Bull. Nov. Comp. Center, Comp. Science, 9 (1998), 1–14

Distributed architecture for implementing fast parallel algorithms for two combinatorial optimization problems

S.M. Achasova

Using the Parallel Substitution Algorithm, as a formal model of parallel computations, a distributed architecture is designed for implementation of the following two fast parallel algorithms: one for the maximal independent set problem and the other for the minimum weighted vertex cover problem. The former is a randomizing algorithm based on the Monte Carlo method, the latter is a deterministic approximation algorithm based on the primal-dual technique that consists of finding feasible solutions to the problem under consideration and its dual to be close to each other. The topology of the connections between the processors in the designed distributed architecture is similar to the topology of the graph in hand.

Introduction

It is known that sequential algorithms for combinatorial optimization problems cannot always be converted in fast parallel algorithms. One frequently cited instance of such a problem is the maximal independent set (MIS) problem. Given an n-vertex, m-edge undirected graph $G = (V, E)$, find a subset of vertices $I \subseteq V$ having the following properties: 1) I is a set of pairwise independent (i. e., not adjacent) vertices, 2) on addition a new vertex to I, the property 1 does not hold. Such a subset I is a MIS. The obvious sequential algorithm for the MIS problem can be stated as: initialize I to the empty set, for $i = 1, \ldots, n$ if a vertex i is not adjacent to any vertex in I then add i to I. This algorithm cannot be converted in a fast parallel algorithm that requires a number of iterations being expressed by a function of the input size of a problem that grows considerably slower than the linear one [1–3].

This circumstance stimulates a search for completely different approaches to designing parallel algorithms than sequential ones. We refer to two such approaches. The first makes use of random variables and the Monte Carlo method [1, 2], and it is illustrated by a Monte Carlo algorithm for the MIS problem. The second makes use of the primal–dual technique based on the fact that the optimum value of an optimization problem is equal to the optimum value of its dual [4–6]. The second approach is applied to

the minimum weighted vertex cover problem. In the case, given a graph $G = (V, E)$ with vertex weights $w(i)$ $(i \in V)$, find $C \subseteq V$ such that: 1) for each edge $(i, j) \in E$ $(i, j \in V)$ at least one of i and j belongs to C, and 2) $\sum_{i \in C} w(i)$ is minimum.

The Monte Carlo algorithm finds a MIS taking $O(\log n)$ iteration steps [2]. In each iteration, the algorithm randomly selects a subset of vertices which are pretenders to a MIS (the operation is executed for all vertices simultaneously). Further, if the selected subset contains pairs of vertices connected by an edge, then one vertex of each pair is removed from the subset (the operation is executed for all edges simultaneously). The remaining vertices are added to the MIS and removed from the graph along with all adjacent vertices. For the remainder of the graph, the new iteration step is executed and so until the graph is empty.

For the minimum weighted vertex cover problem the dual problem is the maximum edge packing one, which consists of finding the edge weights $w(i, j)$ such that the total weight assigned to the edges incident to any vertex i is at most $w(i)$ and the total weight assigned to all edges is maximum. Both the primal and dual problems are NP–hard [7]. A parallel approximation algorithm based on the primal–dual technique finds a vertex cover of weight at most $2/(1-\varepsilon)$ times the minimum taking $O(\log m)$ iteration steps, where $\varepsilon \in (0, 1)$ is the key parameter for resolution whether a vertex is added to a cover [5]. In each iteration, the edge weights, simultaneously all, increase (the edge packing is formed) at the expense of lowering the weights of the incident vertices. Whenever a vertex i reaches a weight smaller than $\varepsilon w(i)$ the algorithm puts i into the vertex cover and deletes all edges incident to i. For the remainder of the graph, the new iteration step is executed and so until the graph is empty.

