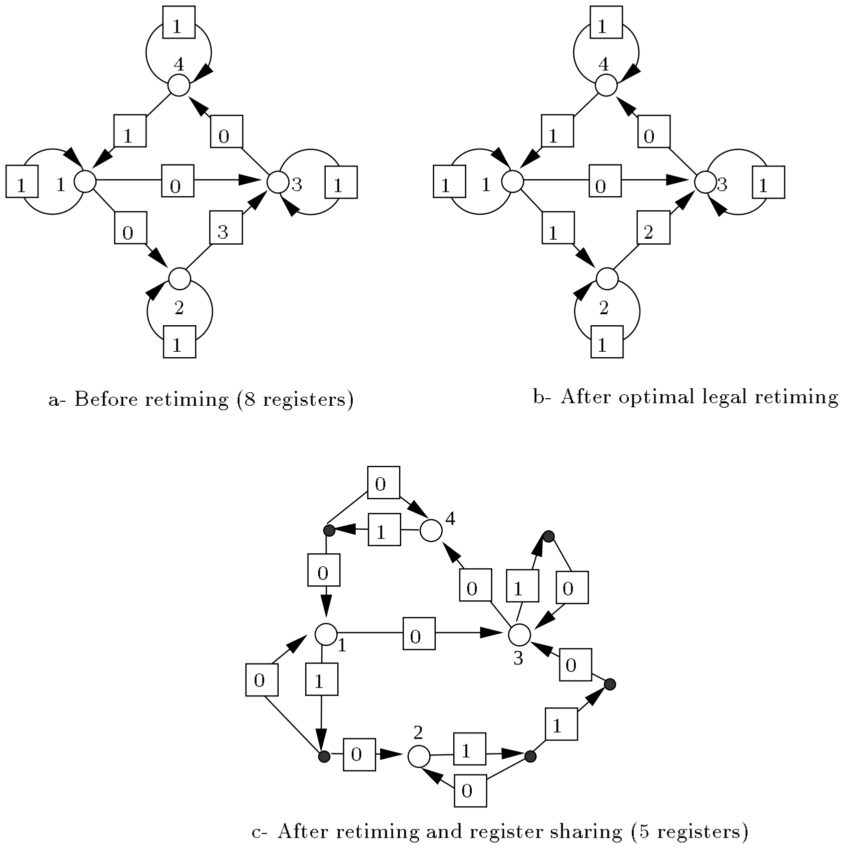
Registers can be viewed as delays in a reduced graph and the number of registers of a circuit $\mathcal{R}$ is the quantity $\Gamma_A(\mathcal{R})$ defined in Section § 6.2. Now, starting from a circuit $\mathcal{R}$, we can always perform the `Forward Splitting` algorithm 6.4 to obtain a circuit $\mathcal{R}'$. As an illustration, consider the example given in Figure 12. The `Forward Splitting` algorithm is called *register sharing* in the context of circuits, see [17].

The Problem *MinReg* is directly connected with the notions introduced in § 4 and § 5. It enables us to propose some complements to the results of [17] for the special case of recycled circuits.

In [17], Leiserson and Saxe define a notion of retiming which is exactly the one of Definition 6.1. They restrict their attention to *legal* retimings, as they are the only ones to have a physical meaning for circuits.

Definition 7.1. *A retiming r is* legal *if $\mathcal{R}_r$ has only non-negative delays.*

An example of legal retiming is given in Figure 15. If we perform register sharing on the original circuit (Figure 15-a), we would obtain a circuit with six registers. After a legal retiming and register sharing, we have only five registers (Figure 15-c).



a- Before retiming (8 registers)

b- After optimal legal retiming

c- After retiming and register sharing (5 registers)

Figure 15: Reducing the number of registers using legal retiming and register sharing.

Leiserson and Saxe prove that retiming and register sharing preserve the functional behavior of the circuit (this is also a direct consequence of Lemma 6.3 and Proposition 6.5). Then they propose an algorithm to compute the optimal legal retiming and also the optimal legal retiming when register sharing is allowed, see § 6.5.

However the question whether other circuit transformations can be used to get a circuit with even fewer registers remains to be answered.

Let us consider the best possible retiming in the original circuit without restricting ourselves to legal retimings. It corresponds to the choice of a minimal consecutive (but not necessarily compatible) cut in the associated dependence graph $\mathcal{D}$, see § 6.4. In the corresponding reduced graph, there may be some negative delays that cannot represent registers. However, it is possible to perform some appropriate modifications to go back to positive delays. It is done by duplicating some nodes in the circuit.

The `Duplicate` algorithm takes as input a graph $\mathcal{R} = (V, E, \Gamma)$ and produces a new graph $\mathcal{R}' = (V', E', \Gamma'), \Gamma' \geqslant 0$. In the description to follow, we use the notion of delays on paths: if P is an oriented path in $\mathcal{R}$ then the delay of P is the sum of all the delays on the arcs of P.

Algorithm 7.2 (Duplicate).
Input: *Reduced graph* $\mathcal{R} = (V, E, \Gamma)$, *functions associated with the nodes:* $\{F_i, i \in V\}$.

1. `Set` $V' = V$ `and` $E' = \emptyset$. `Associated functions` $F'_i = F_i, i \in V$.
2. `For each node` v `in` V, `let` $k(v)$ `be the minimum delay of all paths in` $\mathcal{R}$ `starting in` v.
   - `Set` $v_0 = v$.
   - `If` $k(v) < 0$, `then create` $|k(v)|$ `additional nodes in` V', $v_1, \cdots, v_{|k(v)|}$, `with associated functions`, $F'_{v_i} = F_v$.
3. `For each arc` $(u, v) \in E$ `with delay` $\Gamma(u, v) = \gamma$, `create in` E' `all the arcs of type` $(u_{\max(0, j-\gamma)}, v_j)$ `with delay` $\max(0, \gamma - j)$ `for all` $0 \leqslant j \leqslant \max(0, k(v))$.

Output: *reduced graph* $\mathcal{R}' = (V', E', \Gamma')$, $\Gamma' \geqslant 0$. *Associated functions* $\{F'_i, i \in V'\}$.

For each node $v \in V$, $k(v)$ is finite and is reached on a finite path since the constructivity of $\mathcal{R}$ implies that all circuits in $\mathcal{R}$ have a strictly positive delay.

The following proposition justifies the use of the `Duplicate` algorithm.

Proposition 7.3. *Let $\mathcal{S}$ and $\mathcal{S}'$ be the systems of URE associated with $\mathcal{R}$ and $\mathcal{R}'$ respectively. (i)- S and S' have the same functional behavior. More precisely, borrowing the notations of the* `Duplicate` *algorithm, we have*

