Memory Management Optimization Problems for Integrated Circuit Simulators

Timothée Bossart [a] Alix Munier Kordon [a] Francis Sourd [a]

[a] *LIP6 - Université Pierre et Marie Curie,*
4 place Jussieu, 75252 Paris Cedex 05, France
{*timothee.bossart,alix.munier,francis.sourd*}*@lip6.fr*

Abstract

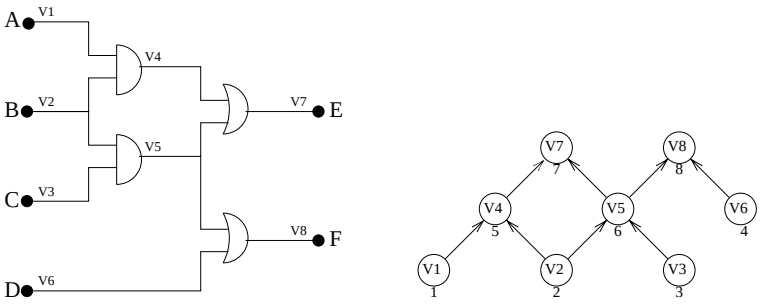
In hardware design, it is necessary to simulate the anticipated behavior of the integrated circuit before it is actually cast in silicon. As simulation procedures are long due to the great number of tests to be performed, optimization of the simulation code is of prime importance. This paper describes two mathematical models for the minimization of the memory access times for a cycle-based simulator.

An integrated circuit being viewed as a directed acyclic graph, the problem consists in building a graph order on the vertices, compatible with the relation order induced by the graph, in order to minimize a cost function that represents the memory access time. For the two proposed cost functions, we show that the two corresponding problems are NP-complete. However, we show that the special cases where the graphs are in trees or out trees can be solved in polynomial time.

Key words: graph ordering, integrated circuit simulation, complexity.

1 Introduction

Simulation is a crucial challenge for the design of integrated circuits [13]. In fact, the task involves the iteration of a design and simulation process before tests on real chips are possible. In very few words, a simulator can be viewed as a computer program that reads in input the physical description of an integrated circuit — a VHDL file for example — and produces, in a so-called *compilation phase*, an *executable simulation code* that simulates the behavior of the circuit. Then, the *test phase* consists in running the executable code on a large number of benchmarks. Each benchmark consists of *input data* and *output data*: the executable code is given the input data of the benchmark and produces output data which is compared to the theoretical output of the benchmark. If produced and theoretical output data are different, the test fails,



(a) (b)

$V_1 \longleftarrow A$
$V_2 \longleftarrow B$
$V_3 \longleftarrow C$
$V_6 \longleftarrow D$
$V_4 \longleftarrow \mathrm{AND}(V_1, V_2)$
$V_5 \longleftarrow \mathrm{AND}(V_2, V_3)$
$V_7 \longleftarrow \mathrm{OR}(V_4, V_5)$
$V_8 \longleftarrow \mathrm{OR}(V_5, V_6)$
$E \longleftarrow V_7$
$F \longleftarrow V_8$

(c)

Fig. 1. Code generation for the simulation of a circuit

which means the circuit is not correct. As the executable code that simulates the circuit is run a very large number of times — some test phases may last several days — improving the compilation phase so that the produced code runs faster is of practical interest to significantly reduce the length of the test phase. In this paper, we propose two theoretical graph problems to model the optimization of the code production.

In this paper, an integrated circuit will be seen as a set of *logical gates* (such as AND, OR, NOT...) interconnected through wires represented through a directed acyclic graph (see Figure 1(a) and 1(b)). The value of the output of a gate is directly derived from its inputs so that the role of the code simulating the circuit is to sequentially compute the values of all the wires in order to compute the output. Clearly, since all the inputs must be computed in order to compute the output, the values of the wires must be computed in a topological order induced by the digraph (see Figure 1(c)). Conversely, any topological order of the digraph yields a different code so that our problem is to find a topological order that produces the fastest code. The main difficulty in building the model is to find an estimate for the speed of the code.

Therefore, the generated code has a very special structure. On the one hand, there is no loop nor branching instructions. On the other hand, there are a great number of instructions and of variables to deal with in each block of code. Conventional compilers, such as `gcc`, are not devised to optimize such a code. In fact, given the great number of variables induced by a large integrated circuit, memory management is of key importance in order to use different cache levels at best. So the two models proposed in Section 2 aims at minimizing total memory access time.

As the problem is to specify an order of the vertices of a graph, it is closely related to *graph layout problems* whose goal is to number the vertices of an input graph in such a way that a given objective function is optimized. The reader is referred to the recent survey by Díaz et al. [7] for a state of the art of these problems which are also referred to as graph (linear) ordering, (linear) arrangement, numbering or labeling problems. These problems are known to be very useful to optimize the processing of large data: for example, "bandwidth (minimization) had received much attention during the fifties in order to speed up several computations on sparse matrices" [7]. However, most research was devoted to non oriented graphs. Our model is based on a directed graph. For such graphs, bandwidth, cut width and linear arrangement problem are known to be NP-complete [11,9]. Approximation algorithms were proposed by Even et al. [8] and improved by Rao and Richa [14]. Detti and Pacciarelli proposed a branch-and-bound algorithm for a generalization of the directed linear arrangement problem [5].

We show in Section 2 that one of the two models we proposed can be seen as the minimization of the Directed Sum Cut, which generalized the Sum Cut, an objective function that has been studied in the context of non-oriented graphs [6]. To the best of our knowledge, the oriented version of this problem has never been studied. We prove the problem is NP-complete and we present to polynomial algorithms for in trees and out trees.

Our second model was initially born of the Register Allocation problem [16]. In [4], the problem of code minimization for a k-register machine was proved to be NP-complete for $k = 1$. Our model, called Uniform Cost Stack, is derived from this model because it relies on a similar set of operators for memory access. The main difference is that the memory is modeled by a stack structure, and the cost function is linear. We prove that the problem is NP-complete even for graphs of height at most 1, along again with two polynomial cases (in tree and out tree).

The two models are introduced in Section 2. Section 3 is devoted to the theoretical study of the Directed Sum Cut and Section 4 presents analogous theoretical results — but proofs are very different — for the Uniform Cost Stack model. In conclusion, some insights into the practical relevance of the

two models are presented.

2 Models

This section mathematically introduces and discusses two combinatorial optimization problems that model the cache optimization problem for the simulation of circuits.

Let $G = (V, A)$ be a directed acyclic graph, it represents the dependence between the variables of the simulation code. The number of nodes of G is denoted by $n = |V|$ and the number of arcs is denoted by $m = |A|$. For each $u \in V$, $\Gamma^+(u)$ (*resp.* $\Gamma^-(u)$) is the set of successors (*resp.* predecessors) of u and let $\delta^+(u) = |\Gamma^+(u)|$ and $\delta^-(u) = |\Gamma^-(u)|$ denote the out- and in-degrees. The set of *source nodes* (*i.e.* nodes u such as $\Gamma^-(u) = \emptyset$) is denoted S.

Since any possible code is represented by a numbering of the nodes of G, the *feasible solutions* of the problem are formally described by a bijection φ that maps V to $\{1, \cdots, n\}$ and satisfies the constraint $\forall a = (u, v) \in A, \varphi(u) < \varphi(v)$. φ is called a *graph ordering function* or a *graph order*. We will often use the notation $\varphi^-1(i)$ for some $i \in \{1, \cdots, n\}$ to refer to the node whose rank is i in the order φ.

Given a graph ordering function φ and an arc $(u, v) \in A$, let $\mathcal{C}(\varphi, (u, v))$ be the loading cost of the variable u for the evaluation of v. This function will depends on the cache model considered. Then, the total loading cost of a variable $u \in A$ is defined as :

$$\mathcal{C}(\varphi, u) = \sum_{v \in \Gamma^+(u)} \mathcal{C}(\varphi, (u, v))$$

The total cost of the order φ on G will then be defined in the following way:

$$\mathcal{C}(\varphi, G) = \sum_{u \in V} \mathcal{C}(\varphi, u)$$

The problem is to find the order φ that minimizes this objective function. In the rest of this section, two models are proposed to evaluate by two different ways the cache access costs $\mathcal{C}(\varphi, u)$. In both these models, $\mathcal{C}(\varphi, u)$ is a deterministic function that only depends on φ and u. This hypothesis can be criticized because cache policies may be randomized and the real access times depends of numerous other parameters such as the cache size, the operating system (and its settings), the memory state when the simulation code is run, the programs that are concurrently run and many others. However, as the simulation code is run many times — eventually on different machines — we are

interested in minimizing its average running time. So, we are going to assume that the deterministic value $\mathcal{C}(\varphi, u)$ represents the *average* cache access cost.

