Discrete Mathematics and Theoretical Computer Science **(subm.)**, by the authors, 2–rev

On the Relation Between Parallel Real-Time Computations and Logarithmic Space[†]

Stefan D. Bruda and Selim G. Akl

Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada K7L 3N6; Email: {bruda,akl}@cs.queensu.ca

We show that all the problems solvable by a nondeterministic machine with logarithmic work space (NLOGSPACE) can be solved in real time by a parallel machine, no matter how tight the real-time constraints are. We also show that, once real-time constraints are dropped, several other real-time problems are in effect solvable in nondeterministic logarithmic space. Therefore, we conjecture that NLOGSPACE contains exactly all the computations that admit efficient ($poly(n)$ processors) real-time parallel implementations. The issue of real-time optimization problems over independence systems is also investigated. We identify the class of such problems that are solvable in real time. Finally, we address the problem of obtaining approximate real-time solutions for problems not solvable in real time. In the process, we determine the computational power of directed reconfigurable multiple bus machines (DRMBMs) with polynomially bounded resources (processors and buses) and running in constant time, which is found to be exactly the same as the power of directed reconfigurable networks of polynomially bounded size and constant running time. In addition, we show that sophisticated and of questionable feasibility write conflict resolution rules (such as Priority or even Common) do not add computational power over the Collision rule, and are thus unnecessary, and that a bus of width 1 (i.e., a wire) suffices for any constant time computation on DRMBM.

Keywords: real-time computation, timed ω-languages, parallel complexity theory, reconfigurable multiple bus machines, approximation schemes, independence systems, matroids, bin packing

1 Introduction

The area of real-time computations has a strong practical grounding, in domains like operating systems, databases, and the control of physical processes. Besides these practical applications, however, research in this domain is primarily focused on formal methods and on communication issues in distributed real-time systems. Considerably less work has been done in the direction of algorithms and complexity theory.

One direction within this research area was, however, started by the introduction of *well-behaved timed* ω*-languages* [5]. Unlike previous models of real-time computation (such as, for example, the *real-time Turing machine* [23]), timed languages bridge the long standing gap between the complexity theorists and the real-time systems community. Indeed, the systems researchers use "real-time" to refer to those computations in which the notion of correctness is linked to the notion of time [22]. In theoretical circles,

[†]This research was supported by the Natural Sciences and Engineering Research Council of Canada.

on the other hand, this term is used as a synonym for *on-line* or *linear time*. While well-behaved timed ω-languages create a formal model, they also capture all the features of real-time computations as understood by the systems community. Such a claim is supported by the work from [5], where the formalism is used in order to model real-time computations encountered in highly practical areas. Real-time complexity classes, as well as complexity theoretic properties of real-time computations, are studied in [7, 8]. In particular, it is shown that real-time computations form an infinite hierarchy with respect to the number of processors, and such a hierarchy is independent of the underlying parallel abstract machine.

However, the real-time computations analyzed in [8] do not exhibit explicit deadlines. Instead, the real-time qualifier is given to those computations by the input (and its real-time characteristics). Still, most practical applications do require that computations are carried output within well-defined deadlines. For this reason, our main focus in this paper consists in computations with explicit deadlines. Based on the theory of timed ω-languages, we study (classical) languages that can be recognized in nondeterministic logarithmic space (NLOGSPACE), augmented with real-time constraints (including but not limited to deadlines). We show that all such computations can be carried out successfully in parallel, no matter how tight the time constraints are. Conversely, we show that, although hard to recognize in real time, the languages developed and analyzed in [8] can be accepted in deterministic logarithmic space once the time constraints are eliminated. Thus, we conjecture that logarithmic space contains in effect exactly all the computations that admit efficient ($poly(n)$ processors) real-time parallel implementations.

Supported by such a conjecture, we identify the class of optimization problems over independence systems that are solvable in real-time, and we are able thus to extend the results obtained in [2]. Indeed, we show that the solution obtained by a parallel algorithm is arbitrarily better than the solution reported by a sequential one not only for the real-time minimum-weight spanning tree (as shown in [2]), but for any real-time maximization problem over a matroid for which the size of the optimal solution can be computed in real time. As well, we identify as a promising research direction the process of identifying those problems that, even if not solvable in the real-time environment imposed by their use, admit good approximate, real-time computable solutions. In particular, we show that the *bin packing* problem does admit a good approximation parallel real-time algorithm, even if the exact variant is NP-complete.

Besides these main results, we also offer a tight characterization of constant time computations on *reconfigurable multiple bus machines* (RMBMs). We show that constant time directed RMBMs have the same computational power as the directed *reconfigurable networks*, and that there is no need for such powerful write conflict resolution rules as Priority or Common. Indeed, they do not add computational power over the easily implementable Collision rule. We also find an interesting gap result. Indeed, as far as constant time computations on RMBMs are concerned, we show that a unitary bus width is enough. That is, a simple wire as bus will do for all constant time computations on directed RMBM.

The results in this paper are presented as follows: In Section 3 on page 6 we show that exactly all nondeterministic logarithmic space languages can be recognized in constant time using a directed fusing reconfigurable multiple bus machine (F-DRMBM) with $poly(n)$ processors and $poly(n)$ buses, each of width 1. Based on this result, we establish the computational power of RMBMs. Our main results on real-time computations are the subject of Section 4 on page 11, where we establish that any NLOGSPACE language is computable in real time on RMBMs, no matter how tight the real-time restrictions actually are, and we state the aforementioned conjecture. The issue of optimization problems over independence systems is considered in Section 5 on page 14. We sketch a possible future research direction on approximation real-time algorithms in Section 6 on page 18, and we conclude in Section 7 on page 21.

2 Preliminaries

For some set Σ, $\mathcal{P}(\Sigma)$ stands for the power set of Σ, that is, $\mathcal{P}(\Sigma) = 2^{\Sigma}$. The cardinality of $\mathbb{N}$, the set of natural numbers is denoted by ω. $poly(n)$ expresses the upper bound for polynomial functions of one variable n, that is, $poly(n) = n^{O(1)}$. The empty word is denoted by λ.

Given some total function $f : \mathbb{N} \to \mathbb{N}$, we denote by SPACE($f(n)$) (NSPACE($(f(n))$)) the set of languages that are accepted by a deterministic (nondeterministic) Turing machine which uses at most $O(f(n))$ space (not counting the input tape) on any input of length n. LOGSPACE (NLOGSPACE) is a shorthand for SPACE($\log n$) (NSPACE($\log n$)). The class P (NP) contains exactly all the languages accepted in deterministic (nondeterministic) polynomial time. Finally, NC denotes the class of languages accepted in polylogarithmic time by some parallel machine using $poly(n)$ processors. Given some class C of languages (that is, boolean functions) and some (non-boolean) function f, we say by abuse of notation that $f \in C$ whenever the extension from language to function does not alter the complexity of computation.

2.1 Timed ω-languages

A sequence $\tau = \tau_1\tau_2\ldots \in \mathbb{N}^{\omega}$ is a *time sequence* if it is an infinite sequence of positive values, and $\tau_i \leq \tau_{i+1}$ for all $i > 0$. Any subsequence of a time sequence is a time sequence. A *well-behaved* time sequence is a time sequence for which, for every $t \in \mathbb{N}$, there exists some finite $i \geq 1$ such that $\tau_i > t$. A (well-behaved) *timed* ω-*word* over some alphabet Σ is a pair (σ,τ), where $\tau \in \mathbb{N}^k$ is a (well-behaved) time sequence, $k \in \mathbb{N} \cup \{\omega\}$, and $\sigma \in \Sigma^k$. Some τ_i from τ represents the time at which σ_i becomes available as input. For some timed ω-word $w = (\sigma,\tau)$, detime$(w) = \sigma$. By abuse of notation, detime$(L) = \{\text{detime}(w) | w \in L\}$.

The concatenation of two timed words is defined as the union of their sequences of symbols, ordered in nondecreasing order of their arrival time. Given two timed ω-languages L_1 and L_2, the concatenation of L_1 and L_2 is $L_1L_2 = \{w_1w_2 | w_1 \in L_1, w_2 \in L_2\}$. The notation $\prod_{i=1}^{n} w_i$ ($\prod_{i=1}^{n} L_i$) is a shorthand for $w_1w_2\cdots w_n$ ($L_1L_2\ldots L_n$).

A *real-time algorithm* A consists in a *finite control*, an *input tape* that always contains a (not necessarily well-formed) timed ω-word, and an *output tape* containing symbols from some alphabet Δ. The input tape has the same semantics as a timed ω-word. During any time unit, A may add at most one symbol to the output tape. The content of the output tape of A working on w is denoted by $O(A,w)$. There exists a designated symbol $f \in \Delta$. A real-time algorithm A *accepts* the timed ω-language L if, on any input w, $|O(A,w)|_f = \omega$ iff $w \in L$.

Let $w = (\sigma,\tau)$ be some timed ω-word. For $i_0 = 0$ and any $j > 0$, let $s_j = \sigma_{i_{j-1}+1}\sigma_{i_{j-1}+2}\ldots\sigma_{i_j}$, such that (*a*) $\tau_{i_{j-1}+1} = \tau_{i_{j-1}+2} = \cdots = \tau_{i_j}$, and (*b*) $\tau_{i_j+1} \neq \tau_{i_j}$. Then, the size $|w|$ of w is $|w| = \max_{j>0}|s_j|$. Given a total function $f : \mathbb{N} \to \mathbb{N}$, and some model of parallel computation M, the class rt-PROC$^M(f)$ includes exactly all the well-behaved timed ω-languages L for which there exists a real-time algorithm running on M that accepts L and uses no more than $f(n)$ processors on any input of size n. By convention, the class rt-PROC$^M(1)$ of sequential real-time algorithms is invariant with M.