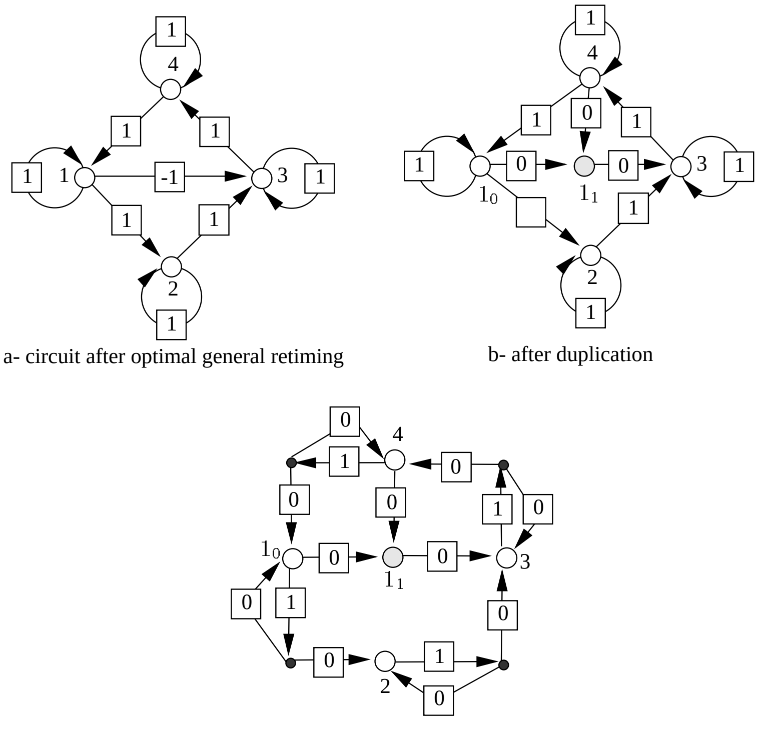
$$X'_{v_j}(n) = X_v(n + j), \ \forall v_j \in V', \forall n \in \mathbb{Z}\,; \tag{12}$$

(ii)- we have $\Gamma_B(\mathcal{R}) = \Gamma_B(\mathcal{R}')$.

Proof. The proof of *(i)* follows directly from the construction rules of $\mathcal{R}'$. Consider (12), specialized to $j = 0$, we get $X'_{v_0}(n) = X_v(n)$. The original circuit is embedded in the new circuit. As for *(ii)*, note that all the arcs exiting a duplication node (of type $v_j, j > 0$) have a zero delay. As for the new arcs from the nodes of type v_0, they all have delays smaller than or equal to the delays of the arcs of the original graph. The original arcs are kept with their original delays unchanged. We conclude that $\Gamma_B(\mathcal{R}) = \Gamma_B(\mathcal{R}')$. □

It is important to remark that the reduced graph $\mathcal{R}'$ (hence the associated system of URE) obtained by duplication is not recycled anymore.

We illustrate the algorithm `Duplicate` on an example. The circuit of Figure 16-a is obtained from the one of Figure 15-a by performing a non-legal retiming. By applying the `Duplicate` algorithm, we obtain the circuit of Figure 16-b, where node 1 has been duplicated into nodes 1_0 and 1_1.



a- circuit after optimal general retiming

b- after duplication

c- after duplication and register sharing (4 registers)

Figure 16: Reducing the number of registers using retiming, duplication and register sharing.

Now, after performing register sharing, the resulting circuit has only 4 registers (see Figure 16-c). The minimal number of registers we could get with only legal retimings and register sharing was 5, see Figure 15-c.

We now state the general result.

Theorem 7.4. *Let us consider a recycled digital circuit. When the functions computed at each node are general, a circuit with the same functional behavior and a minimal number of registers can be obtained by performing solely the three following operations (in this order):*
1. General retiming; 2. `Duplicate` *algorithm; 3.* `Forward Splitting` *algorithm (register sharing).*

Proof. Let us consider a recycled graph $\mathcal{R} = (V, E, \Gamma)$ (associated dependence graph $\mathcal{D}$). We realize the following operations: 1. Perform the optimal general retiming, to obtain a graph $\mathcal{R}_1$.
2. Apply the `Duplicate` algorithm to $\mathcal{R}_1$ to obtain the graph $\mathcal{R}_2$.
3. Transform the graph $\mathcal{R}_2$ into $\mathcal{R}_3$ by applying the Forward Splitting algorithm.

Let us detail the first operation. We find a minimal consecutive cut C of $\mathcal{D}$ (Proposition 4.12). Let $\{(i, r(i)), i \in V\}$ be the right section of C. We define the retimed graph $\mathcal{R}_1 = \mathcal{R}_r$. Using Lemma 6.6, $\Gamma_B(\mathcal{R}_1)$ is equal to the cardinal of C.

The three operations preserve the functional behavior of the circuit, see Lemma 6.3, Propositions 6.5 and 7.3. Furthermore, as a consequence of Proposition 7.3 and Equation (9), we have $\Gamma_B(\mathcal{R}_1) = \Gamma_B(\mathcal{R}_2) = \Gamma_A(\mathcal{R}_3)$. We conclude that $\Gamma_A(\mathcal{R}_3) = |C|$. It remains to be proved that there exists no other circuit, having the same functional behavior as $\mathcal{R}$, and with fewer registers than $\mathcal{R}_3$.

We consider $\mathcal{R}' = (V', E', \Gamma')$ another circuit which has the same functional behavior as $\mathcal{R}$. Let $\mathcal{D}'$ be the dependence graph associated with $\mathcal{R}'$. The preservation of the functional behavior implies that the set of nodes in $\mathcal{D}$ is included in the set of nodes in $\mathcal{D}'$. The mapping of the nodes of $\mathcal{D}$ onto $\mathcal{D}'$ that preserves the functional behavior is denoted by ϕ. By definition, $X_i(n) = X'_{\phi(i)}(\phi(n))$, if the node (i, n) in $\mathcal{D}$ is mapped on the node $(\phi(i), \phi(n)) = \phi(i, n)$ in $\mathcal{D}'$.
We consider a minimal cut C of $\mathcal{D}$. In $\mathcal{D}'$, we suppose that there exists a cut C' such that $|C'| < |C|$. We can assume that in $\mathcal{D}'$, we have $\phi(C)$ on the 'left' of C' and $\phi(C + k)$ on the 'right' of C', by choosing k large enough.
We recall that $\mathcal{W}$ denotes the finite set of possible values for a variable $X_i(n)$. In the dependence graph $\mathcal{D}$, there exists a flow (node-disjoint paths) from C to $C+k$ of size $|C|$ (Corollary 4.8). It implies that there is a general dependence between the variables attached to C and the variables attached to $C + k$, which can be put under the form of a general function $F : \mathcal{W}^{|C|} \to \mathcal{W}^{|C|}$. In particular, the functions F_i in Equation (2) can be chosen such that F is bijective. For example, choose $F_i(X_j(n - \gamma), \dots) = X_j(n - \gamma)$ if the arc $(j, n - \gamma) \to (i, n)$ belongs to the flow. In this case, the function F is merely a permutation of the coordinates.
Now let us consider the graph $\mathcal{D}'$, using the existence of the cut C', the function F can be decomposed as $F : \mathcal{W}^{|C|} \to \mathcal{W}^{|C'|} \to \mathcal{W}^{|C|}$. As $\mathcal{W}$ is finite, it contradicts the fact that F can be bijective.
The smallest cut C' in $\mathcal{D}'$ is at least as large as C. Finally, by using Lemma 6.6, we conclude that $\Gamma_A(\mathcal{R}') \geqslant \Gamma_B(\mathcal{R}') = |C'| \geqslant |C| = \Gamma_A(\mathcal{R}_3)$. $\square$

As recalled above, in Leiserson and Saxe [17], only legal retimings were considered. In Theorem 7.4, by considering all possible retimings, we were able to obtain a circuit with less registers, and even a minimal number of registers. However, in doing so, we obtain a circuit with a possibly larger number of functional elements. Hence, the practical interest of Theorem 7.4 also needs to be discussed in terms of the compared costs of functional elements and registers. In fact, the number of functional elements is increased both by the `Duplicate` and by the `Forward Splitting` algorithms. On the one hand, in the `Forward Splitting` case, only "dummy functions" are added. In the context of circuits, they consist in simple wire connections and do not perform any operation so that they should be very cheap to implement. On the other hand, in the `Duplicate` case, the added elements are equivalent to some of the original functional elements. Hence they could be more expensive to implement.

