# A Randomized BSP/CGM Algorithm for the Maximal Independent Set Problem

Afonso Ferreira*
SLOOP
CNRS - INRIA - UNSA
2004, Rt. des Lucioles
F-06902 Sophia Antipolis, France
Afonso.Ferreira@sophia.inria.fr

Nicolas Schabanel
LIP
UMR CNRS - ENS Lyon - INRIA 5668
46 allée d'Italie
F-69364 Lyon Cedex 07, France
Nicolas.Schabanel@ens-lyon.fr

**ABSTRACT**

This paper presents a randomized parallel algorithm for the Maximal Independent Set problem. Our algorithm uses a BSP-like computer with $p$ processors and requires that $\frac{n+m}{p} = \Omega(p)$ for a graph with $n$ vertices and $m$ edges. Under this scalability assumption, and after a preprocessing phase, it computes a maximal independent set after $O(\log p)$ communication rounds, with high probability, each round requiring linear computation time $O(\frac{n+m}{p})$. The preprocessing phase is deterministic and important in order to ensure that degree computations can be implemented efficiently. For this, we give an optimal parallel BSP/CGM algorithm to the $p$-quantiles search problem, which runs in $O(\frac{m \log p}{p})$ time and a constant number of communication rounds, and could be of interest in its own right, as shown in the text.

**Keywords:** *Parallel Algorithms, Maximal Independent Set, Randomized Algorithms, Graph Algorithms, Coarse-Grained Models, BSP, CGM, p-quantiles search, Sorting.*

## 1 Introduction

A Maximal Independent Set (MIS) of an undirected graph is a set of vertices in which no two vertices are connected by an edge and which is maximal for inclusion. If edges represent conflict between nodes, a MIS can be seen as a maximal set of nodes with no conflict. Finding a MIS is a fundamental problem, mainly used as a tool to solve network problems, such as election and deadlock resolution [20], and mobile communications network design [6].

*Partially supported by the NATO project N. 971467: *Mathematical Models and Algorithms for Coarse Grained Parallel Computation.*

We denote by $n$ and $m$, respectively, the number of nodes and edges of the graph.

*Previous work*

The standard sequential greedy MIS algorithm was proved to be $NC^1$-complete, in [7], and a lower bound of $\Omega(\frac{\log n}{\log\log n})$ on the computation time of PRAM algorithms using a polynomial number of processors, was proved in [15]. With respect to upper bounds, a $O(\log^4 n)$ time deterministic PRAM algorithm using $O((\frac{n}{\log n})^3)$ processors was designed in [17], and more efficient randomized PRAM algorithms were proposed to find a MIS in $O(\log n)$ time using $\Theta(m)$ processors [1,19]. These schemes were then derandomized at the cost of an enormous increase in the number of processors, to $\Theta(n^2 m)$ [19].

Motivated by a number of applications where a MIS has to be computed distributedly on a network of communicating elements, distributed algorithms were also designed for this problem. In the case of bounded degree networks, an efficient distributed algorithm was proposed in [14], which finds a MIS in $O(\log^* n)$ time. This result was proved to be optimal in [18]. However, if the network degree is unbounded, the problem is again very difficult to solve deterministically, and the best known deterministic distributed algorithm is intricate and takes $O(n^{o(1)})$ time [3].

*Parallel models of computation*

From a parallel systems viewpoint, theoretical speedups obtained with PRAM algorithms do not necessarily match the speedups observed on real machines [2]. Given sufficient slackness in the number of processors, Valiant's *Bulk Synchronous Parallel* (BSP) approach overcomes this obstacle, yielding optimal simulation of PRAM algorithms by distributed memory parallel systems, in which $g = \frac{\text{local computation speed}}{\text{router bandwidth}}$ is low, or fixed, even for increasing number of processors [23]. Unfortunately, this is not the case for most currently available multiprocessors, and PRAM simulations cannot be performed efficiently, if the factor $g$ is high. Moreover, one may want to design algorithms that utilize local computations and minimize global operations, as pointed out in [23].