Pursuit and evasion on a ring For $k > 0$, the languages L^k, modeling the k-dimensional version of the pursuit and evasion on a ring problem [6] are developed in [8]. We shall give here a very brief overview of these languages, directing the interested reader to [8]. For a given k, and for given positive constants r, p, and c, L^k is a well-behaved subset of the language $L_0\prod_{i>0}L_{ci}$, where detime$(L_0) \in \{a,b\}^r$. Each $w^i \in L_{ci}$ denotes the move made by the pursuee at time $t = ci$, under the form of a *modulo k direction* and a *sequence* of at most p a's and/or b's. The sequence is to be inserted into the j-th segment (of length

r/k and viewed as a conceptual circle) of the initial word expressed by $w^0 \in L_0$, according to the given direction (both j and the direction being given by what we called above "the modulo k direction."

A word $w \in L^k$ is accepted iff it can be decided at some finite time T that the word available at that moment in time has an equal number of a's and b's (intuitively speaking, the accepting real-time algorithm catches the pursuee—or the input—at time T). In order to eliminate the ambiguity generated by the somehow generic notations used in [8], we shall denote henceforth L^k by PURSUIT_k, for any $k > 0$.

2.2 Models with reconfigurable buses

Two main models with reconfigurable buses have been developed in the literature: the *reconfigurable network* (or RN for short) [4] and the *reconfigurable multiple bus machine* (or RMBM) [21]. While both models have similar characteristics, RMBM features a clear separation between buses and processors. Throughout this paper, RMBM is thus our model of choice. We shall, however, briefly define RN, since we also refer to this model.

The reconfigurable multiple bus machine An RMBM [20, 21] consists a set of p *processors* and b (electronic, nondirectional) *buses*. For each processor i and bus b there exists a *switch* controlled by processor i. By these switches, a processor have access to the buses by being able to read or write from/to any bus. As well, a processor may be able to *segment* a bus, obtaining thus two independent, shorter buses. Any processor is allowed to *fuse* any number of buses together by using a *fuse line* perpendicular to and intersecting all the buses. A fuse line can be electrically connected to any number of buses, simultaneously. Two buses that are connected to the same fuse line are said to be fused, and act as a unique, longer bus.

DRMBM, the *directed* variant of RMBM [21], is identical to the undirected model (in particular, the buses continue to be nondirectional), except for the definition of fuse lines. In a DRMBM, each processor features two fuse lines (*down* and *up*) perpendicular to and intersecting all buses. At the processor's control, each of these fuse lines can be electrically connected to any bus. Assume that, at some given moment, buses i_1, i_2, ..., i_k are all connected to the down (up) fuse line of some processor. Then, a signal placed on bus i_j is transmitted in one time unit to all the buses i_l such that $l \geq j$ ($l \leq j$). It is argued in [20] that the fuse lines must use active components anyway, such that a directional connection is as practically realizable as a nondirectional one.

For ease of presentation, one can consider RMBM as a special case of DRMBM, in which the up and down fuse lines are "synchronized," in the sense that the down fuse line of some processor p_i is connected to some bus j iff the up fuse line of p_i is connected to bus j. We shall adopt in the following this uniform characterization, and thus we assume that each processor in any RMBM variant has two (up and down) fuse lines, even if these fuse lines may in fact act as one bidirectional line. Furthermore, as we shall emphasize below, it is clear from this construction that, for any nondirectional RMBM there exists a DRMBM simulating it, that uses the same amount of resources (time, processors, buses, bus width).

If some RMBM (DRMBM) is not allowed to segment buses, then this restricted variant is denoted by F-RMBM (F-DRMBM).

As far as the process of reading and writing on the buses is concerned, one can distinguish between CREW (concurrent read, exclusive write) and CRCW (concurrent read, concurrent write) RMBMs. Theoretically, exclusive read, exclusive write (EREW) RMBMs are possible as well, but we shall not consider such, since we believe that the ability of all the processors to listen to a common bus is a trivial feature (that is, some extra effort in order to insure exclusive read appears to be necessary).

For CRCW (concurrent read, concurrent write) RMBMs, one should establish a conflict resolution rule

for the process of writing a value to some bus. The most realistic such a rule is Collision (indeed, such a technique is widely used nowadays in the MAC network layer protocols, like CSMA-CS from which the Ethernet protocol is derived [19]), where two values simultaneously written on a bus result in the placement of a special collision value on that bus. Other conflict resolution rules (used for either RMBM or other models of parallel computation) are Common (two processors are allowed to simultaneously write on the bus only if the values written by them are identical), Arbitrary (some arbitrary processor succeeds in writing on the bus and the write request of all the others are discarded), Priority (the write request of the highest priority processor is the only one to succeed), and Combining (a combination of the values written by all the processors is placed on the bus). The use of the latter three rules for a bus (i.e., a spatially distributed resource) is indeed questionable. We will, however, consider all these possible rules. On one hand, this is done for completeness reasons. On the other hand, these rules are in fact equivalent, at least for the computational settings we are interested in (directed RMBMs with constant running time), as we shall show in Corollary 3.8 on page 10. We restrict only the Combining mode, requiring that the combining operation be associative and computable in nondeterministic linear space. We believe that these are reasonable restrictions, as they clearly hold for any reasonable combining operation.

As for most models of computation, the word size of each processor in an (D)RMBM is limited to $O(\log n)$ [21]. Furthermore, we are interested in constant time computations. Thus, we can assume without loss of generality that a processor has only a constant number of internal registers (indeed, even if there are an infinite number of registers, a processor can access only a constant number of them given the time restrictions). It follows that the *internal configuration* or *internal state* c_i of some processor p_i (which contains the content of p_i's registers and the state of p_i's finite control) in an RMBM can be expressed by a word of size $O(\log n)$. For similar reasons ($O(\log n)$ word size and constant running time) and by information theoretic arguments, it follows that, at any given time, one can fully describe which buses are fused together or segmented by a given processor, using a word of size $O(\log n)$. These limitations can be formally captured by introducing the concept of *uniform family* of RMBMs, similar to the concept of RN family [4].

An RMBM (DRMBM, F-DRMBM, etc.) *family* $\mathcal{R} = (R_n)_{n\geq 1}$ is a set containing one RMBM (DRMBM, F-DRMBM, etc.) construction for each $n > 0$. A family $\mathcal{R}$ solves a problem P if, for any n, R_n solves all inputs for P of size n.

A description of some (D)RMBM family using $p(n)$ processors and $b(n)$ buses is a list of $p(n)$ tuples $(i, c_i, up_i, down_i, segment_i)$, $1 \leq i \leq p(n)$. Such a tuple describes the configuration of processor p_i. Specifically, c_i denotes the internal configuration of p_i, and up_i ($down_i$, $segment_i$) represents a set of rules that determine which buses are fused by the up fuse line (fused by the down fuse line, segmented), depending on c_i. In the case of F-RMBM or F-DRMBM, the set $segment_i$ is always empty (no buses are ever segmented).

We say that some RMBM family $\mathcal{R}$ is a *uniform RMBM family* (or that $\mathcal{R}$ is *uniformly generated* in SPACE$(\log(p(n) \times b(n)))$) if there exists a Turing machine M that, given n, produces the description of R_n using $O(\log(p(n) \times b(n)))$ cells on its working tape. Since we deal only with uniform families here, we henceforth drop the "uniform" qualifier, with the understanding that any RMBM family described in this paper is uniform.

Assume that some family $\mathcal{R} = (R_n)$ solves a problem P, and that each R_n, $n > 0$, uses $p(n)$ processors, $b(n)$ buses, and has a running time $t(n)$. We say then that $P \in \text{RMBM}(p(n), b(n), t(n))$ (or $P \in \text{F-DRMBM}(p(n), b(n), t(n))$, etc.), and that $\mathcal{R}$ has *size complexity* $p(n) \times b(n)$ (it is customary [14, 21] to consider the size of a network as being the product between the number of processors and

the number of buses) and *time complexity* $t(n)$.

It should be noted that, as shown above, a directed RMBM can simulate a nondirected RMBM by simply keeping all the up and down fuse lines synchronized with each other:

Observation 1 $XY\text{RMBM}(x(n),y(n),z(n)) \subseteq XY\text{DRMBM}(x(n),y(n),z(n))$ for any $x,y,z : \mathbb{N} \to \mathbb{N}$, $X \in \{\text{CRCW},\text{CREW}\}$, and $Y \in \{\text{F-},\lambda\}$.

The *bus width* of some RMBM (DRMBM, etc.) denotes the maximum size of a word that may be placed (and read) on (from) any bus in one computational step. It is immediate that the bus width of any RMBM from an RMBM family is upper bounded by $O(\log n)$.

The reconfigurable network An RN [4] is a network of processors that can be represented as a connected graph whose vertices are the processors and whose edges represent fixed connections between processors. Each edge incident to a processor corresponds to a (bidirectional) port of the processor. A processor can internally partition its ports such that all the ports in the same block of that partition are electrically connected (or fused) together. Two or more edges that are connected together by a processor that fuses some of its ports form a bus which connects ports of various processors together. CREW, Common CRCW, Collision CRCW, etc. are defined as for the the RMBM model.

The *directed* RN (DRN for short) is similar to the general RN, except that the edges are directed. The concept of (uniform) RN family is identical to the concept of RMBM family. The class $\text{RN}(s(n),t(n))$ ($\text{DRN}(s(n),t(n))$) is the set of problems solvable by RN (DRN) uniform families with $s(n)$ processors ($s(n)$ is also called the *size complexity*) and $t(n)$ running time.