7.3 Summary and complexity

The transformations done to a circuit in order to obtain an equivalent circuit with the minimal number of registers can be summarized by the following scheme.

$$\begin{array}{ccccccc}
\mathcal{R} & \xrightarrow[\text{retiming}]{\text{optimal}} & \mathcal{R}_1 & \xrightarrow{\text{duplicate}} & \mathcal{R}_2 & \xrightarrow[\text{splitting}]{\text{forward}} & \mathcal{R}_3 \\
 & & \Gamma(\mathcal{R}_1) \in \mathbb{Z}^{E_1} & & \Gamma(\mathcal{R}_2) \in \mathbb{N}^{E_2} & & \Gamma(\mathcal{R}_3) \in \{0,1\}^{E_3} \\
 & & \Gamma_B(\mathcal{R}_1) \text{ minimal} & & \Gamma_B(\mathcal{R}_2) = \Gamma_B(\mathcal{R}_1) & & \Gamma_A(\mathcal{R}_3) = \Gamma_B(\mathcal{R}_2)
\end{array}$$

Corollary 7.5. *Let us consider a recycled digital circuit. A new circuit solving Problem MinReg can be obtained with an algorithm of complexity* $O(\Gamma_A(\mathcal{R})^2|V|^2 + |V|^3)$.

Proof. In the proof of Theorem 7.4, the algorithms used to go from $\mathcal{R}$ to $\mathcal{R}_1$, from $\mathcal{R}_1$ to $\mathcal{R}_2$ and from $\mathcal{R}_2$ to $\mathcal{R}_3$ are all polynomial, and $\mathcal{R}_3$ is a solution to the Problem *MinReg*.
To obtain $\mathcal{R}_1$, we apply the algorithm of Proposition 4.12 whose complexity is $O(\Gamma_A^2|V|^2)$. To obtain $\mathcal{R}_2$, we apply the `Duplicate` algorithm 7.2, whose complexity is $O(|V|^3 + \Gamma_A|E|)$. Let us justify this complexity. The first step consists in computing the quantities $k(u), u \in V$. It is equivalent to the search of a minimal weight path in a weighted graph. This can be done using Floyd algorithm with a complexity $O(|V|^3)$, see for instance Gondran and Minoux [14], Chapters 2 and 3. Let $M = \max_{v\in V}(0, -k(v))$. From the constructiveness, it follows that M has to be smaller than Γ_A. Now, the second step of the algorithm consists in creating at most $M|V|$ nodes and $M|E|$ arcs. The complexity of this step is at most $O(\Gamma_A|E|)$.
To obtain $\mathcal{R}_2$, we apply the `Forward Splitting` algorithm 6.4. In this algorithm, we create at most Γ_B nodes and $|E|$ arcs, which accounts for a complexity $O(\Gamma_B + |E|)$. □

Corollary 7.5 is interesting as it is not straightforward to extend the original algorithm of Leiserson and Saxe (for legal retimings, see § 6.5) to general retimings.

8 Application 2 : Task Graphs Evaluation

Task graphs are widely used in the modeling and analysis of parallel programs and architectures, see [1]. Yet, the performance evaluation of task graphs is difficult in general.
The term task graphs covers a wide variety of models, with the following common feature: each task depends on a finite number of tasks, and can be executed only when all the tasks it depends on are completed. Here, we consider *repetitive task graphs* which are bi-infinite task graphs generated by the periodic replication of a given finite task graph (with set of nodes V), see [3].
Let us denote by $X_i(n), i \in V, n \in \mathbb{Z}$, the epoch when the n-th occurence of task i is completed. Let the sets $\Delta_i, i \in V$, describe the dependences between tasks. The variables $X_i(n)$ are given by a recursion of the following form:

$$X_i(n) = \max_{(j,\gamma)\in\Delta_i} (X_j(n-\gamma) + \sigma_{j,i,\gamma}(n)),\ i \in V, n \in \mathbb{Z},\ \sigma_{j,i,\gamma}(n) \in \mathbb{R}\,. \qquad (13)$$

We will present the optimization problem which arises in the fast parallel computation of the evolution equations of task graphs, and apply the preceding results to solve it.

8.1 Max-Plus recurrences

The evolution equations (13) can be viewed as both a specialization and a generalization of a system of URE. On the one hand, the functions have a specific form, implying only the operations max and $+$. On the other hand, the functions depend on n. We call a "Max-Plus Recurrence" (MPR), an equation of the form (13). From now on, we assume that the MPR is *constructive* and *recycled*, i.e. that $\forall i, (i,1) \in \Delta_i$. It is a natural assumption for task graphs as it means that the n-th occurence of a task can not start before the completion of the $(n-1)$-th occurence of the same task.
The (max,+) formalism is a convenient tool to work with MPR. We briefly introduce it.

Definition 8.1. *The* (max,+) semiring $\mathbb{R}_{\max}$ *is the set* $\mathbb{R} \cup \{-\infty\}$*, equipped with the two operations* max *and* $+$*, denoted respectively by* $\oplus$ *and* $\otimes$ *(*$a \oplus b = \max(a,b)$ *and* $a \otimes b = a + b$*). The elements* $-\infty$ *and* 0 *are the neutral elements of the laws* $\oplus$ *and* $\otimes$ *respectively.*

For matrices of appropriate sizes, we define $(A \oplus B)_{ij} = A_{ij} \oplus B_{ij} = \max(A_{ij}, B_{ij})$, $(A \otimes B)_{ij} = \bigoplus_l A_{il} \otimes B_{lj} = \max_l (A_{il} + B_{lj})$, 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 (13) with the previously defined notations. Let $X(n)$ be the column vectors of coordinates $X_i(n)$ and let $A(\gamma, n)$ be the matrix with coordinates $A(\gamma, n)_{ij} = \sigma_{j,i,\gamma}(n)$ if $(j,\gamma) \in \Delta_i$ and $A(\gamma, n)_{ij} = -\infty$ otherwise. Now let $U = \{\gamma \mid \exists i, j \text{ s.t. } (j,\gamma) \in \Delta_i\}$ and $\overline{\gamma} = \max_U \gamma$. We have

$$X(n) = \bigoplus_{\gamma \in U} A(\gamma, n) \otimes X(n-\gamma) \, . \tag{14}$$

This is a *linear system* in the $(\max, +)$ semiring.

Representation of order one A standard step in the analysis of linear systems is the transformation of a recurrence like (14) into an "equivalent" system of order 1, such as (15), where $\tilde{X}(n), \tilde{X}(n-1) \in \mathbb{R}_{\max}^{\tilde{V}}$ and $A(n-1) \in \mathbb{R}_{\max}^{\tilde{V} \times \tilde{V}}$.

$$\tilde{X}(n) \;=\; A(n-1) \otimes \tilde{X}(n-1) \; . \tag{15}$$

The system in Equation (15) is "equivalent" to the original one if it preserves the functional behavior, meaning that $\forall i \in V$, $\exists u \in \tilde{V}$, $\exists c_i \in \mathbb{Z}$ s.t. $\forall n \in \mathbb{Z}$, $X_i(n) = \tilde{X}_u(n + c_i)$. We say that the system in (15) is an *order one representation* of the original one.
Assume that all the delays γ in (14) are greater or equal to 1. Then the transformation can be done by setting

$$\tilde{X}_{i|V|+j}(n) = X_j(n-i), \quad i = 0, \ldots, \overline{\gamma} - 1, \quad j \in V \, .$$

In this case, the dimension of the order 1 representation is $|\tilde{V}| = |V|\overline{\gamma}$. We will see below that an order 1 representation can be obtained without any assumption on the delays (except constructivity).
We can now define the main problem to be addressed in this section:

Problem 3 (MinSize). *Given a recycled MPR, find an equivalent MPR of order 1 and of minimal dimension.*

For strictly positive delays, it follows from the discussion above that the minimal dimension is at most equal to $|V|\overline{\gamma}$. We will see that in general it is much lower. Problem *MinSize* is very natural. A practical motivation for it is provided in next section.