For the fast parallel algorithms based on the above approaches, the question arises of mapping these into a distributed architecture. The key features of distributed architectures are massive parallelism and local connections between processors. The algorithms of interest are ideally suited to these features in structure. Indeed, the algorithms assume massive parallel computations (a parallel processing of all vertices or all edges of a graph), and these are satisfied with local connections in an architecture having the topology similar of the graph in hand (a processing of a vertex requires the information only from the adjacent vertices or from the incident edges; a processing of an edge requires the information only from the incident vertices).

In this paper, design of distributed architectures is based on an original model of parallel computations named Parallel Substitution Algorithm (PSA) [8]. The PSA enables us to represent an algorithm as a set of copies of the basic procedures distributed in space and executed simultaneously in time. The PSA offers the following flexible mechanisms for control of parallel computations in time and in space: (a) applicability of the parallel

substitutions by condition for control of processing data in time, (b) the context in the parallel substitutions for controlling cause-and-effect relationship between operations over data distributed in space, and for controlling the order of processing data in time, (c) the naming functions in the parallel substitutions for organizing parallel operations over data distributed in space. For designing and debugging parallel substitution algorithms a simulating system named Animated Language Tools (ALT) has been developed which has tools for visual and textual representation of computational processes in a distributed structure [8–10].

In the remainder of the paper, the basic ideas of the PSA are given, and the parallel substitution algorithms for the above combinatorial optimization problems are presented.

1. The parallel substitution algorithm

The PSA is a formal model of distributed computations that is intended for organizing the joint work of a massive number of simple processors with the aim to solve a given problem.

The processors take names. A set of names M corresponds to the structure of the input data of the problem being solved. Traditionally, M is either the set of coordinates of the 2D or 3D Cartesian space, if the data are located at the nodes of an integer Cartesian grid, or a set of symbols for the graph problems, if the topology of the connections between the processors is assumed to be similar to the topology of the graph in hand.

A processor is assumed to have a set of states A. The sets M and A are finite. A pair (a, m), where $a \in A$ and $m \in M$, is called a cell. A finite set of cells with no pair of cells having one and the same name is termed a cellular array K.

An elementary operation over a cellular array is a substitution

$$S_1 * S_2 \Rightarrow S_3,$$

where S_1, S_2 and S_3 are sets of cells, $S_1 = \{(a_1, m_1) \dots (a_p, m_p)\}$ is the base of a substitution, $S_2 = \{(b_1, m_{p+1}) \dots (b_q, m_{p+q})\}$ is the context of a substitution (all names $m_1, \dots, m_{p+q}$ are different), $S_3 = \{(c_1, m_1) \dots (c_p, m_p)\}$ is the right-hand part of a substitution. The base and the right-hand part of a substitution are of the same cardinality and contain cells with the same set of names. Note, that $c_1, \dots, c_p$ can be both merely states and functions of states $a_1, \dots, a_p$, $b_1, \dots, b_q$. In the latter case, the substitutions are named functional.

An elementary substitution $S_1 * S_2 \Rightarrow S_3$ is applicable to a cellular array K if $S_1 \cup S_2 \subseteq K$. The substitution is executed by substituting the cells of the right-hand part S_3 for the cells of the base S_1. The cells of the context do not change. The context is the condition for a substitution to be applicable.

For representation of a number of copies of the same operation which can be executed in parallel in a cellular array a substitution is generalized to a parallel substitution

$$\theta : S_1(m) * S_2(m) \Rightarrow S_3(m),$$

in which

$$S_1(m) = \{(a_1, \varphi_1(m)) \dots (a_p, \varphi_p(m))\},$$
$$S_2(m) = \{(b_1, \psi_1(m)) \dots (b_q, \psi_q(m))\},$$
$$S_3(m) = \{(c_1, \varphi_1(m)) \dots (c_p, \varphi_p(m))\},$$

where $\varphi_i(m)$ $(i = 1, \dots, p)$ and $\psi_j(m)$ $(j = 1, \dots, q)$ are functions over a set of names M with both the domain and the range equaled M. The functions are called naming. The values of naming functions for any $m \in M$ must be different. The naming functions define the neighborhood of the processors.