3 RMBM and NLOGSPACE computations

In this section, we first show that the *graph accessibility problem* (GAP) can be solved by a DRMBM in constant time. Then, we investigate the relation between RMBM and NLOGSPACE computations. We show that RMBMs with polynomially bounded resources and constant running time recognize exactly all the languages in NLOGSPACE.

Definition 3.1 (Graph accessibility problem) $GAP_{1,n}$ denotes be the following problem: Given a directed graph $G = (V,E)$, $V = \{1,2,...,n\}$ (expressed, for example, by the (boolean) incidence matrix I), determine whether vertex n is accessible from vertex 1. In general, the problem of determining whether vertex j is accessible from vertex i is denoted by $GAP_{i,j}$.

Lemma 3.1 $GAP_{1,n} \in \text{CRCW F-DRMBM}((n^2-n)/2,n,2)$. *Furthermore, the F-DRMBM family solving* $GAP_{1,n}$ *uses the Collision resolution rule and has bus width* 1.

Proof. The following RMBM algorithm is a variant of the algorithm that computes the shortest path in a directed graph [14] (which is itself an adaptation of the algorithm for the minimum spanning tree [20]). However, we are not interested in the length of an eventual path, so that our construction requires considerably less resources.

For convenience, each processor is denoted by p_{ij}, $1 \le i < j \le n$. When we say that some processor fuses buses k and l, we imply that this fusion is directional, such that a signal placed on bus k is seen on bus l, but not vice versa. We assume that each processor p_{ij} knows the value of both I_{ij} and I_{ji}, where I is the incidence matrix. Then, the algorithm performs the following steps:

1. Each processor p_{ij}, $1 \leq i < j \leq n$ fuses buses i and j iff $I_{ij} = True$. Simultaneously, p_{ij} fuses buses j and i iff $I_{ji} = True$.

2. p_{13} places a signal on bus 1, and p_{12} listens to bus n. p_{12} reports[‡] *True* if it receives some signal (either the original one emitted by p_{13} or the signal corresponding to a collision), and *False* otherwise.

Note that, even if only one processor writes on the busses, the algorithm cannot be implemented on an exclusive-write RMBM, as the signal emitted by p_{13} may reach some bus on more than one path. We must show that p_{12} reports true iff vertex n is accessible from vertex 1. In fact, it can be easily proved by induction on the length of the path from s to t that, for any s, t, $1 \leq s,t \leq n$, a signal placed on bus s reaches bus t iff vertex t is accessible from vertex s, and this completes the proof (just put $s = 1$ and $t = n$). Indeed, both steps of the algorithm can be clearly performed in one machine cycle each. As well, note that the content of the signal emitted by p_{13} is immaterial, so that a bus width 1 suffices. □

Corollary 3.2 *If the input graph $G = (V,E)$ of $GAP_{1,n}$ is given by a list of vertices L instead of an incidence matrix, then $GAP_{1,n} \in$ CRCW F-DRMBM$(m,n,O(1))$, where $m = |E|$ and $n = |V|$.*

Proof. Identical to the algorithm in the proof of Lemma 3.1 on page 6, except that, at step 1 of the above algorithm, processor p_{ij} fuses buses i and j iff $(i,j) \in L$. □

It is worth mentioning that the algorithm presented in [20] uses a CREW DRMBM (as opposed to the CRCW F-DRMBM used in Lemma 3.1 on page 6 and Corollary 3.2). Furthermore, this algorithm computes the shortest path between two vertices. Therefore, it implicitly computes $GAP_{1,n}$. This lets us conclude that $GAP_{1,n} \in$ CREWDRMBM$(2mn, n^2, O(1))$. However, in what follows, we will use the result based on the CRCW F-DRMBM since, on one hand, it uses resources more efficiently, and, on the other hand, we believe that a Collision conflict resolution rule is just as realistic as exclusive write.

Consider now some language L in NSPACE$(\log n)$. It follows that there exists a nondeterministic Turing machine $M = (K, \Sigma, \delta, s_0)$ that accepts L and uses $O(\log n)$ working space (by abuse of notation, we call M an NSPACE$(\log n)$, or NLOGSPACE, Turing machine). Without loss of generality, consider that the working (and input) alphabet of M is $\Sigma = \{0,1\}$. Let k be the number of states of M, that is, $k = |K|$. The transition function is denoted by δ, $\delta : (K \times \Sigma \times \Sigma) \rightarrow \mathcal{P}((K \cup \{h\}) \times (\Sigma \cup \{L,R\}) \times \{L,R\})$, and the initial state by s_0. For the sake of simplicity, we consider that M has one working tape only (the extension for multiple working tapes is immediate [12, 18]). It should be noted that M also has a (read-only) input tape.

A *configuration* of M working on input x is defined as containing the current state, the content of its tapes, and the head position on each tape. Denote such a configuration by (s,i,w,j), where s is the state, i and j are the positions of the heads on input and working tape, respectively, and w is the content of the working tape. Note that the content of the input tape is established at the beginning of the computation (indeed, the input tape contains the input x) and does not change. Therefore, the input tape does not change the configuration, except for its head position. For two configurations v_1 and v_2, we write $v_1 \vdash v_2$ iff v_2 can be obtained by applying δ exactly once on v_1.

Since M is nondeterministic, the set of possible configurations of M working on x forms a directed graph (denote it by $G(M,x) = (V,E)$) as follows: V contains one vertex for each and every possible configuration of M working on x, and $(v_1,v_2) \in E$ iff the configuration corresponding to v_2 can be reached

[‡] In fact, neither p_{13} nor p_{12} have any special characteristics, and any pair of distinct processors will do.

from the configuration corresponding to v_1 in one step of M (that is, iff $v_1 \vdash v_2$). In the following, we refer to both a configuration and the vertex denoting that configuration in the associated graph simply as "configuration," as long as the exact meaning is understood from the context.

It is clear that $x \in L$ iff some configuration (h, i_h, w_h, j_h) is accessible in $G(M,x)$ from the initial configuration (s_0, i_0, w_0, j_0). One should also note that there are $poly(n)$ possible configurations of M. Indeed, for any configuration (s,i,w,j), i can take $n = |x|$ values. Furthermore, since $|w| = O(\log n)$, there are at most $poly(n)$ possible contents of the working tape, and j can take $O(\log n)$ values.Given that the set of states K is fixed, the number of possible configurations is $poly(n)$.

Therefore, for any language $L \in \text{NSPACE}(\log n)$ and for any x, determining whether $x \in L$ can be reduced to the problem of computing the graph accessibility problem (GAP) for the graph $G(M,x) = (V,E)$, where M is some Turing machine deciding L, $M \in \text{NSPACE}(\log n)$. In fact, a stronger result is immediate: Given x, L, M, and $G(M,x)$ as above, we consider without loss of generality that the initial state is represented by vertex 1 and the (unique) final state by vertex n in $G(M,x)$. Then, any problem in $\text{NSPACE}(\log n)$ can be reduced to $GAP_{1,n}$. Indeed, we are interested only in the reachability of vertex n (final state) from vertex 1 (initial state).

Lemma 3.3 *Fix a language $L \in \text{NSPACE}(\log n)$. Let $M = (K, \Sigma, \delta, s_0)$ be an $\text{NSPACE}(\log n)$ Turing machine that accepts L. Then, given some word x, $|x| = n$, there exists a CREW F-DRMBM algorithm that computes $G(M,x)$ (as an incidence matrix) in $O(1)$ time, and uses $poly(n)$ processors and $poly(n)$ buses of width* 1.

Proof. The configurations of $G(M,x)$ do not depend on x, but only on M. Therefore, we consider that these configurations are known in advance. That is, the set V of vertices of $G(M,x)$ is known beforehand, even if the set E of edges changes with x. In addition, the transition function δ is known to all the processors.

Put $n' = |V|$ ($n' = poly(n)$). Then, the RMBM algorithm uses $(n + (n'^2 - n')/2)$ processors, as follows: The first n processors, denoted by p_i, $1 \le i \le n$, contain the current input x (in the sense that each p_i contains x_i, the i-th symbol of x). At the beginning of each computational step, p_i writes x_i to bus i. Since $x_i \in \{0,1\}$, a bus width 1 is enough.

We shall refer to the remaining $(n'^2 - n')/2$ processors as p_{ij}, $1 \le i < j \le n'$. Initially, a processor p_{ij} holds a false initial value for the elements I_{ij} and I_{ji} of the incidence matrix I. Then, each p_{ij} considers the (potential) edges (v_i, v_j) and (v_j, v_i) corresponding to I_{ij} and I_{ji}, respectively. If such edge(s) exist, p_{ij} writes $True$ to I_{ij} and/or I_{ji} as appropriate. Otherwise, it does nothing. There is no interprocessor communication between processors p_{ij}, $1 \le i < j \le n'$, thus any RMBM model is able to carry on this computation.