Remark 8.2. In Problem *MinSize*, the optimality of the size of the 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 realization* in the sense of linear system theory [20, 10], which is normally smaller than ours. Finding a minimal realization is a difficult problem, and algorithms are known only in very specific cases. A deeper investigation of the relations between the two approaches is an interesting direction for further research.

8.2 Parallel evaluation of MPR

The *evaluation* of a MPR consists in computing all the variables $X(n)$. We assume that we want to perform this evaluation using a parallel machine. If we have an order 1 representation of the system, a possible and efficient algorithm is the *parallel prefix principle*: as the multiplication of matrices in (max,+) is associative, it is possible to divide the computation of $A(n) \otimes \cdots \otimes A(1)$ into smaller products $A(p) \otimes \cdots \otimes A(q)$ which may be computed by different processors.
The number of operations required to compute the variables up to $X(n)$ on a CREW-PRAM machine with P processors is $O(\ell^3(n/P + \log(P)))$, where ℓ is the size of the matrix of the linear system. Since n and P are fixed parameters, the complexity is minimized by having an order 1 representation of the MPR of minimal dimension.

8.3 Reduced graphs

As for any system of URE, we can associate a reduced graph $\mathcal{R} = (V, E, \Gamma)$ to a given MPR. To each node $i \in V$, we associate the sequences $\{\sigma_{j,i,\gamma}(n), n \in \mathbb{Z}\}$, $(j, \gamma) \in \Delta_i$.
We will show that the solution to Problem *MinSize* is a matrix of dimension $\min_r \Gamma_B(\mathcal{R}_r), r \in \mathbb{Z}^V$. This result seems to be new.

We transform any reduced graph by the following procedure. For each node in the reduced graph, we create new (dummy) nodes and new arcs in a tree-like fashion, as in Figure 17, such that each arc in the new reduced graph has a delay at most one. The added dummy nodes are recycled (the recyclings are not shown on the figure).
More formally, the transformation can be done using the algorithm `Forward Splitting-II` described below. We use the notation $u^\bullet$ to denote the set of successor arcs of u. It is not assumed that the delays are positive.

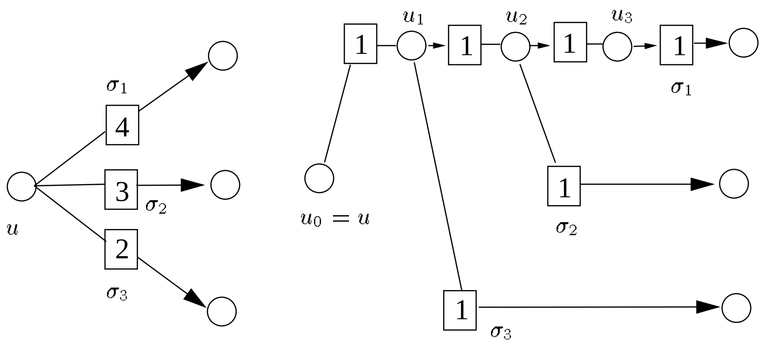


Figure 17: `Forward Splitting-II` of a reduced graph.

Algorithm 8.3 (Forward Splitting-II).
Input: *Recycled reduced graph* $\mathcal{R} = (V, E, \Gamma)$. *Sequences* $\{\sigma_{j,i,\gamma}(n), n \in \mathbb{Z}\}$.

1. `We set` $V' = V, E' = \emptyset$.
2. `For all node` $u \in V$, `let` $\delta(u) = \max_{a \in u^\bullet} \Gamma(a)$. `As the graph is recycled, we have` $\delta(u) \geqslant 1$.

   `If` $\delta(u) = 1$ `then` u `and` $u^\bullet$ `remain unchanged.`

   `If` $\delta(u) > 1$, `then`

   - `in` E', `set` $u_0 = u$ `and create` $\delta(u)-1$ `recycled nodes` $u_1, \cdots, u_{\delta(u)-1}$. `Create the arcs` $a_i = (u_i, u_{i+1})$ `with delay` $\Gamma'(a_i) = 1$ `for` $i = 0, \ldots, \delta(u) - 2$. `The sequences associated with nodes` $u_i, i > 0$ `are:` $\{\sigma'_{u_i,u_{i+1},1}(n), n \in \mathbb{Z}\} = 0$.
   - `For each arc` (u, v) `in` E `with delay` γ,
     - `if` $\gamma \leqslant 1$ `then create in` E' `the arc` (u_0, v_0) `with delay` γ `and sequence` $\{\sigma'_{u_0,v_0,\gamma}(n)\} = \{\sigma_{u,v,\gamma}(n)\}$
     - `else create in` E' `the arc` $(u_{\gamma-1}, v_0)$ `with delay 1 and sequence` $\{\sigma'_{u_0,v_{\gamma-1},1}(n)\} = \{\sigma_{u,v,\gamma}(n)\}$

Output: *Recycled reduced graph* $\mathcal{R}' = (V', E', \Gamma')$. *Sequences* $\{\sigma'_{u,v,\gamma}(n), n \in \mathbb{Z}\}$.

The new reduced graph has a maximum delay per arc equal to 1.

Proposition 8.4. *We use the notations defined in the above algorithm.*
(i)- The reduced graph $\mathcal{R}'$ *has the same functional behavior as the original graph* $\mathcal{R}$:

$$X'_{u_i}(n) = X_u(n - i), \forall u \in V, \forall n \in \mathbb{Z}\,. \tag{16}$$

(ii)- The number of nodes in $\mathcal{R}'$ *is* $|V'| = \Gamma_B(\mathcal{R})$.

Proof. Point (i) follows directly from the algorithm. Let us prove point (ii). Using the notations of Algorithm 8.3, in the new reduced graph, we have $\delta(u)$ nodes $(u_0, u_1, \cdots, u_{\delta(u)-1})$, for each node u in $\mathcal{R}$. The total number of node in $\mathcal{R}'$ is $\sum_{u \in \mathcal{R}} \delta(u) = \Gamma_B(\mathcal{R})$. $\square$

Proposition 8.4 is already an improvement over the standard representation as we obtain an order 1 MPR of dimension $\Gamma_B(\mathcal{R})$ instead of $\overline{\gamma}|V|$. Another improvement consists in finding first a retiming r of the graph such that $\Gamma_B(\mathcal{R}_r)$ is minimized.
To fix the notations, let $\mathcal{R} = (V, E, \Gamma)$ be the original reduced graph and $\mathcal{R}_1 = (V, E, \Gamma_1)$ be a retimed reduced graph minimizing Γ_B. We perform the `Forward Splitting-II` algorithm on $\mathcal{R}_1$ to get a new graph $\tilde{\mathcal{R}} = (\tilde{V}, \tilde{E}, \tilde{\Gamma})$ with all delays smaller or equal to one.
Some of the delays of $\tilde{\Gamma}$ might be negative. However, we prove that it is still possible to get an order 1 representation of dimension $|\tilde{V}|$ of the MPR associated with $\tilde{\mathcal{R}}$. We denote by $a^{\bullet}$ the ending node of an arc $a \in \tilde{E}$.