We observe that the expression of the total cost $\mathcal{C}(\varphi, G)$ deliberately ignores the time spent at computing the value of the output of the logical gate once the input are read. In fact, this time is assumed to be constant. So, the total time for running the simulation code is the sum of a constant computation time and a cache access time depending on φ. Only this second value is minimized.

In the first model, the estimation of $\mathcal{C}(\varphi, u)$ is based on the number of instructions executed between to successive use of the variable u. The second model is more complex as it keeps track of all the memory moves.

2.1 Directed Sum Cut

This first model is based on the observation that φ induces a numbering of the lines of the simulation code (see Figure 1(c)): $\varphi(v_i)$ is the number of the line at which v_i is created. In order to introduce the cost function, we consider the use of some variable u after its creation. The first access is made in order to compute the first successor of u w.r.t. φ, which is denoted by $s_1(\varphi, u)$ or simply by $s_1(u)$. The cost for reading u is assumed to be a function of the number of instructions executed since the creation of u, that is $\mathcal{C}(\varphi, (u, s_1(u))) = f(\varphi(s_1(u)) - \varphi(u))$. As discussed before, f is assumed to be a deterministic function, which is non-decreasing to model the fact that a variable that has been in memory for a longer time takes longer to be read.

After this computation, both variables u and $s_1(u)$ are supposed to be equivalently cached in memory. So, when u is accessed by its second successor $s_2(u)$, the access cost is $\mathcal{C}(\varphi, (u, s_2(u))) = f(\varphi(s_2(u)) - \varphi(s_1(u)))$ since $\varphi(s_2(u)) - \varphi(s_1(u))$ is the number of instructions executed since the creation of $s_1(u)$. Therefore, the total cost related to the access to variable u is

$$\mathcal{C}(\varphi, u) = \sum_{i=1}^{\delta^+(u)} f(\varphi(s_i(u)) - \varphi(s_{i-1}(u))), \text{ if } \delta^+(u) \geq 1, \text{ and 0 else}$$

where $\delta^+(u)$ is the out-degree of u in G, $s_0(u) = u$ and $s_1(u), s_2(u), \cdots$ are the direct successors of u numbered w.r.t. φ. In order to simplify the model, we will consider that the cache access cost function is simply the identity function Id. The choice of such a simple function for f is motivated by the fact that there is no hardware- or software-dependent parameters. Furthermore, this choice yields an interestingly simple expression for the total cache access cost

$$\mathcal{C}(\varphi, u) = \left(\max_{(u,v) \in A} \varphi(v) \right) - \varphi(u) \tag{1}$$

and for the objective function of the problem

$$\mathcal{C}(\varphi) = \mathcal{C}(\varphi, G) = \sum_{u \in V} \left(\left(\max_{(u,v) \in A} \varphi(v) \right) - \varphi(u) \right).$$

The following theorem shows that the expression of this objective function can be linked to a classical criterion in graph layout problems. Namely, the *vertex cut* at position i, denoted by $\delta(i, \varphi, G)$, is defined as $|\{u \in V : \varphi(u) \leq i \wedge (\exists v : \varphi(v) > i \wedge (u, v) \in A)\}|$ [7]. It represents the number of vertices numbered before i that have at least one successor v numbered after i. In terms of memory management, the interpretation of $u \notin \{u \in V : \varphi(u) \leq i \wedge (\exists v : \varphi(v) > i \wedge (u, v) \in A)\}$ is the following: either u is not used anymore, or has not been created yet.

Theorem 1 *We have the equality*

$$\mathcal{C}(\varphi, G) = \mathrm{DSC}(\varphi, G)$$

where DSC *is the* Directed Sum Cut *of G ordered by φ and is defined as* $\mathrm{DSC}(\varphi, G) = \sum_{1 \leq i \leq n} \delta(i, \varphi, G)$.

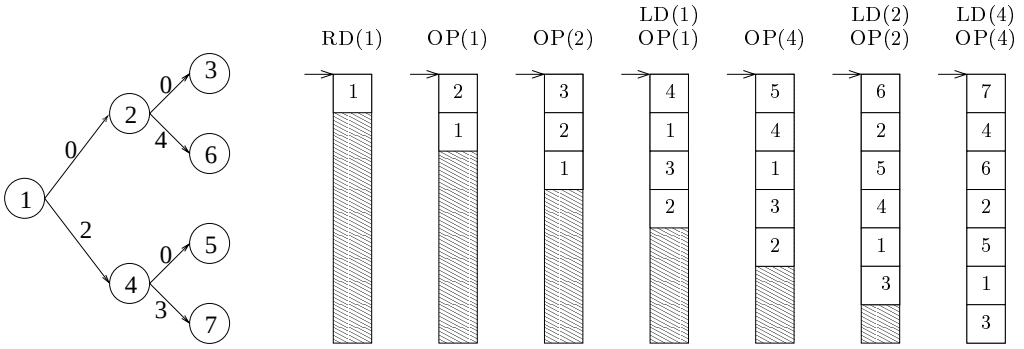
PROOF. Let us consider the indicator $\delta(u, v)$ that is equal to 1 if and only if there is an arc $(u, w) \in A$ such that $\varphi(u) \leq \varphi(v) < \varphi(w)$ and equal to 0 otherwise. By definition, we have $\delta(\varphi(v), \varphi, G) = \sum_u \delta(u, v)$ so that $\mathrm{DSC}(\varphi, G) = \sum_v \sum_u \delta(u, v) = \sum_u \sum_v \delta(u, v)$. The inner sum is equal to the number of vertices v that are numbered in the interval $\{\varphi(u), \cdots, \max_{(u,w) \in A} \varphi(w) - 1\}$, that is $\left(\max_{(u,v) \in A} \varphi(v)\right) - \varphi(u)$. So we have proved the equality.

This result is the counterpart of the equality between the profile and the reversed sum cut for undirected graphs [7, Observation 2.2 citing [12]]. In the rest of this paper, this first objective function will be referred to as DSC.

Notations introduced for $\mathcal{C}$ are directly adapted to the DSC cost function : $\mathrm{DSC}(\varphi, (u, s_i(u))) = \varphi(s_i(u)) - \varphi(s_{i-1}(u))$ and $\mathrm{DSC}(\varphi, u) = \max_{(u,v) \in A} \varphi(v) - \varphi(u)$.

2.2 Uniform Cost Stack

The UCS model (for *Uniform Cost Stack*) is intended to represent the load costs of variables which are stored during the execution of a program. This model is an extension of the well-known model of Sethi presented in [16] for the register allocation problems.



(a) Out tree with UCS costs

(b) Stack evolution

Fig. 2. UCS computation

The memory is seen as a stack, on which three operators are available:

- RD(α) reads the value of the input for the variable α and push it to the top of the stack. The duration of this operation is assumed to be a constant so that its cost in the model is 0.
- LD(α) moves the variable α stored in the stack to the top of the stack. The cost of this operation is proportional to the number of variables stored between α and the top before the move.
- OP($\alpha_1, \cdots, \alpha_k$) applies an operator — generically denoted by OP — to the values of the variables $\alpha_1, \cdots, \alpha_k$. It is supposed that $\alpha_1, \cdots, \alpha_k$ have been previously moved (with LD-operations) to the first k levels of the stack but these k variables can be in any order inside these first k levels of the stack. This assumption can be justified by the fact that, in a real processor, the parameters of the operators are stored in registers and the order in which the registers are initialized has no importance. The result of the computation of OP is then moved to the top of the stack, the order between the input values staying unchanged. Since the cost of an operation is supposed to be constant, we set it equal to zero.

For example, let us consider a graph $G = (V, A)$ pictured by Figure 2(a). The ordering function corresponds to the numbers printed inside the nodes. For this ordering function, we can derive the list of RD, LD and OP operations that are executed to evaluate the vertices of the graph. Figure 2(b) represents these operations for our example with the successive states of the stack — LD(i) means "load variable $\varphi^{-1}(i)$". The total cost of an execution is then the sum of the costs of the LD-moves : each of them is associated with an arc $(u, v) \in A$. In Figure 2(a), the arcs are valued with the corresponding cost. In this way, we get a total UCS cost equal to 9.