It remains to show that determining whether there exists an edge (v_i, v_j) is computable in constant time by one processor (p_{ij} or p_{ji}). Clearly, given a configuration v_i, p_{ij} can compute in constant time any configuration v_l accessible in one step from v_i (if $v_i = (s,z,w,y)$, then v_l is obtained by possibly changing the state s, incrementing, decrementing or keeping z and/or y unchanged, and changing at most one symbol from w, everything according to δ). Recall now that $\delta : (K \times \Sigma) \to \mathcal{P}((K \cup \{h\}) \times (\Sigma \cup \{L,R\}) \times \{L,R\})$, and note that $|\mathcal{P}((K \cup \{h\}) \times (\Sigma \cup \{L,R\}) \times \{L,R\})| = O(2^k)$ (since $|\Sigma| = 2$, and $|K| = k$). That is, the number of configurations that are accessible from some given configuration is constant ($O(2^k)$). In other words, p_{ij} computes (in constant time) a constant number (at most $O(2^k)$) of possible configurations. Note that, in addition, p_{ij} can hold s and w in two of its registers, and it has access to any symbol x_i of the input by simply reading bus i. After this, p_{ij} can decide whether v_j is accessible from v_i in constant time

by simply checking the membership of v_j in the set of the newly computed configurations. It follows that p_{ij} computes I_{ij} and I_{ji} in constant time, and this completes the proof. □

Some comments on the RMBM algorithm developed in the proof of Lemma 3.3 on page 8 are in order. One can note that the constant running time of this algorithm may be quite large ($O(2^k)$; furthermore it depends on the number of states in the initial Turing machine). On the other hand, the subsequent use of Lemma 3.3 will emphasize the need for the RMBM algorithm to be as fast as possible. Thus, even if theoretically sound, the dependency of the running time to the number of states is not a desirable feature.

However, given some nondeterministic Turing machine $M = (K, \Sigma, \delta, s_0)$, one can build an equivalent Turing machine $M' = (K', \Sigma', \delta', s_0)$ such that, for any $s2 = \delta'(s1)$, $|s2| \leq 2$. Indeed, take some state $s \in K$ such that $S' = \delta(s, \alpha, \beta)$ for some $\alpha, \beta \in \Sigma$, and $|S'| > 2$. Then, introduce a set K_s of new, distinct states (which do not change the tapes' content or head positions) to K', such that the graph corresponding to δ' restricted to $K_s \cup \{s\}$ is a binary tree rooted at s, with exactly all the terminal nodes in S', and with all the nonterminals (except the root) from K_s. Clearly, M' is equivalent to M, in the sense that they accept the same language and use the same amount of space.

One can now build the algorithm A from Lemma 3.3 based on M' instead of M. Then, although $G(M,x)$ may grow (still, $|V|$ remains $O(n)$), the running time of A is now upper bounded by a very small constant, and this constant no longer depends on the number of states of M (or M' for that matter).

From Lemma 3.1 on page 6 and Lemma 3.3 on page 8, it follows that

Lemma 3.4 NLOGSPACE $\subseteq$ CRCW F-DRMBM$(poly(n), poly(n), O(1))$, *with Collision resolution rule and bus width* 1.

Proof. Given some language L in NSPACE$(\log n)$, let M be the (NSPACE$(\log n)$) Turing machine accepting L. For any input x, the F-DRMBM algorithm that accepts L works as follows: Using Lemma 3.3 on page 8, it obtains the graph $G(M,x)$ of the configurations of M working on x (by computing in effect the incidence matrix I corresponding to $G(M,x)$). Then, it applies the algorithm from Lemma 3.1 on page 6 in order to determine whether vertex n (halting/accepting state) is accessible from vertex 1 (initial state) in $G(M,x)$, and accepts or rejects x, accordingly. In addition, note that the values I_{ij} and I_{ji} computed by (and stored at) p_{ij} in the algorithm from Lemma 3.3 are in the right place as input for p_{ij} in the algorithm from Lemma 3.1. It is immediate given the aforementioned lemmas that the resulting algorithm accepts L and uses no more than $poly(n)$ processors and $poly(n)$ buses of constant width. □

Conforming to Lemma 3.4, any NLOGSPACE language can be accepted in constant time by a directed RMBM. In fact, the relation between directed RMBMs and NLOGSPACE languages is even stronger:

Lemma 3.5 CRCW DRMBM$(poly(n), poly(n), O(1)) \subseteq$ NLOGSPACE, *for any write conflict resolution rule and any bus width.*

Proof. Let R be some RMBM in CRCW DRMBM$(poly(n), poly(n), O(1))$ performing step d of its computation ($d \leq O(1)$). Suppose that there exists a Turing machine M_d that generates the description of R after step d using $O(\log n)$ space. Then, by standard techniques [12], one can modify M_d (obtaining M'_d) such that M'_d receives n and some i, $1 \leq i \leq n$, and outputs the ($O(\log n)$ long) description for processor i instead of the whole description. We establish the existence of M_d (and thus M'_d) by induction over d, and thus we complete the proof.

Clearly, M_0 exists by the definition of a (uniform) RMBM family. We now assume the existence of M_{d-1} (M'_{d-1}) and show how M_d is constructed. For each processor p_i and for bus k read by p_i during

step d, M_d performs (sequentially) the following computation: M_d maintains two words b and ρ, initially empty. For every p_j, $1 \leq j \leq poly(n)$, M_d determines whether p_j writes on bus k. This implies the computation of $GAP_{j,i}$. $GAP_{j,i}$ is clearly computable in nondeterministic $O(\log n)$ space (it is a simplification of the Graph Accessibility Problem, which is NLOGSPACE-complete [18]; the local configurations of fused and segmented busses at each processor are obtained by calls to M'_{d-1}). If p_j writes on bus k, then M_d uses M'_{d-1} to determine the value v written by p_j, and updates b and ρ as follows: If b is empty, then it is set to v (p_j is currently the only processor that writes something to bus k), and ρ is set to j. Otherwise,

1. If R uses the Collision resolution rule, the collision signal is immediately placed in b. The value of ρ is immaterial.
2. When the Common rule is used, M_d compares b and v. If they are different, the input is rejected immediately. The value of ρ is again immaterial.
3. If the conflict resolution rule is Priority, ρ and j are compared; if the latter denotes a processor with a larger priority, then b is set to v and ρ is set to j. Otherwise, neither b nor ρ are modified. The Arbitrary rule is handled similarly, except that the decision whether to modify b and ρ is made arbitrarily instead of being based on the values of j and ρ.
4. If R uses the Common resolution rule with $\circ$ as combining operation, b is set to the result of $b \circ v$. The operation can be performed in $O(\log n)$ space, since the length of both b and v is $O(\log n)$, and $\circ$ is computable in linear space. As well, the operation $\circ$ is associative. It follows that, once all the processors p_j have been considered, the content of b is the correct combination of all the values written on bus k.

Once the content of bus k has been determined, the configuration of p_i is updated accordingly, b and ρ are reset to the empty word, and the same computation is performed for the next bus read by p_i or for the next processor.

The space required by M_d is the space for the configuration of p_i itself, plus the space for the configuration of one other processor, plus the space required b, v, and ρ. The latter three values cannot be of size larger than $O(\log n)$ (since the word size of any processor is $O(\log n)$ and the number of processors is $poly(n)$), and the configurations clearly take $O(\log n)$ space. Thus, the whole computation of M_d takes $O(\log n)$ space, and the induction is complete. □

Lemma 3.4 on page 9 and Lemma 3.5 on page 9 imply the following results:

Theorem 3.6 $\text{CRCW DRMBM}(poly(n), poly(n), O(1)) = \text{NLOGSPACE}$, *for any write conflict resolution rule and any bus width.*

For any write conflict resolution rule and any bus width, $\text{CRCW DRMBM}(poly(n), poly(n), O(1)) = \text{CRCW F-DRMBM}(poly(n), poly(n), O(1))$ *with Collision resolution rule and bus width* 1.

Corollary 3.7 $\text{DRMBM}(poly(n), poly(n), O(1)) = \text{DRN}(poly(n), O(1))$.

Proof. Immediate from Theorem 3.6 on page 10, since $\text{NLOGSPACE} = \text{DRN}(poly(n), O(1))$ [4]. □

The following is a generalization of Theorem 3.6 on page 10:

Corollary 3.8 *For any problem P solvable in constant time by some (directed or nondirected) RMBM family using $poly(n)$ processors and $poly(n)$ buses, it holds that* $P \in \text{CRCW F-DRMBM}(poly(n), poly(n), O(1))$ *with Collision resolution rule and bus width* 1.

Proof. From Theorem 3.6 on page 10 and Observation 1 on page 6. □

We note that the power of (nondirected) RMBMs has been investigated in [21], where it is shown that nondirected RMBMs are exactly as powerful as nondirected RNs, and that the Collision, Common, Arbitrary, and Priority rules are equivalent in power. In addition, RNs (and thus RMBMs) solve in constant time exactly all the problems in LOGSPACE [4]. By Theorem 3.6 on page 10 and Corollary 3.7 on page 10 we extend these results to the directed variants of RMBMs and RNs running in constant time. As expected, DRMBMs, DRNs, and logarithmic space bounded nondeterministic Turing machines are found to have the same computational power. Corollary 3.8 on page 10 shows that, again, the Collision, Common, Arbitrary, and Priority rules are equivalent to each other. In addition though we show that a resolution rule apparently much more powerful than the others, namely Combining, adds no computational power either. Then, for constant time computations on DRMBM, bus width does not matter; any problem can be solved using buses of unitary width. Finally, as is the case of (undirected) RMBMs, it follows from Corollary 3.8 on page 10 that segmenting buses does not add computational power over fusing buses.

4 Small space computations are real-time

We have now all the necessary ingredients to state the first result linking real time with logarithmic space computations. First though, we have to make an additional assumption: We henceforth consider that the deadlines imposed on real-time computations are reasonably large compared to the processor clock frequency. We believe that this is a reasonable assumption. Indeed, nowadays processors operate at frequencies around (and sometimes exceeding) 1GHz; still, we are not aware of any real-time application that requires deadlines measured in nanoseconds.