Parallel computations in a cellular array K are determined by a set of parallel substitutions $\Phi = \{\theta_1, \dots, \theta_k\}$, which are applied to K in accordance with the following iterative procedure.

Let us assume that $K(t)$ is a cellular array being the result of execution of a set of the parallel substitutions Φ in K during t iteration steps. Further,

- if no substitution $\theta_j \in \Phi$ is applicable to $K(t)$, then $K(t)$ is the result of the computation, else
- all applicable to $K(t)$ substitutions are executed simultaneously, and $K(t)$ is transformed to $K(t+1)$ that is the result of the $(t+1)$-th iteration step.

Determinacy of a parallel computation performed in accordance with the above synchronous procedure is provided with the property of non-contradictoriness of a set of parallel substitutions Φ. This property lies in the fact that the application of Φ to any K (specified over the given sets A and M) cannot give a set of cells in which there are if only two cells with the same names and different states [8].

The non-contradictory set of parallel substitutions Φ together with the above iterative procedure of its execution is called a parallel substitution algorithm.

2. A parallel substitution algorithm for the maximal independent set problem

The Monte Carlo algorithm for the maximal independent set problem [2] contains two basic procedures VER and ED which are performed in each iteration step, first VER and then ED. VER is applied to all vertices of a

graph in parallel, ED – to all edges of a graph in parallel. VER constructs a set X of vertices which are pretenders to a MIS I. A vertex i is put into X with probability $1/2d(i)$, where $d(i)$ is the degree of i. ED works with each pair of vertices belonging to X and connected by an edge and puts one of two edges of such a pair into I, namely that has the greater degree than the other. In [2] it is proved that the algorithm can be performed in $O(\log n)$ iteration steps.

The above Monte Carlo algorithm generates n pairwise independent random variables [11] in each iteration by the following rule:

$$r(i) = (x + y \times i) \bmod q, \qquad i = 1, \ldots, n,$$

where x, y are random numbers from the interval $[0, q-1]$, q is a prime number from the interval $[n, 2n]$, it defines the cardinality q^2 of a probability space each point $\{(x, y)\}$ of which corresponds to one of variants of X. Let $G' = (V', E')$ be a subgraph of

$$G = (V, E), \qquad N(W) = \{i \in V' : \ \exists j \in W, (i, j) \in E'\}$$

be the neighborhood of $W \subseteq V'$, $d'(i)$ be the current degree of a vertex i. Here is a pseudocode of the Monte Carlo algorithm (q is assumed to be given).

The algorithm MC

```
begin G'(V', E') := G(V, E); I := ∅;
   while G' ≠ ∅, do
      begin
         in parallel for all i ∈ V'
            compute d'(i);
            if d'(i) = 0, then add i to I and remove from V';
         randomly select x and y from the interval [0, q − 1];
         X := ∅;
         in parallel for all i ∈ V' do
            p(i) := q/2d'(i);
            r(i) := (x + y × i) mod q;
            if r(i) ≤ p(i), then add i to X;
         in parallel for all (i, j) ∈ E'
            if i ∈ X and j ∈ X, then
               if d'(i) ≤ d'(j), then remove i from X,
               else remove j from X;
         I := I ∪ X;
         Y := X ∪ N(X);
         V' := V' \ Y;
      end
end
```

The algorithm MC can be implemented in a distributed architecture of the following kind.

Let there be n nodes. The topology of connections of the nodes is similar to the graph in hand. A node has a name i as its respective vertex. Each node i is connected with the nodes $j_1^{(i)}, \ldots, j_k^{(i)}$, where $k = d(i)$, as their respective vertices are connected with a vertex i. Each node i combines six elementary processors which have the names $\langle i, 1\rangle, \ldots, \langle i, 6\rangle$. In what follows, we say "cells" instead of "processors" and for brevity, we say "a cell m" instead of "a cell having the name m". Two cells $\langle i, 5\rangle$ and $\langle i, 6\rangle$ are merely the memory ones for the constants i and q.