For general graphs, the code generation associated with graph order φ is more

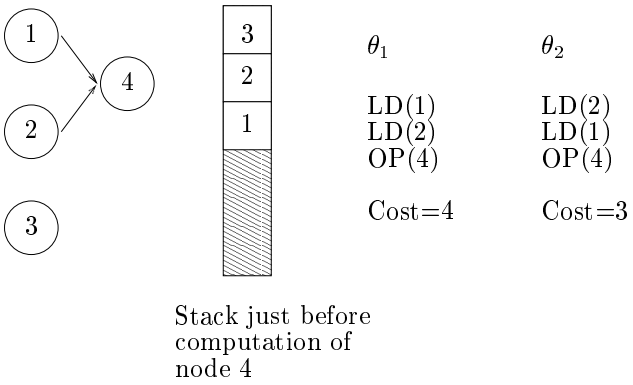


Fig. 3. Two different stacking orders to evaluate $\varphi^{-1}(4)$

complicated: indeed, if a vertex $u \in V$ has several predecessors, we have to decide in which order they will be loaded in the stack before u to minimize the cost.

2.2.1 Optimal execution of an order

For a given execution order φ, the UCS model as it has been defined so far does not guarantee the unicity of the score, because an order between the LD-operations, called *stacking order*, has to be defined for nodes which have several predecessors. Indeed, let us consider the example pictured by Figure 3. The total cost of an execution depends on the loading order of the variables $\varphi^{-1}(1)$ and $\varphi^{-1}(2)$ for the evaluation of $\varphi^{-1}(4)$. If $\varphi^{-1}(1)$ is loaded before $\varphi^{-1}(2)$, the cost is 4, while if the order is reversed, the cost is equal to 3.

In the following, we present an optimal simple — that is algorithmic and polynomial — stacking policy. With this policy added to our model, the UCS cost becomes unambiguously defined.

Definition 2 (Stacking order) *The stacking order θ_u of a vertex $u \in V$ (which is not a source of G) is a permutation of the elements of $\Gamma^-(u)$. It represents the order in which the predecessors of u must be loaded before the computation begins. The* total staking order *is the set $\theta = \{(u, \theta_u), u \in V\}$. The cost of θ for the graph order φ is denoted by $\mathrm{UCS}_\theta(\varphi, G)$ or shortly $\mathrm{UCS}_\theta(\varphi)$.*

For the example pictured by Figure 3, $\varphi^{-1}(1)$, $\varphi^{-1}(2)$ and $\varphi^{-1}(3)$ have no stacking order. The stacking order of $\varphi^{-1}(4)$ is $(\varphi^{-1}(1), \varphi^{-1}(2))$.

Let us consider a vertex u that has to be evaluated. If a subset W of $\Gamma^{-1}(u)$ is stored in the first $|W|$ levels of the stack, these variables need not to be loaded. So, only the variables $x \in \Gamma^{-1}(u)$ such that there is some $y \notin \Gamma^{-1}(u)$ with y closer to the top of the stack than x have to be loaded. Such a variable x is called a TBL (to be loaded) variable.

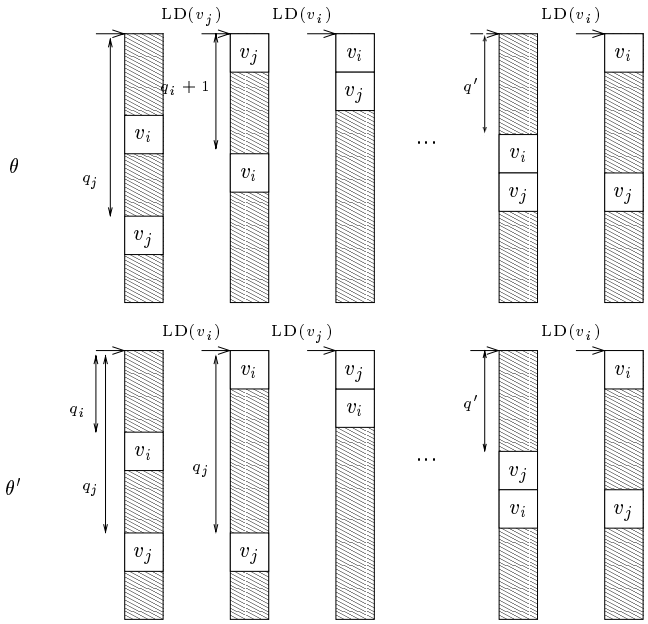


Fig. 4. Comparison between the stacking orders θ and θ'

Lemma 3 *The predecessors of any vertex $u \in V$ must be loaded according the following rule: load first the TBL-variable that is the closest to the top for the evaluation of u.*

PROOF. Let $p = |\Gamma^-(u)|$ and let $v_1, \cdots, v_p$ be the TBL-predecessors of u ordered from the top of the stack. Let $q_1, \cdots, q_p$ be their respective distance to the top of the stack just before the execution of u. Let us suppose that there exists an optimal stacking order θ different from the order of the lemma.

Let us now consider the minimal integer $k \in \{1, \cdots, p-1\}$ such as $\theta_u(k) = v_i$, $\theta_u(k+1) = v_j$ and $j < i$ (the first inversion). Let θ'_u be the stacking order defined by the inversion of $\theta_u(k)$ and $\theta_u(k+1)$ and let θ' be the total stacking order derived from θ after changing (u, θ_u) into (u, θ'_u).

We prove now that $\mathrm{UCS}_{\theta'}(\varphi) \leq \mathrm{UCS}_{\theta}(\varphi)$. Let us compare the successive stack states for the two executions corresponding to θ and θ', as it is illustrated by Figure 4. Before the program arrives at the pair of operations "LD(v_j);LD(v_i)" of θ (which corresponds to "LD(v_i);LD(v_j)" for θ') the θ-stack and the θ'-stack are identical at each step. Afterwards, v_j and v_i are swapped in the stack until an operator LD(v_i) or LD(v_j) is met. So, clearly, the cost difference between $\mathrm{UCS}_{\theta'}(\varphi)$ and $\mathrm{UCS}_{\theta}(\varphi)$ are due to the three LD operations we have just emphasized. Let q' be the height of v_i in the θ-stack before the third LD,

q' is also the height of v_j in the θ'-stack at the same time. So, we have

$$\mathrm{UCS}_\theta(\varphi) - \mathrm{UCS}_{\theta'}(\varphi) = (q_j + q_i + 1) - (q_i + q_j) + \begin{cases} q' - (q' + 1) \\ (q' + 1) - q' \end{cases}$$

according that the third operation is $\mathrm{LD}(v_i)$ or $\mathrm{LD}(v_j)$. Clearly, in any case, $\mathrm{UCS}_{\theta'}(\varphi) \leq \mathrm{UCS}_\theta(\varphi)$.

For the following, we will suppose that the TBL-variables are always loaded in the optimal order $\theta^\star$, so the cost of a graph order can be denoted without ambiguity by $\mathrm{UCS}(\varphi) = \mathrm{UCS}(\varphi, G) = \mathrm{UCS}_{\theta^\star}(\varphi, G)$. $\mathrm{UCS}(\varphi, (u, v))$ will denote the load of u in order to compute v.

2.3 Remarks

The DSC cost function can be trivially computed in $\mathcal{O}(m)$ time. The UCS cost function can be naively computed in $\mathcal{O}(mn)$ by using a linked list to represent the stack. However, by using AVL trees [1] instead, the computation time can be improved to $\mathcal{O}(m \log n)$ [15].

For both the DSC and UCS cost functions, the problem can be naturally decomposed when the precedence graph has several connected components. Formally, for $\mathcal{C} \equiv \mathrm{DSC}$ or $\mathcal{C} \equiv \mathrm{UCS}$, if the graph G has k connected components $G_1, \cdots, G_k$, then $\min_\varphi \mathcal{C}(\varphi, G) = \sum_{i=1}^{k} \min_{\varphi_i} \mathcal{C}(\varphi_i, G_i)$.

3 The DSC model

This section is dedicated to the DSC model. Firstly, we prove that the problem is NP-complete, even for graphs with depth equal to 2. Then, we prove that the problem is polynomial for in trees and out trees.

3.1 Complexity

We did not find in the literature any proof of the complexity of the directed sum cut problem. We prove in this section that the problem is unsurprisingly NP-complete. The problem is NP-complete even for the digraphs of depth 2 — the depth being the length in number of arcs of the longest path from a

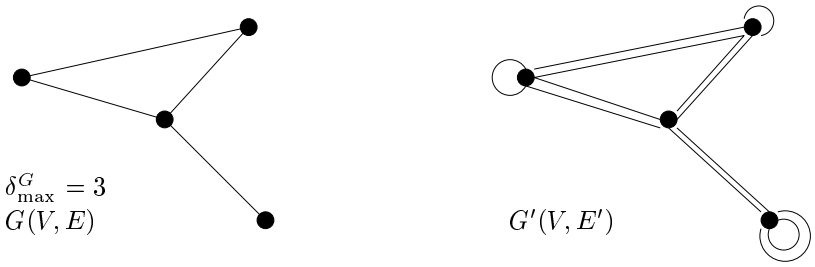


Fig. 5. Transforming MinLA to MinMaxEdge

source to a sink. We consider the following decisional variant of the problem of the minimization of the directed sum cut.