Note now that the potential existence of a *deadline* can be modeled as a well-behaved timed ω-word [5] by $W_d = (\sigma, \tau)$, where, for some special, designated symbols w and d,

(i) $\sigma_i = w$ and $\tau_i = i$ for $i > 0$; or

(ii) $\sigma_1 \in \mathbb{N} \cap [max, 0)$, $\tau_1 = 0$; for $i > 0$, if $\tau_i < t_d$, then $\tau_i = i$ and $\sigma_i = w$. Let i_0 be the index such that $\tau_i = t_d$. Then, for all $i \geq i_0$, $\tau_i = i_0 + \lfloor (i - i_0)/2 \rfloor$, and

$$\sigma_i = \begin{cases} d & \text{if } i - i_0 \text{ is even} \\ 0 & \text{otherwise; or} \end{cases} \tag{1}$$

(iii) Same case (ii), except that equation (1) becomes

$$\sigma_i = \begin{cases} d & \text{if } i - i_0 \text{ is even} \\ u(\tau_i) & \text{otherwise.} \end{cases} \tag{2}$$

The above description of W_d has the following semantics: The special symbol w is present whenever the current time does not exceed the deadline; if the deadline passed, then the symbols that arrive as input are all d. If the computation is completed at a moment in which the input symbol is w, then it has met the associated deadline; otherwise, the deadline has passed.

Case (i) models a computation without deadlines. Such a case is provided for completeness, since, even in a real-time environment, it is possible that some tasks have no associated deadline. Provided that it terminates at all, any such a computation meets its deadline (that is, terminates at some time when the arriving input is w). Case (ii) represents a computation with a firm deadline at time t_d. A computation completing after the deadline is useless, and this is expressed by the presence of the zeroes (meaning zero utility) arriving together with the symbols d that signal the fact that the deadline has passed. Finally, case (iii) models a computation featuring a soft deadline at time t_d with the utility function $u : \mathbb{N} \cap [t_d, \omega) \to \mathbb{N} \cap [0, max]$. At any moment $t > t_d$, the signal d comes together with the usefulness of the associated computation (between 0, meaning useless, and some maximum value *max*), provided that the computation completes at time t.

With this definition of W_d, and for any problem $P \in \text{NSPACE}(\log n)$, let $P_\tau = \{(\sigma\sigma_d, \tau) | \ \sigma \text{ is some input for } P, \sigma_d = \text{detime}(W_d) \text{ for some timed word } W_d \text{ modeling a deadline, and } \tau \text{ is some well-behaved time sequence}\}$. In other words, P_τ represents the problem P in the (potential) presence of deadlines. Then, the relation between NLOGSPACE and real-time computations can be informally stated as follows: Suppose one has a (possibly infinite) set of inputs for a bunch of problems in NLOGSPACE. We impose some (any) deadline for each of these inputs, and we feed them at various time moments to some machine. If that machine happens to be a CRCW F-RMBM, then it is able to handle the input successfully. Formally, given Theorem 3.6 on page 10 (and noting that the size complexity of an RMBM with $poly(n)$ processors and $poly(n)$ buses is $poly(n)$), we have the following relation linking NLOGSPACE with real-time computations.

Theorem 4.1 $\bigcup \left(\prod_{P \in \text{NSPACE}(\log n)} P_\tau \right) \subseteq \text{rt-PROC}^{\text{CRCW F-DRMBM}}(poly(n))$, *where n is the maximum input size for problems P.*

Proof. All the processing implied by Theorem 3.6 on page 10 (namely, the algorithms from Lemmas 3.1 on page 6 and 3.3 on page 8) takes very little (and constant) time, and thus accommodates any reasonable (in the sense of the above assumption) time sequence τ associated with the computation. □

In some sense, one may argue that the inclusion relation from Theorem 4.1 is in fact an equality, conforming to Theorem 3.6 on page 10. Indeed, NLOGSPACE computations are *the only* computations in the classical sense that can be performed in constant time by DRMBMs, no matter how many processors and buses are used; thus, given any deadline (in effect imposing a constant upper bound on the running time), it follows that no computation outside NLOGSPACE can be successfully carried out. However, this inclusion cannot be improved upon, since there might exist real-time computations (for example, not exhibiting explicit deadlines and thus not necessarily having constant time constraints) that are not in NLOGSPACE but can still be performed within the given resource bounds (that is, a polynomial number of processors and buses).

Indeed, one candidate for such computations can be the family of timed ω-languages PURSUIT_k, $k \geq 1$, presented in [8] and summarized in Section 2.1 on page 3. Those languages, modeling the k-dimensional version of the *pursuit and evasion on a ring* problem, do not feature explicit deadlines. The real-time qualifier is instead given by the "movements of the pursuee," that is, by the real-time input arrival. We shall try to see what is the classical computation corresponding to this problem.

In Theorem 4.1, we *added* deadlines (that is, real-time constraints) to problems. We face now the reversed problem, namely how can one *eliminate* the real-time qualifier from the specification of some problem. Analyzing the form of the word W_d modeling deadlines offers the clue. Indeed, one can notice that, from some time on, the symbols from W_d no longer represent the input. Instead, they consists of

symbols w and d that model the timing constraints imposed on the computation. Similarly, in a real-time problem for which the input is virtually endless, a prefix of that input represents the same problem, except that in the case of such a prefix, the input "stops coming" at some time. This is the most general restriction to a classical environment one can model, since the input is finite in such an environment:

Definition 4.1 Consider some well-behaved timed omega-language L. For some $(\sigma,\tau) \in L$, $i > 0$ is a *progression point* iff[§] $\tau_i \neq \tau_{i+1}$.

Let $L_s = \{\sigma' | \text{there exists some finite progression point } n \text{ such that } (\sigma,\tau) \in L \text{ and } \sigma' = \sigma_{1\ldots n}\}$ (each word in L_s is constructed by taking a word from L, restricting its length to some finite n, and discarding the time sequence). If, for some complexity class C, $L_s \in C$, then we say that $L \in C/rt$ (L is the *real-time counterpart* of L_s; alternatively, L_s solves the same problem as L, but without real-time constraints, and thus L_s is the *static version* of L).

Note in passing that Definition 4.1 on page 13 not only allows us to study the pursuit problem in the context of Theorem 4.1 on page 12, but it offers a more concise formulation of Theorem 4.1 itself:

Theorem 4.2 $\text{NSPACE}/rt(\log n) \subseteq \text{rt-PROC}^{\text{CRCW F-DRMBM}}(poly(n))$.

It is immediate that the two formulations are equivalent, while the one expressed by Theorem 4.2 is easier to understand.

We now show that pursuing something is easy outside the real-time paradigm: Recall from Section 2.1 on page 3 that PURSUIT_k denotes the "k-dimensional version" of the pursuit and evasion problem [8]. Then,

Theorem 4.3 *For any $k > 0$, $\text{PURSUIT}_k \in \text{SPACE}/rt(\log n)$.*

Proof. Let C be a class such that $\text{PURSUIT}_k \in C/rt$. We shall show that $C = \text{LOGSPACE}$ and we are done. According to Definition 4.1 on page 13, a word w_s in the static version of PURSUIT_k has the following structure: Denote $|w_s|$ by n; then, w_s contains

- An initial word $w^0 \in \{a,b\}^r$ for some $r \leq n$; this is the initial configuration, which the pursuee modifies as time passes.
- Some number m of moves by the pursuee (denoted by some words $w^i \in L_{ci}$, $1 \leq i \leq m$); such a move in effect changes a maximum of p symbols from w^0, $p < r$.

It is clear that $r, p, m \leq n$, since n is the length of the whole input. Consider now a deterministic Turing machine M accepting the static version of PURSUIT_k. In order to determine the number of a's and b's in w^0, M simply keeps two counters C_a and C_b, one for a's and the other for b's, respectively. As the input is scanned, the two counters are incremented accordingly.

Once the end of w^0 is reached, M performs the following step for each w^i, $1 \leq i \leq m$: M identifies that portion of w^0 which is changed by w^i. Then, M scans this portion, decrementing C_a or C_b for each a or b it encounters during this procedure. Finally, M identifies that portion of w^i that changes w^0 and scans it, incrementing C_a and/or C_b accordingly. It is clear that, at the end of step m of such a computation, C_a and C_b contain precisely the number of a's and b's, respectively, that are present in w^0 as it is changed by all

[§] One does not want to split a bunch of symbols arriving at the same time, since such a bunch often represents a nondivisible piece of the input ...

w^i, $1 \leq i \leq m$. Therefore, when the end of the input is reached, M simply compares C_a and C_b and accepts the input iff they are identical.

Clearly, C_a and C_b take $\log r$ space each (since there are at most r a's and at most r b's in w_0). The identification procedure mentioned above uses two pairs of counters, each pair delimiting the portions of interest of w^0 and the current w^i, respectively. Each of these four counters holds an index in the current input, hence it can be stored in $\log n$ space. Finally, setting these counters involves simple arithmetic operations on indices (that is, numbers bounded above by n), hence they are computable in LOGSPACE. Therefore, the space required by the whole computation is $O(\log n)$, as desired. □

Theorem 4.3 on page 13 is an interesting result. Indeed, even if PURSUIT$_k$ is a problem that requires a lot of computational effort (in particular, it cannot be solved at all if less than $2k$ processors are available [8]), it becomes a very simple problem (not only in NLOGSPACE, but even in LOGSPACE) once the real-time constraints are eliminated. Thus, Theorem 4.3 justifies the following conjecture:

Claim 1 $\text{NSPACE}/rt(\log n) = \text{rt-PROC}^{\text{CRCWF-DRMBM}}(poly(n))$.

5 Independence systems and real-time computation

We focus our attention now to optimization problems. In this context, we identify the class of such problems that can be computed in real time if a parallel machine is used. Based on this identification, we also extend previous results [2].