Lemma 8.5. *Let $\{\tilde{X}_i(n), i \in \tilde{V}, n \in \mathbb{Z}\}$ be the variables associated with $\tilde{\mathcal{R}}$. We have*

$$\tilde{X}(n) = B(n) \otimes \tilde{X}(n-1), \tag{17}$$

with $B_{ij}(n) = \max_{\pi \in \Pi_{j,i}} \sum_{a \in \pi \cap \tilde{E}} \sigma_{a,\tilde{\Gamma}(a)}(n - m(a, \pi))$, where $\Pi_{j,i}$ is the set of all the paths from j to i in $\tilde{\mathcal{R}}$ with total delay equal to one and $m(a, \pi)$ is the total delay on the path π from $a^{\bullet}$ to node i.

Proof. The first stage of the proof consists in showing that all paths ending in node i have a total delay at least 1 provided they are long enough. Let π be a path ending in i. The length (number of nodes) of π is denoted by $l(\pi)$. Let h be the sum of the negative delays in $\tilde{\mathcal{R}}$: $h = \sum_{a \in \tilde{E}} \min(0, \tilde{\Gamma}(a))$. Assume that $l(\pi) > (-h + 2)|\tilde{V}|$. Therefore, the path π must contain at least $-h + 2$ cycles. By constructivity, each cycle has a total delay which is strictly positive. The set of cycles contained in π is denoted $C(\pi)$. Then,

$$\begin{aligned} \tilde{\Gamma}(\pi) &= \sum_{p \in \pi} \tilde{\Gamma}(p) \\ &= \sum_{p \in C(\pi)} \tilde{\Gamma}(p) + \sum_{p \in \pi \setminus C(\pi)} \tilde{\Gamma}(p) \ \geqslant \ |C(\pi)| + h \ \geqslant 2\,. \end{aligned}$$

All the paths in $\Pi_{j,i}$ have a length smaller than $(-h + 2)|\tilde{V}|$. Since the graph $\tilde{\mathcal{R}}$ is finite, then $\Pi_{j,i}$ is a finite set.
Now, the equation on variables $\tilde{X}_i(n)$ in $\tilde{\mathcal{R}}$ can be written as

$$\begin{aligned} \tilde{X}_i(n) &= \max_{(j,\gamma) \in \tilde{\Delta}_i} (\tilde{X}_j(n - \gamma) + \tilde{\sigma}_{j,i,\gamma}(n)) \\ &= \max_{(j,\gamma) \in \tilde{\Delta}_i, \gamma = 1} (\tilde{X}_j(n-1) + \tilde{\sigma}_{j,i,1}(n)) \vee \max_{(j,\gamma) \in \tilde{\Delta}_i, \gamma \leqslant 0} (\tilde{X}_j(n - \gamma) + \tilde{\sigma}_{j,i,\gamma}(n)) \end{aligned}$$

In the latest equation, we replace all the variables $\tilde{X}_j(n - \gamma), \gamma \leqslant 0$, by their value until getting only variables of the type $\tilde{X}_j(n - \gamma), \gamma = 1$. By using the distributivity of $+$ with respect to max, we get Equation (17). $\square$

Using the results of the previous sections, and in particular Theorems 4.11 and 7.4 and Lemma 6.6, we obtain the following theorem.

Theorem 8.6. *Given a recycled MPR, its associated MPR of order 1 and of minimal size (Problem* MinSize*), has the same size as the minimal cut in the dependence graph of the MPR.*

Proof. We provide a sketch of the proof, which is an adaptation of the one of Theorem 7.4. There, we used in a critical way the existence of a *finite* set $\mathcal{W}$ of possible values for the variables in a digital circuit. In a MPR, the variables take their value in $\mathbb{R} \cup \{\infty\}$, but on the other end, the function involved are all (max,+)-linear. Hence, we adapt the argument of Theorem 7.4 as follows.
We have to prove that a (max,+) linear function from $\mathbb{R}^{|C|}$ to $\mathbb{R}^{|C|}$ ($|C|$ being the size of the minimal cut in the dependence graph $\mathcal{D}$) where the coefficients $\sigma_{ijk}(n)$ are arbitrary may not be further reduced.
Let $C = \{(i_1, n_1), \cdots, (i_{|C|}, n_{|C|})\}$ be a minimal cut in $\mathcal{D}$ and let $\{X_{i_j}(n_j), j = 1, \cdots, |C|\}$ be the corresponding set of variables. Now, choose an integer k large enough such that C and $C+k$ do not share any node. There exists $|C|$ node-disjoint paths in $\mathcal{D}$ from C to $C+k$. Let $\mathcal{F}_C$ be the flow constituted by these paths. On each one of these paths, choose all the σ variables to be equal to 0. On all the arcs between C and $C+k$ which do not belong to those paths, set the σ variable to be smaller than $-\max_{j,k} |X_{i_j}(n_j) - X_{i_k}(n_k)|$. Let $(i_l, n_l + k)$ be the successor on $(C+k)$ of (i_j, n_j) following the flow $\mathcal{F}_C$. We have $X_{i_l}(n_l + k) = X_{i_j}(n_j)$.
The set $\{X_{i_j}(n_j + k), j = 1, \cdots, |C|\}$ is formed by independent variables for all k and cannot be reduced. The corresponding matrices $B(n)$ are permutation matrices (i.e. there exists a permutation σ such that $B_{\sigma(j),j} = 0$ and $B_{ij} = -\infty$ for $i \neq \sigma(j)$). The remainder of the argument follows Theorem 7.4. □

8.4 Summary and complexity

A summary of the algorithm to find a solution to problem *MinSize* is given by the following scheme.

$$\begin{array}{ccccccccc}
 & \text{reduced} & & \text{optimal} & & \text{forward} & & & \\
 & \text{graph} & & \text{retiming} & & \text{splitting II} & & \text{Lemma 8.5} & \\
\text{MPR} & \longrightarrow & \mathcal{R} & \longrightarrow & \mathcal{R}_1 & \longrightarrow & \mathcal{R}_2 & \longrightarrow & \text{MPR'} \\
\text{order } k & & & & \Gamma(\mathcal{R}_1) \in \mathbb{Z}^{E_1} & & \Gamma(\mathcal{R}_2) \leqslant 1 & & \text{order } 1 \\
\text{dim. } |V| & & & & \Gamma_B(\mathcal{R}_1) \text{ minimal} & & |V_2| = \Gamma_B(\mathcal{R}_1) & & \text{dim. } |V_2|
\end{array}$$

Corollary 8.7. *Let us consider a recycled MPR. An order 1 representation solving Problem MinSize can be obtained with an algorithm of polynomial complexity to obtain graph $\mathcal{R}_2$ and pseudo-polynomial complexity to construct matrix $B(n)$.*

Proof. This is a sketch of the proof. Let $\mathcal{R} = (V, E, \Gamma)$. The graph $\mathcal{R}_1$ is obtained using the algorithm of Proposition 4.12 whose complexity is $O(\Gamma_A(\mathcal{R})^2 |V|^2)$. The graph $\mathcal{R}_3$ is obtained by applying the `Forward Splitting-II` Algorithm. In this algorithm, we create at most $\Gamma_B(\mathcal{R}_2)$ nodes and $\Gamma_B(\mathcal{R}_2) + |E|$ arcs. Its complexity is at most $\Gamma_B(\mathcal{R}_2) + |E|$. The computation of an element of matrix $B(n)$ defined in Lemma 8.5 is NP-complete (reduction of knapsack with multiplicities). However, it can be obtained with pseudo-polynomial complexity using dynamic programming techniques (similar to knapsack). □

Given an initial reduced graph $\mathcal{R}$, Problem *MinReg* in § 7 and Problem *MinSize* in § 8 are solved using the same graph $\tilde{\mathcal{R}}$, which is the retimed graph of $\mathcal{R}$ minimizing Γ_B. However, the exact solutions of Problems *MinReg* and *MinSize* are obtained by performing two different type of transformations on $\tilde{\mathcal{R}}$, yielding two different graphs, say $\mathcal{R}_{\text{reg}}$ and $\mathcal{R}_{\text{size}}$.
In general $\mathcal{R}_{\text{reg}}$ does not provide a solution to Problem *MinSize* (it does not have a minimal number of nodes) and $\mathcal{R}_{\text{size}}$ does not provide a solution to Problem *MinReg* (it does not have a minimal number of registers). To check this, consider for instance the graph of Figure 12 and apply the `Forward Splitting-II` algorithm to it. We conclude that the problems *MinReg* and *MinSize* are different, although related.