Minimum Directed Sum Cut (MinDSC)
INSTANCE : An acyclic digraph $G = (V, A)$ and an integer K.
QUESTION : Is there an order φ of G such that $\mathrm{DSC}(\varphi, G) \leq K$?

In order to prove that MinDSC is NP-complete, we will consider the following intermediate problem : **Minimum Max Edge (MinMaxEdge)**
INSTANCE : A multi-graph $G = (V, E)$ and an integer K
QUESTION : Is there an order φ of G such that $\sum_{\{u,v\}\in E} \max(\varphi(u), \varphi(v)) \leq K$?

We will start from Minimum Linear Arrangement [10]: **Minimum Linear Arrangement (MinLA)**
INSTANCE : $G = (V, E)$ a graph, an integer B.
QUESTION : Is it possible to find a bijective function $f : V \to \{1, ..., |V|\}$ such that

$$\sum_{\{i,j\}\in E} |f(i) - f(j)| \leq B?$$

Lemma 4 *There exists a polynomial transformation from MinLA to MinMaxEdge.*

PROOF. Let us consider an arbitrary graph $G = (V, E)$ given as an input of MinLA. Let $\delta^G(u)$ denote the degree of the vertex $u \in V$ in G and let $\delta^G_{\max} = \max_{u\in V} \delta^G(u)$ be the maximum degree. We build the (multi)graph $G' = (V, E')$ by doubling each arc in E and adding $\delta^G_{\max} - \delta^G(u)$ loop-edges $\{u, u\}$. Therefore, the degree of u in G' is $2\delta^G(u) + 2(\delta^G_{\max} - \delta^G(u)) = 2\delta^G_{\max}$ (see Figure 5). Since $\max(i, j) = (i + j)/2 + |i - j|/2$,

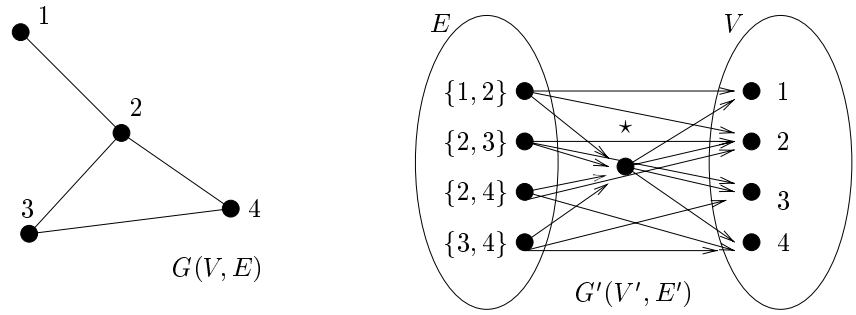


Fig. 6. Transforming MinMaxEdge to MinDSC

$$\begin{aligned}\sum_{\{u,v\}\in E'} \max(\varphi(u),\varphi(v)) &= \delta^G_{\max} \sum_{u\in V} \varphi(u) + \sum_{\{u,v\}\in E'} |\varphi(u)-\varphi(v)|/2 \\ &= \delta^G_{\max}|V|(|V|+1)/2 + \sum_{\{u,v\}\in E, u\neq v} |\varphi(u)-\varphi(v)| \\ &= \delta^G_{\max}|V|(|V|+1)/2 + \sum_{\{u,v\}\in E} |\varphi(u)-\varphi(v)|\end{aligned}$$

In the right side of the final equality, the first member of the sum is a constant while the second one is the linear arrangement of G. So a solution φ for MINIMUM MAX EDGE with a cost $\delta^G_{\max}|V|(|V|+1)/2 + K$ is a solution for MINLA with a cost K and vice versa.

Lemma 5 *There exists a polynomial transformation from MinMaxEdge to MinDSC restricted to the digraphs of depth* 2.

PROOF. Let $G = (V, E)$ be an arbitrary multi-graph (input of MinMaxEdge). We build the graph $G'(V', E')$ with $V' = E \cup \{\star\} \cup V$ (note that E is a multiset and may contain duplicate values corresponding to parallel arcs, V' is however a set: the multiple occurrences of an element of E are differentiated from each other in V'). E' is the union of the three following sets (see Figure 6) :

- $\{(e, \star) | \forall e \in E\}$
- $\{(\star, u) | \forall u \in V\}$
- $\{(\{u, u\}, u) | \forall \{u, u\} \in E\} \cup \{(\{u, v\}, u), (\{u, v\}, v) | \forall \{u, v\} \in E | u \neq v\}$

Clearly, for any order φ' of G', $\varphi'(E) = \{1, \cdots, |E|\}$, $\varphi'(\star) = |E| + 1$ and $\varphi'(V) = \{|E| + 2, \cdots, |E| + |V| + 1\}$. Since, in G', there is no outgoing arc from the nodes in V, the cache function for any order φ' is :

$$\mathrm{DSC}(\varphi', G') = \sum_{\{u,v\}\in E} (\max(\varphi'(\star), \varphi'(u), \varphi'(v)) - \varphi'(\{u, v\})) + \max_{u\in V} \varphi'(u) - \varphi'(\star)$$

Since $\varphi'(\star) < \varphi'(u)$ for any $u \in V$, $\sum_{\{u,v\}\in E} \varphi'(\{u, v\})$ is the sum of the integers $1, \cdots, |E|$, and $\max_{u\in V} \varphi'(u)$ is the last number of the order, that is

$|E| + |V| + 1$, we finally have:

$$\mathrm{DSC}(\varphi', G') = \sum_{\{u,v\} \in E} \max(\varphi'(u), \varphi'(v)) - |E|(|E|+1)/2 + |V|$$

So, when we have a directed order φ' for G', we can build an undirected order for G by taking for any $u \in V$ $\varphi(u) = \varphi'(u) - \varphi'(\star)$. The above equality becomes $\mathrm{DSC}(\varphi', G') = \sum_{\{u,v\} \in E} \max(\varphi(u), \varphi(v)) + |V|$. Therefore if the directed sum cut of φ' is less than $|V| + K$, the cost of the constructed order φ is less than K. Conversely, if we have an order φ for G with cost less than K, we can easily build an order with a directed sum cut less than $|V| + K$ by taking, for any $u \in V$, $\varphi'(u) = \varphi(u) + |E| + 1$, $\varphi'(\star) = |E| + 1$ and by randomly ordering the elements of E.

Now, since MinLA [10] is NP-complete, we deduce the following theorem :

Theorem 6 *MinDSC is NP-complete for digraphs of depth* 2

The approximation techniques of Rao and Richa [14], based on the Divide-and-Conquer approximation method presented by Even et al. [8], can be directly adapted to give an $\mathcal{O}(\log n)$-approximation algorithm for MINDSC.

3.2 Polynomial cases

3.2.1 In tree

Here the precedence graph $G = (V, A)$ is an in tree, *i.e.* each node $v \in V$ has at most one outgoing arc. With this property, the expression of the directed sum cut is greatly simplified

$$\mathrm{DSC}(\varphi, G) = \sum_{(u,v) \in A} \varphi(v) - \varphi(u).$$

This expression shows that when the precedence graph is an in tree, the directed sum cut is equal to the directed linear arrangement. The latter problem has been shown to be polynomial [2].

It can also be observed that the problem is equivalent to the scheduling problem $1|\mathit{intree}, p_i = 1| \sum w_i C_i$, in which each task i corresponds to a node in $v \in V$, the precedence graph is equal to G, and the task weights are $w_i = \delta^-(v)$. This problem is of course polynomial (see for example [3]).

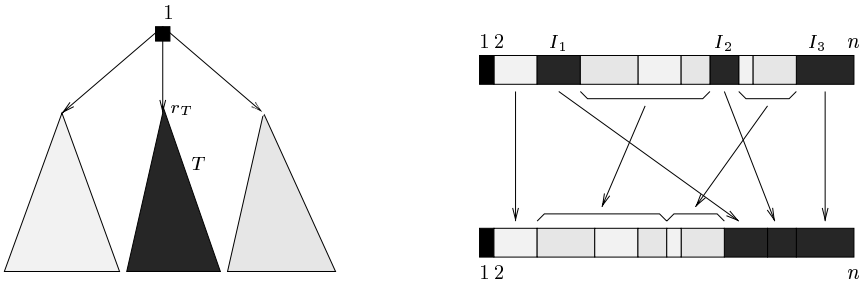


Fig. 7. Renumbering the order of an out tree

3.2.2 Out tree

The precedence graph $G = (V, A)$ is now an out tree, *i.e.*, each node $v \in V$ has at most one ingoing arc. From Section 2.3, we assume w.l.o.g. that G is connected so that $m = n - 1$. We present an algorithm that computes the optimal ordering of the nodes in linear time. This algorithm is based on the following lemma.