The cost of a message also contains a constant overhead cost $s$. The value of $s$ can be fairly large and the total message overhead cost can have a considerable impact on the speedup observed (see e.g. [9]). In this paper, we will use a more practical version of the BSP model, referred to as the *Coarse Grained Multicomputer* (CGM) model [5,9]. It is comprised of a set of $p$ processors $P_1, \ldots, P_p$ with $\frac{\text{size of problem}}{p}$ local memory per processor and an arbitrary communication network. An algorithm consists then of a sequence of supersteps, each alternating local computation and a global communication round. Each communication round consists of routing a single $h$-relation with $h = \frac{\text{size of problem}}{p}$, i.e. each processor can send up to $\frac{\text{size of problem}}{p}$ data and receive up to $\frac{\text{size of problem}}{p}$ data. We require that all information sent from a given processor to another processor in one communication round is packed into one long message, thereby minimizing the message overhead. In the BSP model, a computation/communication round is equivalent to a superstep with $L = \frac{\text{size of problem}}{p} g$ (plus the above "packing requirement").

Finding an optimal algorithm in the coarse grained multicomputer model is equivalent to minimizing the number of communication rounds as well as the total local computation time. This considers all parameters discussed above that are affecting the final observed speedup and it requires no assumption on $g$. Furthermore, it has been shown that minimizing the number of supersteps also leads to improved portability across different parallel architectures ([12,23]). The above model has been used (explicitly or implicitly) in parallel algorithm design for various problems and shown very good practical timing results ([4,11,12]).

*Our work*

In this paper we propose a scalable coarse grained randomized algorithm for computing the MIS of a general graph, in the BSP/CGM model. Our solution is based on a combination of Luby's randomized algorithm [19] and the sequential greedy algorithm, working for any grain $\frac{n+m}{p} = \Omega(p)$. It takes $O(\log p)$ communication rounds (or supersteps), with high probability, and $O(\frac{n+m}{p})$ sequential computation time per round. Note that the number of communication rounds is independent of $n$ and $m$.

In contrast to problems where a natural partition of the input exists (e.g. in planar computational geometry [9]), one of the main challenges we had to face here was how to map the graph onto the processors in order to perform degree computations efficiently. An easy way to get around this problem consists of globally sorting all the edges lexicographically, as proposed in [5]. In the specific case where the nodes are consecutively numbered, integer sorting could be used to implement this phase in linear time [13]. Unfortunately, however, this is rarely the case in real applications, where most of the time inputs are subgraphs of larger instances [3,6]. Hence, in the general case, the method proposed in [5] would require a $\Omega(\frac{m \log m}{p})$ computation step and would then dominate the $O(\frac{n+m}{p} \log p)$ total time of our MIS algorithm.

Therefore, we designed a $p$-quantiles search[1] BSP/CGM algorithm, that yields an efficient partitioning of the graph, allowing for quick degree computations. Our parallel $p$-quantiles search algorithm is deterministic and runs in optimal time $O(\frac{m \log p}{p})$ with a constant number of communication rounds

[1]The *p-quantiles* [8] of an ordered set $A$ of size $n$ are the $p-1$ elements, of rank $\frac{n}{p}$, $\frac{2n}{p}$,..., $\frac{(p-1)n}{p}$, splitting $A$ into $p$ parts of equal size.

and scalability assumption $\frac{n+m}{p} = \Omega(p)$.

It is interesting to notice that our $p$-quantiles search BSP/CGM algorithm could be of interest in connection to BSP/CGM graph and computational geometry algorithms, where $O(\log p)$ communication rounds were also required to solve problems such as connected components, list ranking, and Voronoï diagram [5,10]. Furthermore, it can be used to sort $n$ elements, by sampling, in $O(\frac{n \log n}{p})$ computation time and a constant number of communication rounds, improving the scalability constraints required in [22] (from $\frac{n}{p} = \Omega(p^2)$ to $\frac{n}{p} = \Omega(p))$[2].