8.5 Event graphs

Event Graphs, a subclass of *Petri nets*, are commonly used to model Discrete Event Systems, see [2, 19]. The graphical formalism for Event Graphs is close but different from the one of reduced graphs, see Figure 18.

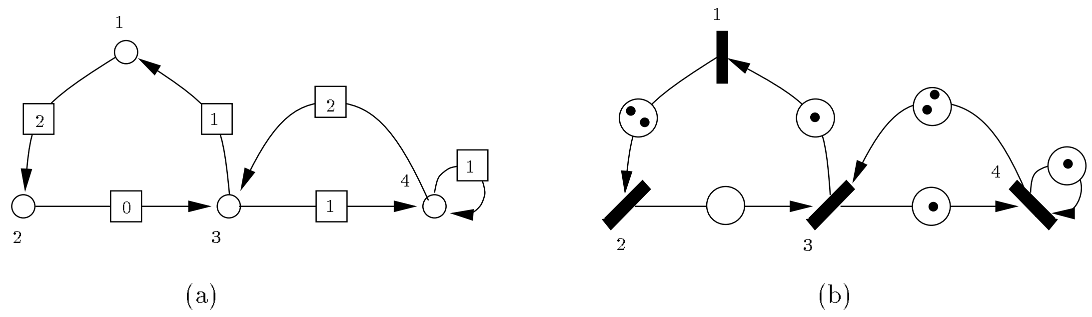


Figure 18: Transformation of a reduced graph (a) into a Petri net (b).

In the case of a Timed Recycled Event Graph, the dynamic can be represented by a Max-Plus Recurrence like in (13) with $\sigma_{i,j,\gamma}(n) = \phi_i(n) + h_{i,j}$, $\phi_i(n) \geqslant 0, h_{i,j} \geqslant 0$. Since the variables $\sigma_{i,j,\gamma}(n)$ are not general, the MPR of order one given by Theorem 8.6 may not be minimal. For example, the removal of *implicit places* may reduce the dimension of an order 1 representation. Let us also mention that preliminary results on the Problem *MinSize* for Event Graphs were proved in [4, 11, 13], precisely the existence of an order 1 representation of dimension $\Gamma_A(\mathcal{R})$ $(\geqslant \Gamma_B(\mathcal{R}))$.

9 Conclusion and Perspectives

Let us summarize the main results obtained. We restricted our attention to recycled systems of URE. We proved that the optimal solutions for Problem *MinPeb* are the same for games $\mathcal{M}_1$ and $\mathcal{M}_2$, Theorem 4.11. On the other hand, an optimal solution for game $\mathcal{M}_3$ may be strictly bigger than one for the games $\mathcal{M}_1$ and $\mathcal{M}_2$, see the example of Figure 10.
There exists polynomial algorithms to solve Problem *MinPeb* for the games $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$, see § 4.3 and § 6.5. These results have been applied to minimize the number of registers in a digital circuit and to minimize the size of a (max,+) recurrence.
To complete the picture, it would be nice to extend all the previous results to the non-recycled case. The key results which would make everything else easy to generalize are of two types.

Results related to cuts, see § 4.1, and results linking cuts and regular configurations, see § 4.2. For example, is it possible to find a minimal cut which is consecutive (generalization of Lemma 4.7)? Can we find a minimal consecutive cut which is a regular configuration (generalization of Lemma 4.9)? We are currently investigating these different issues.

Acknowledgment The authors would like to thank Jean-Claude Bermond, Alain Darte, Stéphane Gaubert, Stéphane Perennes and Mike Robson for numerous discussions and suggestions.

References

[1] V. Adlakha and V. Kulkarni. A classified bibliography of research on stochastic PERT networks: 1966-1987. Technical report, Dept. of Op. Res., University of North Carolina, Chapel Hill, 1987.

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

[3] F. Baccelli, A. Jean-Marie, and Z. Liu. A survey on solution methods for task graph models. In Götz, Herzog, and Rettelbach editor, *Proceedings of the 2nd QMIPS Workshop*, Univ. of Erlangen, 1993.

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

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

[6] G. Cohen, D. Dubois, J.P. Quadrat, and M. Viot. A linear system-theoretic view of discrete-event processes and its use for performance evaluation in manufacturing. *IEEE Trans. Automatic Control*, AC-30:210–220, 1985.

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

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

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

[10] S. Gaubert, P. Butkovič, and R. Cuninghame-Green. Minimal (max,+) realization of convex sequences. *SIAM J. on Control and Opt.*, 36:137–147, 1998.

[11] B. Gaujal. *Parallélisme et simulation de systèmes à événements discrets*. PhD thesis, Université de Nice-Sophia Antipolis, 1994.

[12] B. Gaujal, A. Jean-Marie, and J. Mairesse. Minimal representation of uniform recurrence equations. Technical Report RR-2568, INRIA, Sophia-Antipolis, France, 1995.

[13] B. Gaujal and A. Jean Marie. Computational issues in stochastic recursive systems. In J. Gunawardena, editor, *Idempotency*, pages 209–230. Cambridge University Press, 1998.

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

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

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

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

[18] J. Mairesse. *Stabilité des systèmes à événements discrets stochastiques. Approche algébrique.* PhD thesis, Ecole Polytechnique, Paris, 1995. In english.

[19] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[20] G.J. Olsder. On the characteristic equations and minimal realizations for discrete-event dynamic systems. In *Proc. 7-th Int. Conf. on Anal. and Optim. of Systems*, volume 83 of *LNCIS*, pages 189–201, San Jose, USA, 1986. Springer.

[21] N. Pippenger. Advances in pebbling. In *Proc. ICALP*, number 140 in LNCS, pages 407–417. Springer Verlag, 1982.

[22] J.H. Reif, editor. *Synthesis of Parallel Algorithms.* Morgan Kaufmann Publ., 1993.

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

[24] N. Shenoy and R. Rudell. Efficient implementation of retiming. *Proc. ICCAD 1994*, San Jose, USA, 1994.

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

A Appendix: Scheduling Problems

The problem of organizing efficient computations for URE on parallel computers has been considered by several authors. However, the investigations have often been oriented towards speeding up the execution with no or little consideration for memory requirements. We quickly describe the main results in this area as a way to put our approach in perspective.

A.1 Definition of a schedule

Assume that at time 0, the strictly negative variables $X_i(n), n < 0$, are known. Assume also that each computation of a variable is done in one time unit.