Lemma 7 *There exists an optimal order in which all the nodes in some subtree T of the root node of G are ordered after all the nodes that are in $G\backslash T$.*

$$\forall u \in T, \forall v \in G\backslash T, \varphi(u) > \varphi(v)$$

PROOF. Let us consider an order φ that does not verify this property. By the way of the transformation depicted in Figure 7, we are going to construct a new order φ' such that $\mathrm{DSC}(\varphi, G) \leq \mathrm{DSC}(\varphi, G)$. The root node r clearly satisfies $\varphi(r) = 1$, let T denote the subtree of r that contains the "last-ordered" node $\varphi^{-1}(n)$ and let r_T be the root of T. From our initial assumption, we have that $\varphi(r_T) < n - |T| + 1$, which means that at least one node of $G\backslash T$ is ordered in between the nodes of T. The new order φ' (see Figure 7) is build such that

(1) the relative order between the nodes in T is not modified,
(2) the relative order between the nodes in $G\backslash T$ is not modified,
(3) $\varphi'(T) = [n - |T| + 1, n]$ and, consequently, $\varphi'(G\backslash T) = [1, n - |T|]$.

These three rules clearly define the construction of a unique order φ'. This order is compatible with the topological order because (r, r_T) is the only arc between T and the rest of G and, after the transformation, we still have $\varphi(r) = 1 < \varphi(r_T)$.

In order to show that $\mathrm{DSC}(\varphi', G) \leq \mathrm{DSC}(\varphi, G)$, we consider an arc (u, v) such that $u \neq r$ — the case $u = r$ is studied afterwards. Let $\Delta(u, v) = (\varphi(v) - \varphi(u)) - (\varphi'(v) - \varphi'(u))$. $\Delta(u, v)$ is the decrease of the cost of (u, v) by re-ordering φ in φ'. By construction, $\Delta(u, v)$ is positive. Indeed, if (u, v) is in the subtree T, $\Delta(u, v)$ is equal to the number of nodes $w \notin T$ such that

$\varphi(u) < \varphi(w) < \varphi(v)$. Symmetrically, if (u, v) is not in the subtree T, $\Delta(u, v)$ is equal to the number of nodes $w \in T$ such that $\varphi(u) < \varphi(w) < \varphi(v)$. Therefore, the access cost of each node $u \neq r$ has decreased, that is, with the formulation of the cache cost given in (1), $\mathcal{C}(\varphi', u) \leq \mathcal{C}(\varphi, u)$.

However, the access cost $\mathcal{C}(\varphi, r)$ of the root node r generally increases. The increase is equal to $\mathcal{C}(\varphi', r) - \mathcal{C}(\varphi, r) = \max_{(r,v)\in A} \varphi'(v) - \max_{(r,v)\in A} \varphi(v) \leq \varphi'(r_T) - \varphi(r_T)$. Let $\Delta(r_T)$ denote the value of the right side of this inequality. We observe that $\Delta(r_T)$ is equal to the number of nodes $w \notin T$ such that $\varphi(w) > \varphi(r_T)$.

We complete the proof that $\mathrm{DSC}(\varphi', G) \leq \mathrm{DSC}(\varphi, G)$ by showing that the decrease of the total access cost of all the nodes $u \neq r$ is at least $\Delta(r_T)$. $\varphi^{-1}(T)$ is a subset of $\{1, \cdots, n\}$ so that it can be seen as the union of integer intervals $I_1 < I_2 \cdots < I_k$. Clearly, φ is a bijection between the nodes of T and $\bigcup_{i=1}^{k} I_i$. Let us consider the arc (u_1, v_1) of T such that $\varphi(u_1) \in I_1$ and $\varphi(v_1)$ is maximum. Since T is connected, $\varphi(v_1)$ is in some interval I_{k_1} with $k_1 > 1$. v_1 is by construction the last successor of u_1 according the order φ so that the access cost of u_1 is $\varphi(v_1) - \varphi(u_1)$. If $k_1 < k$, we can iterate the construction: let (u_2, v_2) be the arc such that $u_2 \in I_{k_1}$ and $\varphi(v_2)$ is maximum. Let I_{k_2} be the interval that contains $\varphi(v_2)$. At the end, we construct a sequence of arcs $(u_1, v_1), \cdots, (u_l, v_l)$ and intervals $I_1 = I_{k_0}, I_{k_2}, \cdots, I_{k_l} = I_k$ such that $\varphi(u_i) \in I_{k_{i-1}}$ and $\varphi(v_i) \in I_{k_i}$. We also have $\mathcal{C}(\varphi, u_i) = \varphi(v_i) - \varphi(u_i)$ and $\mathcal{C}(\varphi', u_i) = \varphi'(v_i) - \varphi'(u_i)$. So $\mathcal{C}(\varphi', u_i) - \mathcal{C}(\varphi, u_i)$ is equal to the number of nodes $w \notin T$ such that $\varphi(u_i) < \varphi(w) < \varphi(v_i)$. Therefore, since v_i and u_{i+1} are in the same interval I_{k_i}, the sum $\sum_{i=1}^{l} (\mathcal{C}(\varphi', u_i) - \mathcal{C}(\varphi, u_i))$ is equal to $\Delta(r_T)$. Therefore $\mathrm{DSC}(\varphi', G) \leq \mathrm{DSC}(\varphi, G)$, which completes the proof.

Let us call the subtree T that is ordered after all the other nodes in $G \backslash T$ the *terminal subtree* of the order. In order to derive new properties for the optimal ordering, let us assume that one of the subtree of r, again denoted by T, has been selected to be the terminal subtree of an order φ. We are going to show that this assumption decompose the problem into $\delta^+(r)$ independent problems. Since T is the final subtree, we have that the order of the root node r_T of T is $\varphi(r_T) = n - |T| + 1$. From the order φ of G, an order of T can be clearly derived by setting for any node u of T, $\varphi'(u) \leftarrow \varphi(u) - (n - |T|)$. So, $\mathcal{C}(\varphi, u) = \mathcal{C}(\varphi', u)$ for any node u of T, and therefore an optimal order of G correspond to an optimal order of T. Similarly, we show that an optimal order of G also correspond to an optimal order of $G \backslash T$. Since $G \backslash T$ is a forest, the optimal order is obtained by computing the optimal order of each tree of the forest and concatenating these orders.

Let us now determine how to select the terminal subtree. For each direct descendant u of r, let $T(u)$ denotes the subtree rooted at u. If $T(u^\star)$ denoted

the final subtree, we have $\mathcal{C}(\varphi, r) = n - |T(u^\star)|$, so

$$\mathrm{DSC}(\varphi, G) = n - |T(u^\star)| + \sum_{(r,u)\in A} \mathrm{DSC}(\varphi, T(u))$$

Therefore, in order to minimize the directed sum cut, $u^\star$ must be selected such that $T(u^\star)$ is the largest subtree of r.

So, the decomposition shows that the ordering of an in tree G is given by calling the following recursive algorithm with the root of G as first parameter and 1 as second parameter.

proc orderouttree(r,i) **:**
  $\varphi(r) \leftarrow i$
  if r is not a leaf **then**
    let $u^\star$ be one descendant of r such that $T(u^\star) = \max_{(r,u)\in A} |T(u)|$
    for each direct descendant $u \neq u^\star$ of r **do**
      orderouttree(u,$i+1$)
      $i \leftarrow i + |T(u)|$
    endfor
    orderouttree($u^\star$, $i+1$)
  endif

We finally prove the complexity of this algorithm.

Theorem 8 *The minimal directed sum cut of an out tree can be computed in $\mathcal{O}(n)$ time.*

PROOF. In a preprocessing phase, all the sizes $|T(u)|$ of the subtrees for all the nodes u of G can be computed in $\mathcal{O}(n)$ time. The recursive procedure is called once for each node and selecting $u^\star$ at a given node r takes $\mathcal{O}(\delta^+(r))$ time, so the total time for the algorithm is $\mathcal{O}(n)$.