This paper is organized as follows. The following Section recalls the two main components of our algorithm: the sequential greedy and Luby's algorithms. Section 3 presents the principle of the algorithm and the strategy for mapping the graph onto the processors, along with the $p$-quantiles search algorithm. Finally, Section 4 presents our MIS algorithm in detail. We close the paper with some concluding remarks and ways for further research.

## 2 Two basic components: the PRAM and the sequential algorithms

Our algorithm can be briefly sketched as follows. 1) data mapping, performed in a preprocessing phase; 2) execution of the first rounds of Luby's randomized PRAM MIS algorithm; 3) execution of the greedy sequential algorithm, in one processor.

In this Section, we recall Luby's and the greedy algorithms. In the next Section, we discuss the preprocessing phase.

**Greedy sequential algorithm.** The classical greedy algorithm proceeds as follows: while there subsists vertices, take an arbitrary vertex, put it in the MIS and suppress its neighbors. The resulting set of the chosen vertices is clearly a MIS of the initial graph.

**Luby's scheme.** The principle of this scheme, which is very similar to the independently found solution in [1], is to start with an empty independent set and to make it grow round after round by adding new vertices. The idea is to try to select only one node in each neighborhood so that to ensure that the set of the selected vertices is a set "as big and as independent as possible". Thus, at each round, vertices are selected at random with a probability inversely proportional to their degree (Phase I). Conflicts between adjacent selected nodes are resolved by "unselecting" the one with the lowest degree

[2] We remark that sorting can be optimally solved using other, more complex, techniques. The BSP/CGM algorithm proposed in [16] achieves optimal speedup and a constant number of communication rounds, with scalability assumption $\frac{n}{p} = \Omega(p^{\epsilon})$, for any constant $\epsilon > 0$.

(Phase II). At the end of Phase II, the nodes which are still marked as selected are added to the independent set and deleted from the graph with their neighborhood (Phase III). Once the graph is empty, the resulting independent set clearly forms a MIS of the original graph. The complete PRAM algorithm is described in Figure 1.

```
Input: G = (V, E)
MIS := ∅
while V ≠ ∅ do
 (I)  in parallel for all i ∈ V: Select node i with probability 1/(2 degree(i))
 (II) in parallel for all {i, j} ∈ E: if i and j are selected then
          if degree(i) > degree(j) then unselect j
          else unselect i
(III) I := {i ∈ V : i is selected}          MIS := MIS ∪ I
      V := V ∖ (I ∪ neighborhood(I))      E := E ∩ (V × V)
endwhile
Output: MIS
```

Fig. 1: Luby's algorithm.

Luby shows the following lemma which will be useful to prove our Theorem 6. This lemma yields the expected number of rounds of the while loop in Figure 1. Recall that $\mathbb{E}[X]$ denotes the expected value of a random variable $X$, and $|A|$ denotes the size of a set $A$.

**Lemma 1** *[19] Consider an execution of the algorithm of Figure 1 on an arbitrary graph. Let $E_k$ be the set of edges at the end of $k$-th round ($E_0 = E$). Then, for any $k$, the expected number of edges removed in step (III) satisfies:* $\mathbb{E}[\ |E_k \setminus E_{k+1}|\ ] \geqslant \frac{1}{8}|E_k|$.