5.1 Independence systems and matroids

If S is a set referred to as the *set of feasible solutions*, over which a mapping c is defined ($c : S \to \mathbb{R}$), then a problem of the form

$$\{\max c(s) | s \in S\} \quad \text{or} \tag{3}$$

$$\{\min c(s) | s \in S\} \tag{4}$$

is an *optimization problem* over S. Form (3) defines a *maximization problem*, while form (4) is a *minimization problem*; c is referred to as the *objective function*. In the following, we shall refer to maximization problems whose set of feasible solutions contains only elements of $\{0,1\}^n$. In this case, S can be considered a subset of $\mathcal{P}(E)$, with $E = \{1,2,\ldots,n\}$. Therefore, problems of the form (3) can be restated as

$$\max \left\{ \sum_{i \in R} c_i \,\middle|\, R \subseteq E \text{ and } R \in S \right\}. \tag{5}$$

Notice that in this case $c(s)$ is implicitly defined as $\sum_{i \in s} c_i$ for any set $s \subseteq E$. We consider without loss of generality that $c_i \geq 0$, $1 \leq i \leq n$. The set of optimal solutions to the maximization problem (5) is thus not changed if one replaces S by its *hereditary* closure S^* defined as $S^* = S \cup \{s | s \subseteq s', s' \in S \text{ for some } s' \subseteq E\}$. (E, S^*) is an *independence system* as per Definition 5.1 on page 15.

Definition 5.1 [13] Let E be a finite set and $S \subseteq \mathcal{P}(E)$, such that S has the *monotonicity property*: $s_1 \subseteq s_2 \in S \Rightarrow s_1 \in S$. Then, (E,S) is an *independence system*, and members of S are said to be *independent*.

Let (E,S) be an independence system. For each $F \subseteq E$, the *lower rank* $lr(F)$ (*upper rank* $ur(F)$) of F (with respect to S) is defined as the cardinality of the smallest (largest) maximal independent subsets of F: $lr(F) = \min\{|s||s \in S; s \subseteq F \text{ and } s \cup \{e\} \in S \text{ for all } e \in F \setminus \{s\}\}$; $ur(F) = \max\{|s||s \in S; s \subseteq F\}$.

A *greedy algorithm* for problem (5) on general independence systems is given in [13]:

algorithm GREEDYMAX $(E,S;s_g)$
1. let $(e_1,e_2,\ldots,e_n)$ be an ordering of E with $c(e_i) \geq c(e_{i+1})$
2. $s_g \leftarrow \emptyset$
3. **for** $i \leftarrow 1 \ldots n$ **do**
3.1. **if** $s_g \cup \{e_i\} \in S$ **then** $s_g \leftarrow s_g \cup \{e_i\}$

Proposition 5.1 [13] *Let (E,S) be an arbitrary independence system, s_g the solution returned by algorithm* GREEDYMAX*, and s^* the optimal solution of (5). Then, for any weight function $c : E \to \mathbb{R}^+$,*

$$\min_{F \subseteq E} \frac{lrF}{urF} \leq \frac{c(s_g)}{c(s^*)} \leq 1.$$

It should be noted that the algorithm GREEDYMAX contains one statement which depends on the actual independence system being considered, namely the boolean expression on line 3.1. Indeed, for a general independence system one does not know how the check "$s_g \cup \{e_i\} \in S$" is done. Thus, in order to analyze the complexity of such an algorithm, one can assume the existence of an *oracle* that can answer whether some set s is in S or not.

Definition 5.2 [13] An independence system (E,S) is called a *matroid* if, for any $F \subseteq E$, it holds that $lr(F) = ur(F)$.

From Proposition 5.1 on page 15 and Definition 5.2 it follows that:

Corollary 5.2 *Algorithm* GREEDYMAX *on a matroid (E,S) yields the optimal solution for (5) for all objective functions $c : E \to \mathbb{R}^+$.*

5.2 A real-time perspective

To put Definition 5.2 on page 15 in another way [10, 12], matroids are independence systems with the additional property that all the maximal independent subsets have the same size (therefore, since $c_i \geq 0$, $1 \leq i \leq n$, the greedy algorithm obtains the optimal solution). In light of this formulation, the parallel implementation of GREEDYMAX is immediate [9, 12]:

algorithm PARALLELGREEDYMAX $(E,S;s_g)$
1. sort E, obtaining $(e_1,e_2,\ldots,e_n)$ s.t. $c(e_i) \geq c(e_{i+1})$
2. $s_g \leftarrow \emptyset$; $r_0 \leftarrow 0$
3. **for** $i \leftarrow 1 \ldots n$ **do in parallel**
3.1. $r_i \leftarrow ur\{e_1,e_2,\ldots,e_i\}$
3.2. **if** $r_{i-1} < r_i$ **then** $s_g \leftarrow s_g \cup \{e_i\}$

Algorithm PARALLELGREEDYMAX uses a *rank oracle*: The function $ur\{e_1, e_2, \ldots, e_i\}$ introduced by Definition 5.1 on page 15 and used at step 3.2 gives the size of some (hence, whenever (E,S) is a matroid, any) maximal independent set over $\{e_1, e_2, \ldots, e_i\}$.

Lemma 5.3 *Suppose* $ur\{e_1, e_2, \ldots, e_i\} \in \text{DRMBM}(poly(i), poly(i), t(i))$ *(i.e.,* $ur\{e_1, e_2, \ldots, e_i\}$ *can be computed by a DRMBM in time* $t(i)$ *using a polynomially bounded number of processors and busses). Then,* PARALLELGREEDYMAX $\in \text{DRMBM}(poly(n), poly(n), O(t(n)))$.

In particular, if $t(i) = O(1)$*, then* PARALLELGREEDYMAX $\in \text{DRMBM}(poly(n), poly(n), O(1))$.

Proof. The initial sorting (step 1) can be achieved in constant time on a DRMBM with polynomially bounded resources [1]. It follows that step 1 is computable in constant time on a DRMBM using $poly(n)$ processors and $poly(n)$ busses by Corollary 3.8 on page 10. Steps 2 and 3.2 are trivially computable in constant time with polynomially bounded resources.

However, each on the calls to *ur* in step 3.1 can be performed in $t(n)$ time by using n copies of the RMBM computing *ur*, each of them working independently from each other. Finally, each of the n RMBMs communicate with one other processor. These n new processors implement step 3.2 and report the result. Since both the argument of *ur* and the result returned by this function are polynomial in size, $poly(n)$ busses suffice for such a communication. All the resources are polynomially bounded, and thus PARALLELGREEDYMAX $\in \text{DRMBM}(poly(n), poly(n), O(t(n)))$, as desired.

If $t(i) = O(1)$, PARALLELGREEDYMAX $\in \text{DRMBM}(poly(n), poly(n), O(1))$ is immediate by Corollary 3.8 on page 10. □

Lemma 5.4 *Let* (E,S) *be some independence system,* $E = \{e_1, e_2, \ldots, e_n\}$*, and let A be an algorithm solving a maximization problem of the form (5) over* (E,S)*. Denote by* $t_A(n)$ *(*$t_{ur}(n)$*) the running time of A (the time required to compute* $ur(E)$*) on a DRMBM using a polynomially bounded number of processors and busses. Then,* $t_{ur}(n)$ *is a lower bound for* $t_A(n)$.

Proof. Let $s^* = \{s_1, s_2, \ldots, s_k\}$ be the solution computed by *A*. Since s^* is an optimal solution, it follows that $ur(E) = k$. However, given s^*, k can be computed in constant time on a DRMBM: Assume without loss of generality that the elements of s^* are stored in the registers of n processors p_i, $1 \leq i \leq n$, such that exactly k processors hold one element from s^* each. Then, each processors p_i, $1 \leq i \leq n$, sets a designated register v_i such that $v_i = 1$ if p_i holds a value from s^* and $v_i = 0$ otherwise. Then, a prefix sum over v_i, $1 \leq i \leq n$, computes k. It follows that $|s^*|$ (and thus $ur(E)$) can be computed in constant time given s^*, since prefix sum takes constant time on RMBM [20]. Therefore, $t_{ur}(n) = O(t_A(n))$ (alternatively, $t_A(n) = \Omega(t_{ur}(n))$), as desired. □

Corollary 5.5 *Let* $\mathcal{M}$ *be the class of maximization problems that can be described as a matroid and for which* $ur \in \text{DRMBM}(poly(i), poly(i), O(1))$*. Let P be some maximization problem of form (5) over some independence system* (E,S)*. Then,*

- $P \in \text{DRMBM}(poly(n), poly(n), O(1))$,
- $P \in \text{NLOGSPACE}$, *and*
- $P/rt \in \text{rt-PROC}^{\text{CRCWF-DRMBM}}(poly(n))$,

iff $P \in \mathcal{M}$.

Proof. The "if" part follows from Lemma 5.3 on page 16, and the "only if" part is established by Lemma 5.4 on page 16. □

By Corollary 5.5 on page 16 we have precisely identified—among those optimization problems that can be expressed as independence systems—the class of such problems solvable in parallel real time. We believe that this result may be of interest for at least two reasons:

1. On one hand, consider those independence systems—or problems that can be formulated as such—not in $\mathcal{M}$ (with $\mathcal{M}$ as defined in Corollary 5.5 on page 16). For these problems, finding an exact solution in real time is asymptotically impossible, even if a parallel machine is available (in the sense that the running time of any ($poly(n)$-processor) algorithm solving such a problem exceeds for large enough input size any (implicit or explicit) constant deadline). In such a case, one should probably look for either further restricting the problem (in order to bring it within $\mathcal{M}$), or find a reasonable approximation algorithm that is in NLOGSPACE.