Let us now consider the variant of the DSC cost function where $\mathcal{C}(\varphi) = \sum_u \sum_{i=1}^{\delta^+(u)} f(\varphi(s_i(u)) - \varphi(s_{i-1}(u)))$. If we assume that f is concave and non-decreasing, we can prove, by using the inequality $f(x+y) \leq f(x) + f(y)$ for any $x, y \geq 0$, that the lemma and the algorithm to solve the problem both hold. If the function f is convex, the lemma is not true anymore.

4.1 Complexity of UCS *for a bipartite graph*

We prove here that the decision version of UCS is NP-complete even for bipartite graphs. The problem is defined as follows:
Minimum Bipartite uniform cost stack (BipUCS)
INSTANCE : $G = (V, A)$ a bipartite directed acyclic graph, an integer K.
QUESTION : Is it possible to find a bijective function $\varphi : V \to \{1, ..., |V|\}$ such that $\text{UCS}(\varphi, G) \leq K$?

We prove that BipUCS is NP-complete using a reduction from Minimum Linear Arrangement (MinLA).

Theorem 9 *There exists a polynomial transformation from MinLA to BipUCS.*

PROOF. Let us consider an instance Π of MinLA given by a graph $H = (W, E)$ with $W = \{1, \cdots, n\}$ and an integer B. Let m be equal to $|E|$. We build an associated instance Π' of BipUCS defined by a graph $G = (V, A)$ and an integer K with $V = X \cup Y$ defined as follows (see Figure 8):

- The set Y corresponds to W. We denote by y_i the element of Y that is associated to the vertex i of W.
- The set X is the union of the set $X(E)$, which corresponds to E and the sets $Q(y_1), \cdots, Q(y_n)$. The element of $X(E)$ corresponding to the edge $e = \{i, j\}$ of E is denoted by $x(e)$ or $x(\{i, j\})$. Each set $Q(y_i)$ has n^6 elements and the sets $Q(y_i)$ are pairwise distinct.
- Each vertex in $Q(y_i)$ has one successor that is y_i. Each vertex $x(\{i, j\})$ has exactly two successors y_i and y_j.
- K is set to be equal to $(n^6 + m + 1)B + m^2$.

This transformation is clearly polynomial and the graph G is bipartite.

Let us suppose that the answer to Π is "yes" and let f a solution. In order to simplify the notation, we assume without loss of generality that $f(i) = i$ for each $i \in W$. An order φ of the corresponding instance Π' is computed by numbering vertices y_i of Y correspondingly to the order f of W. In other words, $\varphi(y_1) < \varphi(y_2) < ... < \varphi(y_n)$. Assuming that $\varphi(y_0) = 0$, we number the elements of X as follows :

- from $\varphi(y_{i-1}) + 1$ to $\varphi(y_{i-1}) + n^6$, the n^6 elements of $Q(y_i)$

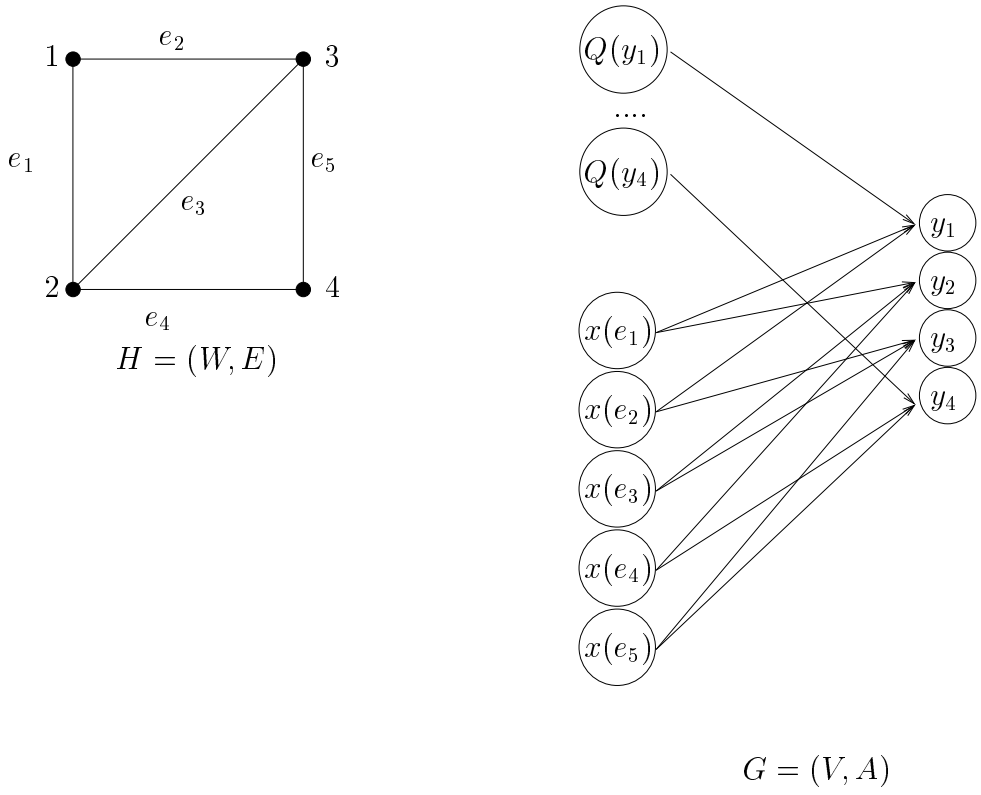


Fig. 8. Transforming MinLA to BipUCS

- from $\varphi(y_{i-1}) + n^6 + 1$ to $\varphi(y_i) - 1$, the elements $x(\{i, j\})$ of $X(E)$ such that $i < j$. Clearly, the vertices $x(\{i, j\})$ with $j < i$ have been numbered before $\varphi(y_j)$.

Now, we prove that $\mathrm{UCS}(\varphi, G) \leq K$. $\forall (x, y) \in A$, $\mathrm{UCS}(\varphi, (x, y))$ denotes the load cost of x for the evaluation of y. We obtain:

$$\mathrm{UCS}(\varphi, G) = \sum_{y \in Y} \sum_{x \in \Gamma^-(y)} \mathrm{UCS}(\varphi, (x, y))$$

In this sum, $\mathrm{UCS}(\varphi, (x, y)) > 0$ only for some $y = y_i$ and $x = x(\{i, j\})$ with $j < i$. Let us consider such a vertex x. x is a predecessor of y_i, it was first pushed in the stack with a RD-operation for the computation of y_j, that was executed before the computation of y_i ($j < i$ implies that $\varphi(y_j) < \varphi(y_i)$). We can set $\mathrm{UCS}(\varphi, (x, y_i)) = \Delta(x, y_j) + \Delta(y_j)$ where $\Delta(x, y_j)$ (*resp.* $\Delta(y_j)$) is the number of vertices stacked between x and y_j with y_j included (*resp.* between y_j and the top of the stack) just before the execution of y_i (see Figure 9).

Every vertex $y_j \in Y$ has at most m predecessors x in $X(E)$, so $\Delta(x, y_j) \leq m$. Moreover, every vertex $y_k \in Y$ has at most $n^6 + m$ predecessors. Since tasks from Y are stacked following φ, we get $\Delta(y_j) \leq (f(i) - f(j))(n^6 + m + 1)$.

$$\mathrm{UCS}(\varphi, G) \leq m^2 + (n^6 + m + 1) \sum_{\{i,j\} \in E} |f(i) - f(j)| \leq m^2 + (n^6 + m + 1) \times B$$

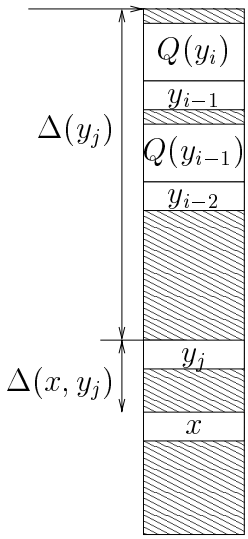


Fig. 9. Computation of y_i

So, we get $\mathrm{UCS}(\varphi, G) \leq (n^6 + m + 1)B + m^2 = K$. φ is then a solution for the instance Π' of BipUCS.

Conversely, let us suppose that φ is a solution to the instance Π' of BipUCS. We have $\mathrm{UCS}(\varphi, G) \leq K$. We can assume w.l.o.g. that tasks in W are numbered such that $\varphi(y_1) < \varphi(y_2) < \cdots < \varphi(y_n)$. Then, we build the order function $f(i) = i$ for any $i \in W$ and we prove that this function $f \equiv \mathrm{Id}$ is a solution for the instance Π of MinLA.