## 3 Preprocessing

We assume that the input graph is described by its $n$ vertices and $m$ edges which are labeled with distinct arbitrary integers. Initially, the nodes and the edges are uniformly and arbitrarily spread among the processors: each processor stores $\frac{n}{p}$ nodes and $\frac{m}{p}$ edges. Without loss of generality and for the sake of simplicity, we will consider that $n/p$ and $m/p$ are integers. In order to satisfy the $O((n+m)/p)$ memory requirement, we need to ensure a compact routing table towards the differents processors storing the nodes. This is obtained using our $p$-quantile algorithm, described in Section 3.1: after this step each processor $P_i$ stores the $((i-1) \times p + 1)^{\text{th}}$ to the $(i \times p)^{\text{th}}$ nodes; thus each processor only needs to know the highest node stored on each of the others to route its messages, which requires only $O(p)$ memory and is achieved in one communication step.

Each step of Luby's algorithm relies on the computation of the degree of each node. The natural way to compute the degree of a node consists of summing its local degrees on each processor. Hence, the initial arbitrary distribution of the edges among the processors is not satisfactory, since each node may be the extreme point of edges stored in all processors. Thus, each processor would have to gather $p$ local degrees for each of its $\frac{n}{p}$ nodes, implying that each processor could receive $\Theta(\frac{n}{p}p)$ data units, which exceeds the local memory capacity $\Theta(\frac{n+m}{p})$.

One way to overcome this problem would be to lexicographically sort the edges by their extreme points. However, this would lead to a local computation time of $\Omega(\frac{m \log m}{p})$. Here, we propose to execute an initial remapping of the edges onto the processors in $O(\frac{m}{p} \log p)$ time, as follows. The undirected edges $\{i, j\}$ are duplicated into two directed arcs $(i, j)$ and $(j, i)$. These arcs are then remapped onto the processors such that all the arcs of a processor $P_q$ are lexicographically smaller than all arcs of any processor $P_r$, $r > q$. We call such a remapping an *increasing edges partitioning.* Such a remapping allows us to prove the following lemma.

**Lemma 2** *Let $G = (V, E)$ be an arbitrary graph, stored in a BSP/CGM with p processors, according to an increasing edges partitioning. Then, the computation of the degree of each node in $G$ can be done in linear local computation time and a single communication step.*

**Proof.** The degree computation proceeds as follows. Each processor computes the local degree of the origins of its arcs, and sends them to the processors storing those nodes. The processor can now sum the local degrees of its nodes to compute their degrees.

The lemma stems from the fact that, in an increasing edges partitioning, the arcs incident to a node $i$ are spread on at most $\lceil \frac{\text{degree}(i)}{m/p} \rceil + 1$ processors. Thus, each processor $P_q$ receives at most $\sum_{\text{Node } i \in P_q} \left( \lceil \frac{\text{degree}(i)}{m/p} \rceil + 1 \right) \leqslant \frac{\sum_{\text{Node } i \in P_q} \text{degree}(i)}{m/p} + 2\ \frac{n}{p} \leqslant 2(p + \frac{n}{p}) = O(\frac{n+m}{p})$ values. ◀

In order to obtain the increasing edges partitioning, we use a regular sampling method. Such a method relies on the parallel extraction of $p-1$ *global pivot* arcs $e_1, \ldots, e_{p-1}$ which split the arcs, according to the lexicographic order, into an increasing partition of $p$ blocks of size $O(\frac{m}{p})$ each. Once these global pivots are computed, each processor $P_q$ can gather, in one communication step, the arcs between $e_{q-1}$ and $e_q$, forming the desired increasing edges partition.

The global pivots are obtained as follows. Each processor locally extract its $p-1$ *local pivot* arcs which lexicographically split its own arcs into $p$ increasing blocks of equal size. In [22] it was shown that the final $p-1$

global pivots $e_1, \dots, e_{p-1}$ can be chosen equally spaced in the list of $p(p-1)$ local pivots.

The method described in [22] obtained the local pivots by locally sorting the arcs, while the global pivots were obtained after a sequential sort was performed on one processor. Its total computation time is then $\Theta(\frac{m}{p}\log\frac{m}{p} + p^2\log p)$ with $O(1)$ communication steps and scalability assumption $\frac{m}{p} = \Omega(p^2)$.