2. On the other hand, Corollary 5.5 on page 16 easily extends previous results, as we shall show in what follows.

5.3 Beyond speedup, revised

The problem of computing the *minimum-weight spanning tree* (MST) of a connected, undirected, and weighted graph in real time is investigated in [2], where it is shown that the best approximate solution to the MST problem returned by a sequential algorithm can be arbitrarily worse than the solution obtained by a parallel algorithm (which actually returns the optimal solution). We shall not, however restrict ourselves to connected graphs, since the extension to unconnected ones (when the tree becomes a forest) is immediate.

One can notice that MST can be trivially transformed from a minimization problem into a maximization one: just negate all the edge weights, and then add to every weight the absolute value of the maximum edge weight. Furthermore, it is immediate that the MST problem can be expressed as a matroid [10]. Thus, we can both tighten and extend the result from [2] by using Corollary 5.5 on page 16.

First, the result in [2] is not tight: Time up to n^{ε}, for some $0 < \varepsilon < 1$, is allowed for each (parallel or sequential) real-time computation leading to the result. This running time, however, asymptotically exceeds any (however large) constant deadline imposed to the computation by some real-time environment. Still, the same result holds for true real-time computations as well.

Indeed, we show in what follows that, for any real-time environment one can encounter, a parallel algorithm can solve MST arbitrarily better than a sequential one. That is, while the parallel implementation is able to return an optimal solution, even an optimal sequential algorithm can only report an approximate result in the limited time which is available due to the real-time constraints. This result, an immediate consequence of Corollary 5.5 on page 16, is given in Lemma 5.6 on page 17 below.

Lemma 5.6 *Let MST denote the problem of computing the minimum-weight spanning forest on undirected and weighted graphs. Then,* $\mathrm{MST} \in \mathrm{DRMBM}(poly(n), poly(n), O(1))$ *(and thus* $\mathrm{MST} \in \mathrm{NLOGSPACE}$, $\mathrm{MST}/rt \in \text{rt-PROC}^{\text{CRCWF-DRMBM}}(poly(n))$*), and the best approximate solution to* MST/rt *returned by a sequential algorithm is arbitrarily worse than the solution obtained by a parallel RMBM algorithm with polynomially bounded resources.*

Proof. Function *ur* for MST can be computed in logarithmic space (and thus in real-time on RMBM): $ur\{e_1, e_2, \ldots, e_i\}$ is simply i minus the number of connected components in the graph induced by $\{e_1, e_2, \ldots, e_i\}$, and can thus be computed by performing a reflexive and transitive closure (which is an NLOGSPACE-complete problem [18]). By Corollary 5.5 on page 16, it follows that MST can be computed *exactly* in real time on an RMBM, no matter how tight the deadlines are.

However, an optimal sequential algorithm solving the same problem has a running time that cannot accommodate even the most generous deadline, and thus a sequential algorithm to some real time variant of MST can only guess *some* solution, and the guess can be arbitrarily bad, as detailed in [2]. □

In fact, the second part of the proof of Lemma 5.6 on page 17 also proves that this type of behavior (namely a parallel algorithm being able to compute an arbitrarily better solution than the optimal sequential one) is not an exclusive feature of the MST problem, but it applies to many more real-time computations instead. Therefore:

Corollary 5.7 *With $\mathcal{M}$ as in Corollary 5.5 on page 16, the best approximate solution to P/rt returned by a sequential algorithm, for any $P \in \mathcal{M}$, is arbitrarily worse than the solution obtained by a parallel RMBM algorithm with polynomially bounded resources.*

In other words, the results from [2] do hold even for the tightest real time environment. In addition, these results are not applicable only to the MST, but to a whole class of problems instead, namely $\mathcal{M}$ from Corollary 5.5 on page 16. That is, there exists not only a problem, but a whole family of them for which a parallel implementation can do something other than speeding up computation, namely improve the offered solution.

6 Real-time approximation schemes

As a consequence of Claim 1 on page 14, the problem of finding approximate solutions computable in real time (in those cases when the exact solution cannot be computed within the given time restrictions) becomes a worthy pursuit. Such an approach is common in classical complexity theory. Indeed, in the sequential case, NP-hard problems are for all practical purpose (unless P equals NP) not computable but for the smallest instances, and thus deterministic polynomial time approximations are usually sought [11]. Similarly, this time in the context of parallel computations, efficient parallel approximations to P-complete (that is, inherently sequential unless NC equals P) problems were also investigated [12].

The identification from Claim 1 on page 14 of NLOGSPACE as the class containing exactly all the problems solvable in real time naturally extends such a search for approximation algorithms: Once a problem is shown as being likely not solvable in real time (that is, not in NLOGSPACE), then an approximate solutions may become attractive. We now offer a incipient discussion on this matter.

First, we define the notion of "good" approximation algorithms by adapting the definitions already used [11, 12] to our framework:

Definition 6.1 Consider some algorithm A working on instance i of a minimization (maximization) problem, and suppose that A delivers a candidate solution with value $A(i)$. With $Opt(i)$ denoting the value of the optimal solution for input i, the *performance ratio* of A on i is $R_A(i) = A(i)/Opt(i)$ ($R_A(i) = Opt(i)/A(i)$). The *absolute performance ratio* of A is defined as $R_A = \inf\{r \geq 1 | R_A(i) \leq r \text{ for all instances } i\}$.

An algorithm A with inputs $\varepsilon > 0$ and $i \in \Pi$ is an *approximation scheme* for Π iff A delivers a candidate solution with performance ratio $R_A(i) \leq 1+\varepsilon$ for all $i \in \Pi$. In addition, if $A \in$ rt-PROC$(poly(|i|))$, then A is a *real-time approximation scheme* for Π.

The body of knowledge regarding NC approximations [12] gives some negative results: Once it is proved that some problem does not admit an NC approximation algorithm, it follows that no NLOGSPACE (and thus real-time) approximation algorithm exists either, since NLOGSPACE $\subseteq$ NC.

Theorem 6.1 *If* P $\neq$ NC*, then there exists no real-time approximation scheme for the following problems:*

- *Lexicografically first maximal independent set [12].*
- *Unit resolution (the problem whether the empty clause can be deduced from a given propositional formula in conjunctive normal form) [17].*
- *Generability (given a finite set W, a binary relation $\bullet$ on W, a subset $V \subseteq W$, and $w \in W$, determine whether w is in the smallest subset of W that contains V and is closed under $\bullet$) [17].*
- *Path systems (given a path system $P = (X,R,S,T)$, $S,T \subseteq X$, $R \subseteq X \times X \times X$, determine whether there exists an admissible vertex in S) [17].*
- *Circuit value [17].*
- *High degree subgraph (given a graph G and an integer k, does G contain an induced subgraph with minimum degree at least k?) for $k \geq 3$ [17].*
- *Linear programming, in both the following cases: the approximation solution should be a vector close to the optimal one, and the approximation solution seeks the objective function to have a value close to optimal [16].*

Proof. It has been proven (reference to proofs are given within the theorem) that any approximation scheme for these problems is P-complete. Since P $\neq$ NC and NSPACE $\subseteq$ NC, Claim 1 on page 14 implies that the above problems do not admit any real-time approximation scheme. □

6.1 Bin packing

We focus now our attention to the *bin packing* problem. True, there is apparently little hope to find real-time approximation schemes for this problem, since bin packing is NP-complete. However, bin packing is closely related to certain scheduling problems (since the item to be packed can be viewed as tasks to be scheduled), and it is thus conceivable that real-time approximation algorithms can be of use for scheduling tasks in real time on a parallel machine (the utility of such a processing being evident). On the other hand, this time with respect to the feasibility of tackling bin packing in a real-time environment, we note that good NC approximation schemes for this problem already exist [3].

The input for the bin packing problem consists in n items, each of size within interval $(0,1)$. The n items should be packed in a minimal number of bins of unit capacity.

One of the successful approaches in developing sequential (that is, in P) bin packing approximation algorithms is the use of simple heuristics. In this respect, one should mention the *first fit decreasing* (FFD) heuristic, which considers the items in nondecreasing order of their size, and places each item

into the first available (that is, with enough free space) bin. Even if simple, the length of the packing returned by FFD, of at most $11/9 \times Opt + 3$ (where Opt is the length of the optimal solution), is a good approximation, qualifying FFD as an approximation scheme. Still, it is not only intuitive that FFD is inherently sequential (that is, P-complete):

Proposition 6.2 [3] *Given a list of items, each of size between 0 and 1, in nonincreasing order, and two indices i and b, it is P-complete to decide whether the FFD heuristic will pack the ith item into the bth bin. This is true even if the item sizes are represented in unary.*

Even if FFD is inherently sequential, an NC algorithm that achieves the same performance as FFD (although by using different techniques) is given in [3]. This algorithm works in two stages, as follows:

1. The first stage packs all the items that have a size of at least $1/6$. Such a stage starts by *sorting* the list of items in nonincreasing order. Then, a constant number of passes are performed, each pass involving two algorithms: (*a*) *merge* two sorted lists of *n* elements each into a sorted list, and (*b*) in a string of length *n* of *opening and closing parentheses*, find the matching pairs.

2. In the second stage, the remaining items are packed. This stage involves a (relatively large) number of *parallel prefix computations*.

Theorem 6.3 *Bin packing admits a real-time approximation scheme A such that* $A(i) \leq 11/9 \times Opt(i) + 3$ *for any instance i.*

Proof. We follow the algorithm from [3], showing how this algorithm can be implemented in real time. According to Theorem 3.6 on page 10, we have a choice of showing that this algorithm is in NLOGSPACE or in DRMBM$(poly(n), poly(n), O(1))$. We chose the latter variant.

First, we note that sorting can be done in constant time on an (nondirected or directed) CREW RMBM using $poly(n)$ processors and $poly(n)$ buses [20]. Then, it is immediate that merging two sequences into a sorted sequence is also computable in constant time on RMBM. Indeed, the quick and dirty method of sorting (using the algorithm mentioned above) the two lists concatenated together will do the trick.