Let $e = \{i, j\} \in E$ be an edge with $i < j$. We estimate a lower bound for the cost of loading $x(e)$ in order to compute y_j. Let us consider the state of the stack before the LD operation. Since none of the y_k with $k < j$ have been loaded after their creation, the elements of Y appear in the $y_1, y_2, \cdots, y_{j-1}$ order in the stack. Moreover, the elements of $Q(y_k)$ for $1 < k < j$ have not been reloaded so that they are stacked between y_{k-1} and y_k. So, the height of y_i in the stack is at least $(j - i - 1)(n^6 + 1) + N_j$ where N_j is the number of elements of $Q(y_j)$ that are φ-ordered just before y_j. Since $x(e)$ has not been reloaded after the computation of y_i, it is deeper in the stack so that $\mathrm{UCS}(\varphi, (x(e), y_j)) \geq (f(j) - f(i) - 1)(n^6 + 1) + N_i$ (we use $f \equiv \mathrm{Id}$).

We now estimate the cost of loading all the variables of $X(E) \cup Q(y_j)$ required to compute y_j. First, the load cost of $x(\{i, j\})$ with $i > j$ are disregarded, that is we simply estimate that $\mathrm{UCS}(\varphi, (x(\{i, j\}), y_j)) \geq 0$. Let $x(\{i_1, j\}), x(\{i_2, j\}), \cdots$ be the variables with $i_k < j$ that must be loaded, we assume they are ordered according to the stacking order. Let L be the highest index, such that there is some elements of $Q(y_j)$ deeper than $x(\{i_L, j\})$ in the

stack. So we have that

$$\mathrm{UCS}(\varphi,(x(\{i_k,j\}),y_j)) \geq \begin{cases} (f(j)-f(i_k)-1)n^6 & \text{if } k \leq L \\ (f(j)-f(i_k))n^6 & \text{if } k > L \end{cases}$$

We finally show that the cost of loading the elements of $Q(y_j)$ is at least Ln^6. If $L=0$, it is obvious so that we study the case where $L \geq 1$. We consider the element x of $Q(y_j)$ which is the deeper in the stack. As, for each $k \leq L$ $x(\{i_k,j\})$ was last used to compute y_{i_k}, then the L elements $y_{i_1},\cdots,y_{i_L}$ of Y are between x and the top of the stack, so that the height of x is, counting $x(\{i_k,j\})$, at least $(L-1)n^6+L+N'$ where $N' < n^6$ is the number of elements of $Q(y_j)$ between $x(\{i_L,j\})$ and the top of the stack. If $N' = n^6-1$, then the cost of loading this element is at least $Ln^6+L-1 \geq Ln^6$. If $N' < n^6-1$, there is at most 2 elements of $Q(y_j)$ deeper than $x(\{i_L,j\})$. The cost of loading these two elements if at least $2((L-1)n^6+L+N') \geq Ln^6$. So, by summing these costs, we have shown that the cost of loading all the variables required to compute y_j is at least $(\sum_k f(j)-f(i_k))n^6$ so that $\mathrm{UCS}(\varphi,G) \geq \sum_{\{i,j\}\in E} |f(j)-f(i)| \times n^6$.

Now, since $\mathrm{UCS}(\varphi,G) \leq K$ and $K=(n^6+m+1)\times B+m^2$, we obtain that

$$(\sum_{\{i,j\}\in E} |f(j)-f(i)| - B) \times n^6 \leq (m+1)B+m^2$$

Furthermore, $m \leq n^2$ and $B \leq nm \leq n^3$, so

$$\sum_{\{i,j\}\in E} |f(j)-f(i)| - B \leq \frac{1}{n}+\frac{1}{n^2}+\frac{1}{n^3} < 1, \forall n > 1$$

and then $\sum_{\{i,j\}\in E} |f(j)-f(i)| \leq B$. f is then a solution to Π.

Corollary 10 *UCS is NP-hard for a bipartite directed acyclic graph.*

4.2 Polynomial cases

4.2.1 In tree

We suppose here that $G=(V,A)$ is an in tree. $\forall u \in V$, $\Gamma^{*-}(u)$ is the set of the ancestors of u and $s(u)$ is the unique successor of u in G. We also denote by $G(u)$ the subtree of G rooted by u and by r the root of G.

Lemma 11 *For any execution order φ, another φ' is built with $UCS(\varphi',G) \leq UCS(\varphi,G)$ and such that all the ancestors of any vertex $u \in V$ are ordered by φ' just before u:*

$$\forall v \in \Gamma^{*-}(u), \varphi'(v) \in \{\varphi'(u)-|\Gamma^{*-}(u)|,\cdots,\varphi'(u)-1\}$$

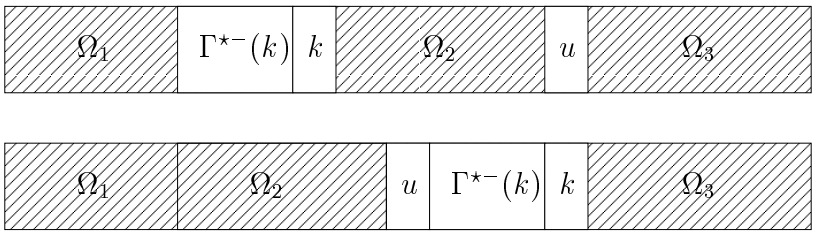


Fig. 10. Transforming an order for the UCS score of an in tree

PROOF. Let us suppose that φ is an optimal order which does not fulfill the condition expressed by the lemma. Let u be the first (according to the order φ) vertex of V which does not verify this condition. Let k be the last node (according to φ) such that $\varphi(k) < \varphi(u)$ and $k \notin \Gamma^{*-}(u)$.

By minimality of $\varphi(u)$, predecessors of k are computed just before k. A new order φ' will be derived from φ by moving k and $\Gamma^{*-}(k)$ just after u (figure 10). For the sake of clarity, the following sets are defined:

$$\Omega_1 = \{v \in V, \varphi(v) < \varphi(k) - |\Gamma^{*-}(k)|\}$$
$$\Omega_2 = \{v \in V, \varphi(k) < \varphi(v) < \varphi(u)\}$$
$$\Omega_3 = \{v \in V, \varphi(u) < \varphi(v)\}$$

Notice that $s(u)$ and $s(k)$ are both belonging to Ω_3. $\forall v \in V$, we set $\Delta(v) = \mathrm{UCS}(\varphi', (v, s(v))) - \mathrm{UCS}(\varphi, (v, s(v)))$. We prove that $\sum_{v \in V - \{r\}} \Delta(v) \leq 0$. The value $\Delta(v)$ depends on the 6 following cases:

(1) If $v = u$, then in the worst case, values from $\{k\} \cup \Gamma^{*-}(k)$ are still all stored between u and the top of the stack at the execution of $s(u)$ following φ'. So, we get $\Delta(u) \leq 1 + |\Gamma^{*-}(k)|$.
(2) If $v = k$, then k is closer to its father for φ', so $\Delta(k) \leq 0$.
(3) If $v \in \Gamma^{*-}(k)$, then $\Delta(v) = 0$ because such vertices are moved along with k.
(4) If $v \in \Omega_2$, then $\Delta(v) = 0$ because $s(v) \in \Omega_2 \cup \{u\}$.
(5) If $v \in \Omega_3$, then $\Delta(v) = 0$ since $\forall l \in \Omega_3$, $\varphi'(l) = \varphi(l)$.
(6) If $v \in \Omega_1$, then if $s(v) \in \Omega_1$, $\Delta(v) = 0$. Now, by hypothesis, $\Omega_1 \cap \Gamma^{*-}(u) \neq \emptyset$. So, there exists $l \in \Omega_1$ with $s(l) \in \Omega_2 \cup \{u\}$. For this value, $\Delta(l) = -1 - |\Gamma^{*-}(k)|$. We deduce that $\sum_{v \in \Omega_1} \Delta(v) \leq -1 - |\Gamma^{*-}(k)|$.

Thus,

$$\sum_{v \in V} \Delta(v) = \Delta(u) + \Delta(k) + \sum_{v \in \Omega_1} \Delta(v) \leq 0$$

Now, let us consider an order φ following the condition of the previous lemma and a vertex $u \in V$ with predecessors $p_1, ..., p_q$ numbered such that $\varphi(p_1) <$

$... < \varphi(p_q)$. If we denote by $|G(u)|$ the number of vertices of $G(u)$, the cost of $G(u)$ is :

$$\mathcal{C}(\varphi, G(u)) = \sum_{j=1}^{q} \mathcal{C}(\varphi, G(p_q)) + \sum_{j=1}^{q} (j-1)|G(p_q)|$$

This value is minimum iff $|G(p_1)| \geq |G(r_2)| \geq ...|G(p_q)|$. So, we deduce the following theorem :

Theorem 12 *Any order φ following the condition of the previous lemma and such that $\forall u \in V, \forall (p_1, p_2) \in \Gamma^-(u)^2, \varphi(p_1) < \varphi(p_2) \Rightarrow |G(p_1)| \geq |G(p_2)|$ is optimal.*

At each step $u \in V$, we have to sort the values $|G(v)|, v \in \Gamma^-(u)$. The complexity of the algorithm is then $\mathcal{O}(n \log n)$.