In the following, we show that an increasing arcs partitioning can be obtained faster in a BSP/CGM computer with scalability assumption $\frac{m}{p} = \Omega(p)$, through a solution to the $p$-quantiles search problem.

### *3.1 BSP/CGM p-quantiles search*

The $p$-quantiles of an ordered set $A$ of size $n$ are the $p-1$ elements, of rank $\frac{n}{p}$, $\frac{2n}{p}$,..., $\frac{(p-1)n}{p}$, splitting $A$ into $p$ parts of equal size. The $p$-quantiles search problem consists of determining those splitting elements of $A$ [8] . The sequential algorithm starts by finding the median of the set of $p$-quantiles of $A$, splitting $A$ into two sets, and applying this scheme recursively until $p-1$ splitters are found. Its complexity is clearly $O(n\log p)$ steps.

In the BSP/CGM setting, we prove the following result.

**Proposition 3** *The $p$-quantiles search algorithm on a set of $n$ elements can be implemented in the BSP/CGM with $p$ processors in $O(\frac{n}{p}\log p)$ time and $O(1)$ communication steps, with the scalability assumption $\frac{n}{p} = \Omega(p)$.*

**Proof.** Each processor extracts $p-1$ local pivots without sorting its data, but by using the sequential $p$-quantiles search algorithm, in time $O(\frac{n}{p}\log p)$. These $p(p-1)$ pivots have to be globally sorted in order to extract the final $p-1$ global pivots. Notice that these $p(p-1)$ pivots can be seen as laying in a mesh with $p$ columns (one per processor) and $p-1$ rows. Hence, the pivots can be sorted in time $O(p\log p)$ and $O(1)$ communication steps by a straightforward coarse grained implementation of the mesh-sort algorithm proposed in [21], which uses a constant number of operations over the rows (here, a communication round) and over the columns (here, a local computation). ◀

**Corollary 4** *Sorting a set of $n$ elements by sampling can be implemented in the BSP/CGM with $p$ processors in time $O(\frac{n\log n}{p})$ and $O(1)$ communication steps, with the scalability assumption $\frac{n}{p} = \Omega(p)$.*

**Proof.** The result follows directly from the algorithm presented in [22], where the search for the pivots is replaced by the $p$-quantiles search algorithm given above. Indeed, once the $p$-quantiles are found and known by all processors, the data can be sent to their final destination, where a final local sort suffices to complete the global sort. ◀

**Corollary 5** *An increasing edges partitioning of a graph with $n$ nodes and $m$ edges can be obtained in the BSP/CGM with $p$ processors in $O(\frac{m}{p}\ \log p)$ time and $O(1)$ communication steps, with the scalability assumption $\frac{m}{p} = \Omega(p)$.*

## 4 The BSP/CGM MIS algorithm

Our scheme works as follows (details on Figure 2, where P␣: and C␣: denote, respectively, a parallel computation step and a communication step; "e.p." is the abbreviation of "each processor"). After the edges remapping phase described in Section 3, it proceeds to a series of selection rounds. At each round, a communication step allows each processor to compute the degree of each of its nodes. After having selected nodes according to the probability rule (Phase I, P1-2, Figure 2), a second communication step provides the information required to unselect nodes in conflict (Phase II, P3, Figure 2). A final communication step provides the information required to remove the new MIS vertices and their neighborhood (Phase III, P4-6, Figure 2). The reduction of the number of communication steps is based on a method known as clipping [5] and consists of gathering the data on one single processor as soon as they can be stored in its local memory, of size $O(\frac{n+m}{p})$. This processor then finishes the task by applying the greedy sequential algorithm. We show here that this paradigm can be applied to randomized schemes as well.

### *4.1 Analysis of the algorithm*

**Number of communication rounds.** The preprocessing phase consists of partitioning the edges. Section 3 has shown that this requires $O(1)$ communications rounds. Note that the remapping of edges is deterministic.