The problem of matching parentheses can be implemented in two steps as follows: First, the unmatched parentheses can be eliminated by a parallel prefix computation. Then, there exists a constant time algorithm on DRN using $poly(n)$ processors for matching the remaining sequence of parentheses [1]. However, this implies the existence of a similar algorithm on RMBM with polynomially bounded number of processors and buses, according to Corollary 3.7 on page 10.

Thus, the only algorithm that is still needed is the parallel prefix computation, which is in CREWRMBM$(poly(n), poly(n), O(1))$ according to [20] (in fact, such an algorithm is the basis for the aforementioned sorting algorithm).

In conclusion, all the algorithms used by the two stages on the NC approximation scheme from [3] are in CREWRMBM$(poly(n), poly(n), O(1))$. Since these algorithms are applied a constant number of times, the whole processing is in CREWRMBM$(poly(n), poly(n), O(1))$ and thus in rt-PROC$(poly(n))$. This completes the proof. □

In passing, one should note that the algorithm from the proof of Theorem 6.3 apparently requires a large (albeit constant) amount of time to complete. However, such a construction is enough to prove that bin packing admits a real-time approximation scheme. Indeed, the existence of an algorithm in CREWRMBM$(poly(n), poly(n), O(1))$ implies the existence of another algorithm, solving the same

problem, but this time in CRCW F-DRMBM($poly(n), poly(n), O(1)$), and whose running time is very small, as shown in Corollary 3.8 on page 10. True, we do not offer a constructive proof for this corollary, and thus the CRCW F-DRMBM($poly(n), poly(n), O(1)$) algorithm cannot be effectively constructed using only the results from this paper. Still, if needed, we believe that, although not a trivial matter, developing such a constructive transformation is feasible.

7 Conclusions

Recently, we addressed a number of questions associated with real-time computations featuring implicit deadlines [8]. In this paper, we focused our attention on computations with explicit deadlines. Specifically, we considered computations that can be performed within specific, fixed deadlines for any input size. Given any language that can be accepted by a machine using logarithmic work space, we showed in Theorem 4.1 on page 12 that such a language can be accepted by a parallel machine with polynomially bounded resources, in the presence of *any* (that is, however tight) real-time constraints.

Theorem 4.3 on page 13 is another interesting result: Even a timed language like PURSUIT$_k$, whose acceptance requires considerable computational effort, can be accepted in logarithmic space once the real-time constraints are dropped. This allows us to state Claim 1 on page 14, which offers a nice counterpart of the parallel computation thesis [12, 15]. In this thesis, NC is conjectured to contain exactly all the computations that admit efficient ($poly(n)$ processors and polylogarithmic running time) parallel implementations. By contrast, we conjecture that NLOGSPACE contains exactly all the computations that admit efficient ($poly(n)$ processors) real-time parallel implementations.

As well, we considered the class of maximization problems over independence systems, showing that a problem pertaining to this class is solvable in real time iff it is a matroid and the size of an optimal solution is computable in real-time. Given this result, we showed that, indeed, there exists not only a problem, but a whole family of them for which a parallel implementation can do something other than speeding up computation, namely unboundedly improve the offered solution.

In light of Claim 1 on page 14, the following research direction becomes useful: Which are those problems that, although possibly not solvable in the real-time environment imposed by some real-time application, admit "good" approximate solutions provably achievable in any real-time environment? Do they form a well-defined complexity class? If so, which are the problems pertaining to such a class? This paper offers a solid basis for the pursuit of this direction, since we identify here a class of candidates for approximating algorithms. In addition, this class of candidates is either NLOGSPACE or F-DRMBM($poly(n)$,$poly(n)$,$O(1)$), whichever is more natural for the given problem, since they are in fact identical as shown by Theorem 3.6 on page 10. As a starter for such a direction, along with identifying some problems not admitting real-time computable approximate solutions, we showed that real-time approximation schemes do exists. Interestingly enough, we found with relative ease such an approximation algorithm for quite a hard (in fact, NP-complete) problem, namely bin packing. This is a nice argument in favor of the relations that we discovered between NLOGSPACE, RMBM, and real-time computations, and a good motivation for the use of timed ω-languages in the study of (approximate or not) real-time computations.

We also determined the computational power of DRMBM running in constant time. We showed that DRMBM and DRN with constant running time have the same computational power. In addition, showed that no conflict resolution rule is more powerful than Collision. According to this result, the discussion regarding the practical feasibility of rules like Priority or Combining on spatially distributed resources such

as a buses is no longer of interest. Indeed, such rules are not only of questionable feasibility, but simply not necessary as well. Finally, we identified a gap in the complexity hierarchy of RMBM computations as well: As far as constant time computations are concerned, there is no need for a large bus width; instead, buses composed of single wires are sufficient.

One interesting open problem naturally arises from the characterization described in the above paragraph: does a form of Corollary 3.8 on page 10 hold for other models of parallel computations? On one hand, we showed in Corollary 3.8 that unrealistic rules like Priority and Combining do not add computational power. However, this result is obtained for the restricted class of DRMBMs running in constant time. Thus, we wonder whether such a result holds for (a) DRMBMs in general, not only those with constant running time, and (b) for other models of parallel computation (RN, PRAM, etc.). On the other hand, we wonder whether the bus width can be bounded for DRNs running in constant time as it has been bounded in the case of DRMBMs. In other words, can the bus width in a DRN be bounded by a constant?

References

[1] S. G. AKL, *Parallel Computation: Models and Methods*, Prentice-Hall, Upper Saddle River, NJ, 1997.

[2] S. G. AKL AND S. D. BRUDA, *Parallel real-time optimization: Beyond speedup*, Parallel Processing Letters, 9 (1999), pp. 499–509. For a preliminary version see http://www.cs.queensu.ca/home/akl/-techreports/beyond.ps.

[3] R. J. ANDERSON, E. W. MAYR, AND M. K. WARMUTH, *Parallel approximation algorithms for bin packing*, Information and Computation, 82 (1989), pp. 262–277.

[4] Y. BEN-ASHER, K.-J. LANGE, D. PELEG, AND A. SCHUSTER, *The complexity of reconfiguring network models*, Information and Computation, 121 (1995), pp. 41–58.

[5] S. D. BRUDA AND S. G. AKL, *Real-time computation: A formal definition and its applications*, to appear in International Journal of Computers and Applications.

[6] ——, *On the necessity of formal models for real-time parallel computations*, Parallel Processing Letters, 11 (2001), pp. 353 – 361.

[7] ——, *Parallel real-time complexity: A strong infinite hierarchy*, in Proceedings of VIII International Colloquium on Structural Information and Communication Complexity, Vall de Núria, Spain, June 2001, Carleton Scientific, pp. 45–59. For an extended version see http://www.cs.queensu.ca/home/-bruda/www/pursuit/.

[8] ——, *Pursuit and evasion on a ring: An infinite hierarchy for parallel real-time systems*, Theory of Computing Systems, 34 (2001), pp. 565–576.

[9] S. A. COOK, *A taxonomy of problems with fast parallel algorithms*, Information and Control, 64 (1985), pp. 2–22.

[10] T. H. CORMEN, C. E. LEISERSON, AND C. STEIN, *Introduction to Algorithms*, MIT press, Cambridge, MA, 2 ed., 2001.

[11] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, 1979.

[12] R. GREENLAW, H. J. HOOVER, AND W. L. RUZO, *Limits to Parallel Computation: P-Completeness Theory*, Oxford University Press, New York, NY, 1995.

[13] R. KANNAN AND B. KORTE, *Approximative combinatorial algorithms*, in Mathematical Programming, R. W. Cottle, M. L. Kelmanson, and B. Korte, eds., Elsevier Science Publishers, Amsterdam, The Nederlands, 1981, pp. 195–248.

[14] N. NAGY, *The maximum flow problem: A real-time approach*, Master's thesis, Department of Computing and Information Science, Queen's University, Jan. 2001.

[15] I. PARBERRY, *Parallel Complexity Theory*, John Wiley & Sons, New York, NY, 1987.

[16] M. J. SERNA, *Approximating linear programming is log-space complete for P*, Information Processing Letters, 37 (1991), pp. 233–236.

[17] M. J. SERNA AND P. G. SPIRAKIS, *The approximability of problems complete for P*, in Optimal Algorithms, International Symposium Proceedings, H. Djidjev, ed., Varna, Bulgaria, May–June 1989, pp. 193–204. Springer Lecture Notes in Computer Science 401.

[18] A. SZEPIETOWSKI, *Turing Machines with Sublogarithmic Space*, Springer Lecture Notes in Computer Science 843, 1994.

[19] A. S. TANENBAUM, *Computer Networks*, Prentice Hall, Upper Saddle River, NJ, 3 ed., 1996.

[20] J. L. TRAHAN, R. VAIDYANATHAN, AND C. P. SUBBARAMAN, *Constant time graph algorithms on the reconfigurable multiple bus machine*, Journal of Parallel and Distributed Computing, 46 (1997), pp. 1–14.

[21] J. L. TRAHAN, R. VAIDYANATHAN, AND R. K. THIRUCHELVAN, *On the power of segmenting and fusing buses*, Journal of Parallel and Distributed Computing, 34 (1996), pp. 82–94.

[22] USENET, *Comp.realtime: Frequently asked questions*, Version 3.4 (May 1998). http://-www.faqs.org/faqs/realtime-computing/faq/.

[23] H. YAMADA, *Real-time computation and recursive functions not real-time computable*, IRE Transactions on Electronic Computers, EC-11 (1962), pp. 753–760.