4.2.2 Out tree

We suppose here that $G = (V, A)$ is a connected out tree ($|A| = m = n - 1$). We prove here that the optimal order computed for DSC in Section 3.2.2, denoted by φ^*, is also optimal for UCS. We first prove an inequality linking UCS and DSC cost functions

Lemma 13 *In an out tree,* UCS$(\varphi) \geq DSC(\varphi) - m$ *for any graph order φ.*

PROOF. Let us consider the unique LD operation associated to the arc (u, v) of the out tree. We prove that UCS$(\varphi, (u, v)) \geq$ DSC$(\varphi, (u, v)) - 1$. If u is already on the top of the stack, it means that it is the result of the previous RD or OP operation. So we have $\varphi(v) = \varphi(u) + 1$ and $v = s_1(\varphi, u)$. Then UCS$(\varphi, (u, v)) = 0 =$ DSC$(\varphi, (u, v)) - 1$. If u is not on the top of the stack, $v = s_i(u)$ for some $1 \leq i \leq \delta^+(u)$. The last time that u was on the top of the stack was when u was created (if $i = 1$) or just before $s_{i-1}(u)$ was computed (if $i > 1$). In any case, at least $\varphi(s_i(u)) - \varphi(s_{i-1}(u)) - 1$ variables were pushed by OP operations afterwards. So, UCS$(\varphi, (u, v)) \geq \varphi(s_i(u)) - \varphi(s_{i-1}(u)) - 1 =$ DSC$(\varphi, (u, v)) - 1$. By summing these inequalities, we eventually have that UCS$(\varphi) \geq DSC(\varphi) - m$.

Theorem 14 *For an out tree G, the optimal solution $\varphi^\star$ of* DSC *is also optimal for* UCS. *So, the minimum UCS can be computed in $O(n)$ time.*

PROOF. We prove in the following that UCS$(\varphi^\star, G) =$ DSC$(\varphi^\star, G) - m$. Since $\varphi^\star$ is optimal for DSC and DSC$(\varphi^\star, G) - m$ is a lower bound for UCS (from Lemma 13), we conclude that $\varphi^\star$ is optimal for UCS.

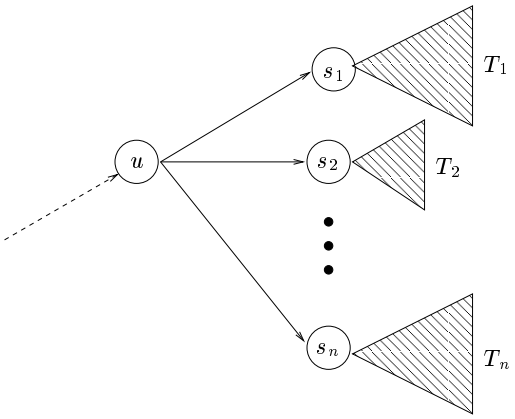


Fig. 11. A node u with its successors in a tree

Let us consider, as illustrated by Figure 11, a vertex u along with its $q = \delta^+(u)$ successors, denoted by $s_1 = s_1(\varphi^\star, u), \cdots, s_q = s_q(\varphi^\star, u)$. So, $\varphi^\star(s_1) < \cdots < \varphi^\star(s_q)$. We finally define $T_i, \forall i \in \{1, \cdots, n\}$, the vertices of the subtree whose root is s_i. At each step $i \in \{1 \cdots q\}$, u is loaded for the computation of s_i. V_i then denotes the set of vertices stacked between u and the top of the stack just before the evaluation of s_i. The UCS score of this load of u can then be expressed as $\mathrm{UCS}(\varphi^\star, (u, s_i)) = |V_i|$. There are 2 cases:

- If $i = 1$, by definition of $\varphi^\star$, $\varphi^\star(s_1) = \varphi^\star(u) + 1$. u is already on top of the stack, so $V_1 = \emptyset$. Therefore, $|V_1| = 0 = \varphi^\star(s_1) - \varphi^\star(u) - 1$.
- Else, if $i > 1$, we prove in the following that $V_i = T_{i-1}$:
  - By definition of $\varphi^\star$, elements of T_{i-1} are computed after the computation of s_{i-1} and before the load of u for the computation of s_i, so $T_{i-1} \subset V_i$.
  - Conversely, let $k \in V_i \backslash T_{i-1}$. $\varphi^\star(k) < \varphi^\star(s_{i-1})$ because the elements of T_{i-1} are numbered from $\varphi^\star(s_{i-1})$ to $\varphi^\star(s_i) - 1$. Therefore, there exists $l \in T_{i-1}$ whose computation requires the load of k because by construction all the elements numbered between s_{i-1} and s_i are in T_{i-1}. Since $k \notin T_{i-1}$, l has 2 predecessors, which contradicts the structure of G. In conclusion, $T_{i-1} = V_i$.

  Therefore:

$$\mathrm{UCS}(\varphi^\star, (u, s_i)) = |T_{i-1}| = \varphi^\star(s_i) - \varphi^\star(s_{i-1}) - 1 = \mathrm{DSC}(\varphi^\star, (u, s_i)) - 1$$

By summing these costs, we finally have that $\mathrm{UCS}(\varphi, G) = \mathrm{DSC}(\varphi, G) - m$, which completes the proof.

5 Conclusion

This paper has proposed two combinatorial optimization models for the problem of minimizing the memory access times for an integrated circuit simulators. The problems are NP-hard even when the depth of the graph is bounded. However, when the graph describing the circuit is an in tree or an out tree, the problems are polynomial for both our criteria.

It can be observed that for the two polynomial cases there is an order that simultaneously minimize both our criteria. Experimentally, some tests have shown that these criteria are correlated on graphs derived from existing integrated circuit. Preliminary tests have also shown that a real speed up can be obtained when the simulation code is based on a good ordering of the graph. However, in order to integrate our results in a real integrated circuit simulator, we are going to refine our theoretical models in order to deal with the hierarchical description of the circuits and to model the complete test procedure.

References

[1] G.M. Adelson-Velskii and Y.M. Landis. An algorithm for the organization of information. *Soviet Moth. Dokl.*, 3:1259–1262, 1962.

[2] D. Adolphson and T.C. Hu. Optimal linear ordering. *SIAM Journal on Applied Mathematics*, 25:403–423, 1973.

[3] P. Brucker. *Scheduling algorithms.* Springer, Berlin, 1998.

[4] J. Bruno and R. Sethi. Code generation for a one-register machine. *J. Assoc. Comp. Mach.*, 23(3):502–510, July 1976. ctr127.

[5] Paolo Detti and Dario Pacciarelli. A branch and bound algorithm for the minimum storage-time sequencing problem. *Naval Research Logistics*, 48(4):313–331, 2001.

[6] J. Díaz, A.M. Gibbons, M.S. Paterson, and J. Torán. The minsumcut problem. *Lecture Notes in Computer Science*, 519:65–79, 1991.

[7] Josep Díaz, Jordi Petit, and Maria Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34(3):313–356, 2002.

[8] G. Even, J. Naor, S. Rao, and B. Schieber. Divide-and-conquer approximation algorithms via spreading metrics. *Journal of the ACM*, 47:585–616, 2000.

[9] S. Even and Y. Shiloach. NP-completeness of several arrangement problems. Technical Report TR-43, Dept. of Computer Sciences, Technion, 1975.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[11] M.R. Garey, R.L. Graham, and D.S. Johnson. Complexity results for bandwidth minimization. *SIAM Journal of Applied Mathematics*, 34:477–495, 1978.

[12] P.A. Golovach and F.V. Fomin. The total vertex separation problem and the profile of graphs. *Diskretnaya Matematika*, 10:87–94, 1998.

[13] J. B. Gosling. *Simulation in the Design of Digital Electronic Systems*. Cambridge University Press, 1993.

[14] S. Rao and A. W. Richa. New approximation techniques for some ordering problems. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, pages 211–219, New-York, 1998. ACM.

[15] A. Robert. Modélisation, analyse et optimisation d'un simulateur de circuits. Master's thesis, University of Paris VI, 2003. in French.

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