The aim is now to evaluate the expected number of rounds, $K$, of the while loop. Our algorithm follows the original Luby scheme presented in Section 2. According to Lemma 1, we have that $\mathbb{E}[|E_k \setminus E_{k+1}|] \geqslant \frac{1}{8}|E_k|$, where $E_k$ denotes the number of edges after the $k$-th round. This implies that for any $k$, $\mathbb{E}[|E_k|] \leqslant \frac{7}{8}\mathbb{E}[|E_{k-1}|] \leqslant (\frac{7}{8})^k m$. Therefore,

$$
\begin{aligned}
\mathbb{E}[K] &= \textstyle\sum_{k=0}^{\infty} k\ \Pr\{K = k\} \\
&= \textstyle\sum_{k=0}^{\infty} \Pr\{K \geqslant k\} \\
&= \textstyle\sum_{k=0}^{\infty} \Pr\left\{|E_k| > \frac{m}{p}\right\} \\
&\leqslant \textstyle\sum_{k=0}^{\infty} \min(1, \frac{1}{m/p}\mathbb{E}[|E_k|]) \qquad \text{(Markov inequality)} \\
&\leqslant \textstyle\sum_{k=0}^{\lceil \log_{\frac{8}{7}} p\rceil} 1 + \sum_{k > \lceil \log_{\frac{8}{7}} p\rceil} \frac{p}{m}\ (\frac{7}{8})^k\ m \\
&\leqslant \lceil \log_{\frac{8}{7}} p\rceil + 9 \leqslant 5.2 \log_2 p + 9
\end{aligned}
$$

*Input*: $G = (V, E)$, $p$ processors.
*Each processor stores* $\frac{n}{p}$ *nodes and* $\frac{m}{p}$ *edges.*

P0/C0: *Nodes mapping.* The nodes are partially sorted accross the processors using the $p$-quantile search. Each processsor sends to the others its final highest node number.
*Edges mapping.* Each processor (abrev. "e.p.") duplicates its edges into two opposite arcs. These arcs are seen as couples of node identities (head,tail). After the preprocessing phase, these arcs, are partially sorted in lexicographical order among the processors, so that processor $P_q$ stores the arcs of rank $r$, with $\frac{(q-1)m}{p} < r \leqslant \frac{qm}{p}$. Each processor sends its highest head arc identities to all the other processors.

*MIS* is set to $\emptyset$.

WHILE $|E| \geqslant \frac{m}{p}$ DO

(Degree computation)

P1: e.p. computes the local input degree of the heads of its arcs.
C1: e.p. receives and sums the local degrees of each of its nodes.

(Selection)

P2: e.p. selects each of its nodes $i$ with probability $\frac{1}{2\ \mathrm{degree}(i)}$.
C2: e.p. informs the other processors whether the heads or tails of their arcs have been selected.

(Conflict resolution)

P3: e.p. considers each of its arcs $(i, j)$, $i < j$. If both nodes $i$ and $j$ are selected, then it unselects the one with the smallest degree.
C3: e.p. informs the other processors whether their nodes have been unselected.

(Edge and node cancellation)

C4: e.p. informs the other processors whether some of their arcs have an extreme in the new *MIS*.
P4: e.p. cancels the arcs that have an extreme in the MIS and stores in a list the other extreme.
C5a: e.p. informs the other processors whether some of their nodes are in the neighborhood of the new *MIS*.
C5b: e.p. informs the other processors whether some of their arcs have an extreme in the neighborhood of the new *MIS*.
P5: e.p. cancels those arcs and those nodes.

(Edge count)

P6: e.p. counts its arcs.
C6: the processors count the total number of remaining arcs.

ENDWHILE

P7: One processor gathers the remaining arcs and nodes and applies the sequential greedy scheme to complete the *MIS*.