Definition A.1 (schedule). *We define a* schedule *as a set of instants* $\{t_i(n), i \in V, n \in \mathbb{N}\}$ *such that* $t_i(n) \geqslant t_j(n-\gamma)+1$ *if* $(j,\gamma) \in \Delta_i$ *(with the convention* $t_i(n) = 0, \forall i \in V, \forall n < 0$*).*

The instant $t_i(n)$ is necessarily larger than the length (number of arcs) of a longest path in $\mathcal{D}$ from column -1 to (i,n). A schedule is said to be *as soon as possible* ('asap') if $t_i(n)$ is exactly equal to the length of the longest path from column -1 to (i,n).
A first question that has been 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-cliques (instead as the minimal cuts of the present paper) in the dependence graph.
Once this question is settled and provided a sufficient number of processors is available (*i.e.* larger than the degree of parallelism of the system), another problem is to characterize the 'asap' schedule. This is often called the *basic scheduling problem.* For the 'asap' schedule, it has been proved, see [5, 6, 2], that:

$$\exists N, \forall n \geqslant N,\ t_i(n) = \lambda_i n + d_i(n)\ , \tag{18}$$

where $\lambda_i \in \mathbb{R}^+$ and $d_i(n), n \geqslant N$, is a periodic real function. The real $\max_i \lambda_i$ is called the *cycle time* of the system. A schedule satisfying (18) is said to be *linear.* For systems of higher dimension (i.e. when $K = \mathbb{Z}^p,\ p > 1$, in Equation (1)), there are some partial results on how to approximate the as soon as possible schedule using linear schedules, see [7, 8].
When the number of available processors is fixed and less than the degree of parallelism of the URE, finding an optimal schedule (i.e. a schedule such that $\lim_n \max_i t_i(n)/n$ is minimal) becomes NP-hard, see [15].

All the results mentioned above are more or less related to the problem of minimizing the number of processors used. On the other hand, in this paper, we considered the dual problem: how many memory locations are necessary to carry out the computation of an URE, the number of processors being unlimited.
First, we should say that, in general, a computation 'asap' requires a lot of memory. It may not even be bounded when the reduced graph $\mathcal{R}$ is connected but not strongly connected. This makes the alternative of 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.

We have shown in the previous sections how to obtain executions using a minimal memory size. In general the schedules associated with these minimal-memory executions are not 'asap'. A natural question to ask is whether they are very far from the 'asap' execution or not.
For minimal-memory executions, the number of processors needed to carry out the computations will in general be greater than the degree of parallelism. A second natural question is whether it is very far from the degree of parallelism or not. These two questions are now investigated.

A.2 Number of processors and linearity of the schedule

In games $\mathcal{M}_1, \mathcal{M}_2$ and $\mathcal{M}_3$, rules (R2), (R2b) or (R2c) do not limit *a priori* the number of moves of type (R3) which are feasible in one step. It corresponds to a computational model where the number of processors available is not limited. However, one can notice *a posteriori* that the total number of processors needed in the computation is bounded by the maximal number of pebbles $\mathcal{P}(e)$ used during the execution e. Hence, the number of processors needed to implement an execution solving Problem *MinPeb* is kept under control.

In mimimal-memory executions, in a single step, we have to compose several of the functions F_i defining the URE to compute the new variables, see § 3.1.
To take into account the 'cost' of function composition, we give to each step a duration. We assume that step t lasts $l(t)$ units of time, where $l(t)$ is the length of the longest path in $\mathcal{D}$ joining a node of $\mathcal{A}(t-1)$ to a node of $\mathcal{A}(t)$ and containing no other node in $\mathcal{A}(t-1)$. Note that $l(t)$ is also the length of the longest chain of function compositions performed during step t. This is consistent with the assumption that each function computation requires one unit of time. With this convention, we can now define the schedule associated with a synchronous execution.

Definition A.2. *Let $e = \{\mathcal{A}(t), t \in \mathbb{N}\}$ be an execution of game $\mathcal{M}_1$ (or $\mathcal{M}_2$ or $\mathcal{M}_3$), and let $l(t), t \in \mathbb{N}$, be the time duration of step t. Then the* schedule *of e, $\{\tau_i(n), i \in V, n \in \mathbb{N}\}$ is defined by*

$$\tau_i(n) = \sum_{t=0}^{T} l(t), \;\; \textit{where } T = \inf\{t \mid (i,n) \in \mathcal{A}(t)\} .$$

One verifies easily that this is indeed a schedule according to Definition A.1. Let us justify Definition A.2. Within one step, the computations are done in parallel. The duration of one step is the one of the longest computation done in the step. After each step, there is a synchronization barrier which makes the duration of the execution to be the sum of the durations of the steps.

One important point is that $\tau_i(n)$ denotes a time instant and not a step of the game. Indeed, there might be a big difference between the time instant and the step at which nodes are computed. Here is an illustration for game $\mathcal{M}_1$ (see also the discussion following Figure 4 in § 3.1). The initial pebbles remain untouched through the whole execution. An additional pebble is used to mark successively all the positive nodes. In such a case, marking a node on column n takes one step and $\Omega(n)$ units of time. Marking all the nodes up to column n requires $\Omega(n)$ steps and $\Omega(n^2)$ units of time. The schedule associated with this execution is *quadratic*.

We proved that minimal-memory executions can always be choosen to be regular, Section § 4. It implies that all steps have a constant duration $l(t) = l$. Moreover, exactly one new variable is computed on each line of $\mathcal{D}$ at each step. It implies that the schedule is linear with cycle time l. Therefore, the linear schedules are dominant for our problem. We conclude that the loss in time efficiency, when compared with 'asap' executions, is kept under control.
Among regular executions, one-pass regular executions (game $\mathcal{M}_3$) are dominant in terms of execution time, since they never "lose" time by computing the same variable twice. However, as seen in Section § 6, they may require more memory than general regular executions.

To summarize, when compared with 'asap' executions, minimal-memory executions may lose a bit in terms of numbers of processors needed and execution speed but may gain a lot in terms of memory size. They provide new insights on the trade-offs between time and space in the computation of a system of URE.

B Appendix: Sequential Executions

In Section § 3.1, we introduced different variants of pebble games. All these games allow simultaneous moves of the pebbles. Here is another one, defined by rule $\mathcal{M}_4$, which is close to the ones defined in [21, 23]. Here, pebbles are put on the nodes one at a time when all direct predecessors already have a pebble.

B.1 Definition of a sequential game

$\mathcal{M}_4$: Sequential Execution Rules

- (R1) *(initial position)* $\mathcal{A}(0) \subset V \times \mathbb{Z}^-, \quad \mathcal{A}(0) \cap (V \times \{0\}) \neq \emptyset$;
- (R2d) *(playing rule)* one step of the game consists in using rule (R3b) at most once followed by any number of moves of type (R4);
- (R3b) *(adding pebbles)* a pebble can be put on an empty node if all the direct predecessors of this node have a pebble;
- (R4) *(removing pebbles)* remove a pebble from a node.

Let us consider the example of Figure 19. It corresponds to the same URE as in Figure 4.

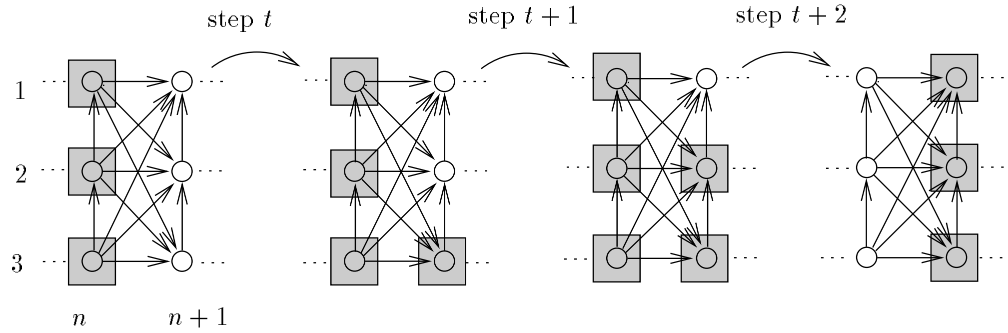


Figure 19: Sequential rule $\mathcal{M}_4$. Five pebbles are needed.