*Output*: *each processor knows which of its nodes are in the MIS.*

Fig. 2: The BSP/CGM MIS algorithm.

The expected number of rounds of the while loop is then $O(\log p)$. Furthermore, the Markov inequality yields that $\Pr\{\text{Number of rounds} \geqslant 200 \log_2 p\} = \Pr\{|E_{200 \log_2 p}| > \frac{m}{p}\} \leqslant \frac{p}{m} \left(\frac{7}{8}\right)^{200 \log_2 p} m \leqslant \frac{1}{p^{37}}$. Hence, this algorithm requires $O(\log p)$ communication rounds, with high probability, and $O(\frac{n+m}{p})$ time per round.

**Analysis of the communications and memory requirements.** We now have to check the BSP/CGM requirements on the size of the all-to-all communications and local memories. All the communication phases of the algorithm consist of exchanges of data between, on the one hand, a processor which stores a node, and on the other hand, the processors which store its incident arcs, and vice-versa.

Consider a transmission "from arcs to their heads" (Steps C1, C3 and C5a). Each processor transmits at most $O(\frac{m}{p})$ data. Thanks to the preprocessing phase, the arcs incident to a node $i$ are spread on at most $\left\lceil \frac{\text{degree}(i)}{m/p} \right\rceil + 1$ consecutive processors. Thus, an analysis similar to the one in Lemma 2 yields that the total amount of data received by each processor is at most $O(\frac{n+m}{p})$.

Conversely, during a transmission "from nodes to their incident arcs" (Step C2, C4 and C5b), each processor receives at most $O(\frac{m}{p})$ data and transmits at most $O(\frac{n}{p})$ data.

Step C6 also verifies the memory requirements, since $p = O(\frac{n+m}{p})$.

**Computation time analysis.** Since each round requires $O(\frac{n+m}{p})$ local computation time and the preprocessing phase runs in $O(\frac{m}{p} \log p)$ time, then, the computation time is $O(\frac{n+m}{p} \log p)$, with high probability.

We have thus proved the following result.

**Theorem 6** *The BSP/CGM algorithm given in Figure 2 computes a MIS on an arbitrary graph, with $n$ vertices and $m$ edges, in $O(\frac{n+m}{p} \log p)$ time and $O(\log p)$ communication rounds, with high probability, with scalability assumption $\frac{n+m}{p} = \Omega(p)$.*

Note that the number of communication rounds is independent of $n$ and $m$.

## 5 Conclusion

We presented a scalable randomized BSP/CGM MIS algorithm, which is well adapted for existing parallel computers. Its required number of communication rounds is independent of the input size, $n$ and $m$, and depends only on the size, $p$, of the parallel system, whenever $\frac{n+m}{p} = \Omega(p)$.

We also presented an optimal deterministic parallel implementation of the $p$-quantiles search, running in a constant number of communication rounds. This algorithm may find extensive applications. Here we used it in a preprocessing phase in order to partition the edges of the graph so that to allow for quick degree computations. Its solution could also be used to improve the scalability assumptions of sorting by sampling.

Further work in BSP/CGM MIS algorithms would consist of reducing the number of communication rounds to a constant, in order to obtain an optimal speedup. Another valuable challenge is the design of a deterministic BSP/CGM algorithm for the MIS problem matching the bounds established in this paper. We remark that Luby's technique to obtain a deterministic PRAM algorithm from its randomized counterpart does not apply here, since it consisted of simulating in parallel all the possible random scenarios, at the cost of multiplying the number of processors by a factor $\Theta(n^2)$ [19]. The same cannot be tried in the BSP/CGM model, where the number of processors is fixed.

**Acknowledgments**

We are grateful to Claire Kenyon for useful suggestions and careful reading of the manuscript.

1. N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. of Algorithms*, 7:567–583, 1986.
2. R.J. Anderson and L. Snyder. A comparison of shared and nonshared memory models of computation. *Proceedings of the IEEE*, 79(4):480–487, 1997.
3. B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Fast distributed network decompositions and covers. In *JPDC*, volume 39, pages 105–114, Dec 1996.
4. G. E. Blelloch, C. E. Leiserson, B. M. Maggs, and C. G. Plaxton. A comparison of sorting algorithms for the Connection Machine CM-2. In *Proc. ACM Symp. on Parallel Algorithms and Architectures*, pages 3–16, 1991.
5. E. Caceres, F. Dehne, A. Ferreira, P. Flocchini, I. Rieping, A. Roncato, N. Santoro, and S. Song. Efficient parallel graph algorithms for coarse grained multicomputers and BSP. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccam ela, editors, *Proc. of ICALP'97*, volume 1256 of *LNCS*, pages 390–400. Springer-Verlag, 1997.
6. B. Chamaret, S. Josselin, D. Wagner, and S. Ubéda. Adaptative BTS location in cellular network. In *Parallel and Distributed Computing and Networks*, pages 287–291, 1997.
7. S. A. Cook. An overview of computational complexity. *Comm ACM*, 26(6):400–408, 1983.
8. T. Cormen, C. Leiserson, and R. Rivest. *Introduction to algorithms.* MIT Press, 1990.
9. F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel geometric algorithms for coarse grained multicomputers. In *9th ACM Symp. on Computational Geometry*, pages 298–307, 1993.
10. M. Diallo, A. Ferreira, and A. Rau-Chaplin. Communication-efficient deterministic parallel algorithms for planar point location and 2D Voronoï diagram.

In *15th Symp. on Theoretical Aspects of Computer Science (STACS)*, LNCS. Springer Verlag, February 1998.

11. M. Diallo, A. Ferreira, A. Rau-Chaplin, and S. Ubéda. Scalable 2D convex hull and triangulation algorithms for coarse-grained multicomputers. *Journal of Parallel and Distributed Computing*, 56(1):47–70, January 1999.
12. A. Ferreira, I. Guérin-Lassous, K. Marcus, and A. Rau-Chaplin. Parallel computation of interval graphs on PC clusters: Algorithms and experiments. In D. Pritchar and J. reeves, editors, *Proceedings of Europar'98*, volume 1470 of *Lecture Notes in Computer Science*, pages 875–886. Springer Verlag, 1998.
13. A. V. Gerbessiotis and C. J. Siniolakis. Primitive operations on the BPS model. Technical Report PRG-TR-23-96, Oxford University, 1996.
14. A. V. Goldberg and S. A. Plotkin. Parallel $(\Delta+1)$-coloring of constant-degree graphs. *Information Processing Letters*, 25(4):241–245, Jun 1987.
15. A. V. Goldberg, S. A. Plotkin, and G. E. Shannon. Parallel symmetry-breaking in sparse graphs. In *19th STOC*, volume 19, pages 315–324, May 1987.
16. M. Goodrich. Communication-efficient parallel sorting. In $28^{th}$ *STOC*, pages 247–255, May 1996.
17. R. M. Karp and A. Widgerson. A fast parallel algorithm for the maximal independent set problem. *JACM*, 32(4):762–773, Oct 1985.
18. N. Linial. Distributive graph algorithms – global solutions from local data. In *28th FOCS*, volume 28, pages 331–335, Oct 1987.
19. M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, Nov 1986.
20. N. A. Lynch. *Distributed Algorithms.* M. K. Publishers, 1996.
21. J. M. Marberg and E. Gafni. Sorting in constant number of row and column phases on a mesh. *Algorithmica*, 3:561–572, 1988.
22. H. Shi and J. Schaeffer. Parallel sorting by regular sampling. *J. of Parallel and Distributed Computing*, 14:361–372, 1992.
23. L. Valiant. A bridging model for parallel computation. *Communication of ACM*, 38(8):103–111, 1990.