Here is a possible execution. At step 0, there are pebbles on the nodes $(1,0),(2,0)$ and $(3,0)$, so that (R1) is satisfied. After step t, suppose that there are three pebbles on the nodes $(1,n),(2,n)$ and $(3,n)$ respectively. At step $t+1$, we can put a pebble on node $(1,n+1)$ since all its predecessors (i,n) have a pebble (rule (R3b)). At step $t+2$, we 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) since they are not needed anymore (rule (R3b) then rule (R4) three times). Then, we reiterate the previous three steps.
It is impossible to use fewer pebbles. The minimal number of pebbles required to compute the graph under rule $\mathcal{M}_4$ is five, to be compared with the three pebbles needed under rules $\mathcal{M}_1$ or $\mathcal{M}_2$.

As for the computational model of the game, $\mathcal{M}_4$ corresponds to performing sequential computations. It is relevant if we want to model the computation of the URE using a sequential computer which has a single processor and makes a single computation at each step. The removal of pebbles being a "passive" operation, several such manipulations are allowed in a single step (the data is not actually removed from the memory, it will just be overwritten the next time this memory location is used).

In game $\mathcal{M}_4$, a single processor is used. Furthermore, any reasonable execution will compute exactly one new node at each step. Hence, the associated schedule is linear with cycle time $|V|$.

Remark B.1. Game $\mathcal{M}_4$ can be adapted to be played on a finite binary tree (for more details, see the proof of Proposition B.2). In this case, the minimal number of pebbles is known as the Strahler's number. This number appears in various fields ranging from hydrology and combinatorics to molecular biology. For a nice survey paper, the reader is referred to Viennot [25].

B.2 Complexity results for $\mathcal{M}_4$

Under game $\mathcal{M}_4$, the problem of determining the minimal number of pebbles to compute a general directed acyclic graph (Problem *MinPeb-DAG*) was proved to be NP-complete by Sethi [23]. We prove a similar result for the Problem *MinPeb*, by showing that an instance of *MinPeb-DAG* can be embedded into an instance of *MinPeb*.

Proposition B.2. *Let $\mathcal{D}$ be the dependence graph associated with a system of URE. Solving Problem* MinPeb *for game $\mathcal{M}_4$ is NP-hard. For the subclass of recycled URE, the same problem is still NP-hard.*

Proof. First we need to define more precisely what is the pebble game $\mathcal{M}_4$ on a DAG. Let V be the set of nodes of a DAG $\mathcal{G}$. A configuration is a subset of V and an execution is a finite sequence of configurations $\{\mathcal{A}_{\mathcal{G}}(t), t=0,\dots,L\}$ (where L is the length of the execution). An execution is successful if all the nodes V receive a pebble during the execution. Rules (R2d), (R3b) and (R4) remain the same. Rule (R1) is modified as follows: (R1b) (initial and final position) $\mathcal{A}_{\mathcal{G}}(0)=\mathcal{A}_{\mathcal{G}}(L)=\emptyset$.
Given a DAG $\mathcal{G}$ with set of nodes V, we construct a dependence graph $\mathcal{D}$ with set of nodes $V\times\mathbb{Z}$ in the following way. If there is an arc between nodes i and j in $\mathcal{G}$, then for all $k\in\mathbb{Z}$, we put an arc between nodes (i,k) and (j,k) and an arc between nodes (i,k) and $(j,k+1)$. We also add the recycling arcs in $\mathcal{D}$ ($(i,k)\rightarrow(i,k+1), \forall i\in V, \forall k\in\mathbb{Z}$). An example is given in Figure 20.

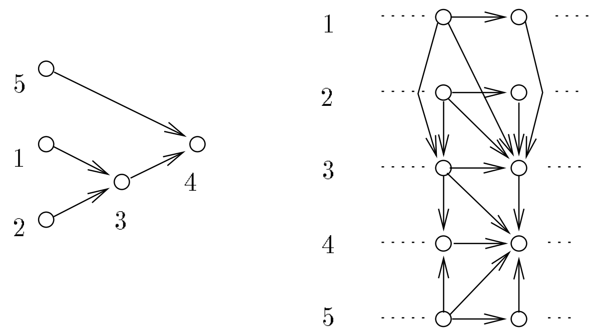


Figure 20: Embedding of an arbitrary DAG in a dependence graph.

Starting from an execution of the pebble game on $\mathcal{G}$, $\{\mathcal{A}_{\mathcal{G}}(t), t = 0, \ldots, L\}$, we construct an execution of the pebble game on $\mathcal{D}$, $\{\mathcal{A}(t), t \in \mathbb{N}\}$, as follows. The initial configuration is $\mathcal{A}(0) = \{(i,0), i \in V\}$. When a pebble is added on node i in $\mathcal{G}$, we add a pebble on node $(i,1)$ in $\mathcal{D}$. When a pebble is removed from node i in $\mathcal{G}$, we remove a pebble from node $(i,0)$ in $\mathcal{D}$. It follows that at step L, we have $\mathcal{A}(L) = \{(i,1), i \in V\}$. Then, we complete the execution $\{\mathcal{A}(t)\}$ by repeating the same steps periodically, i.e. $\mathcal{A}(t+kL) = \mathcal{A}(t) + k, \forall t, k \in \mathbb{N}$ (see (6)). If the execution in $\mathcal{G}$ uses p pebbles, the execution in $\mathcal{D}$ requires $p + |V|$ pebbles.
Conversely, from an execution of the game on $\mathcal{D}$, we construct an execution of the game on $\mathcal{G}$ in the following way. By definition, we have $\mathcal{A}(0) \subset \{V \times \mathbb{Z}^-\}$. We set $\mathcal{A}_{\mathcal{G}}(0) = \emptyset$. When a pebble is added on node $(i,1)$, a pebble is added on node i. When a pebble is removed from node $(i,0)$, a pebble is removed from node i. If the execution in $\mathcal{D}$ uses P pebbles, the execution in $\mathcal{G}$ requires less than $P - |V|$ pebbles.
One can see that the above transformations map optimal executions into optimal executions (optimal in the sense of Problems *MinPeb* and *MinPeb-DAG*). Hence, the minimal number of pebbles needed in $\mathcal{D}$ is the minimal number of pebbles needed in $\mathcal{G}$ plus the number of nodes of $\mathcal{G}$. Therefore, the NP-completeness of the problem for DAG, Sethi [23], implies that the problem for dependence graphs is at least NP-complete, i.e. NP-hard. □

Proposition B.2 contrasts with the results proved for games $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$, for which a minimal-memory execution was obtained in polynomial time, in the recycled case. See § 4.3 (games $\mathcal{M}_1$ and $\mathcal{M}_2$) and § 6.5 (game $\mathcal{M}_3$).

Remark B.3. Let us consider the dependence graph of Figure 20. The execution described in the proof of Proposition B.2 solves Problem *MinPeb* with a number of pebbles which evolves as: $5, 7, 6, 7, 6, 5, \ldots$. The number of pebbles used during an optimal execution is not always constant. Hence, if we consider a non-connected URE, it is not true that the minimal number of pebbles needed is equal to the sum of the minimal number of pebbles needed for each connected component independently. This contrasts with the situation for games $\mathcal{M}_1, \mathcal{M}_2$ and $\mathcal{M}_3$, see § 2.4.