In addition to the above n nodes there is one more node with the name *ran* which is connected with each of the n nodes. The node *ran* combines three cells $\langle \text{ran}, 1\rangle$, $\langle \text{ran}, 2\rangle$ and $\langle \text{ran}, 3\rangle$, two of which generate random numbers x and y, and the third cell works as a counter. A fragment of the architecture is shown in Figure 1.

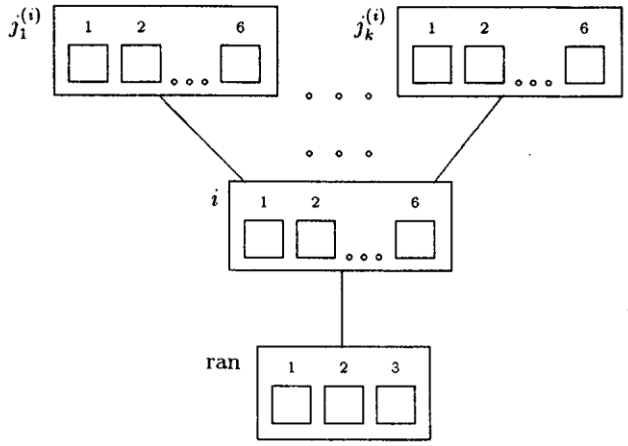

Figure 1

The cell $\langle i, 2\rangle$ computes the current degree $d'(i)$ of a vertex i. The cell $\langle i, 1\rangle$ calculates the random number $r(i) = 2d'(i)((x + iy) \bmod q)$ for its node. The cells $\langle i, 3\rangle$ and $\langle i, 4\rangle$ carry the following information: $\langle i, 3\rangle$ has the state 1 if the vertex i still remains in G', the state 0 if the vertex i has been removed from G'; $\langle i, 4\rangle$ has the state 1 if the vertex i is put into X, the state 2 if the vertex i has been put into I, the state 0 if the vertex i is not put into both X and I.

A parallel substitution algorithm to operate the described set of the processors contains functional substitutions, and so the set of the states of the cells contains variable symbols $\{x, y, z_1, z_2, \ldots\}$ and functional symbols $\{f_1, f_2, \ldots\}$ and also the "don't care" symbol " $-$ ". Both the domain and the range of the functions and also of the variables are equal to the interval of integers $[0, \ldots, 2\Delta(q-1)]$, where Δ is the maximal degree of the vertices of the graph.

A parallel substitution algorithm realizing the MC in the distributed architecture comprises six substitutions $\{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6\}$ (their visual representation is shown on Figure 2):

$$\theta_1: \{(-, \langle \text{ran}, 1\rangle)(-, \langle \text{ran}, 2\rangle)(0, \langle \text{ran}, 3\rangle)\}$$
$$\Rightarrow \{(f_1', \langle \text{ran}, 1\rangle)(f_1'', \langle \text{ran}, 2\rangle)(1, \langle \text{ran}, 3\rangle)\},$$
$$f_1' = \langle \text{a random number } 0 \le x \le q-1 \text{ is generated}\rangle,$$
$$f_1'' = \langle \text{a random number } 0 \le y \le q-1 \text{ is generated}\rangle;$$

$$\theta_2: \{(-, \langle i, 2\rangle)\} * \{(1, \langle i, 3\rangle)(z_{j_1}, \langle j_1^{(i)}, 3\rangle) \ldots (z_{j_k}, \langle j_k^{(i)}, 3\rangle)(0, \langle \text{ran}, 3\rangle)\}$$
$$\Rightarrow \{(f_2, \langle i, 2\rangle)\},$$
$$f_2 = z_{j_1} + \ldots + z_{j_k};$$

$$\theta_3: \{(-, \langle i, 1\rangle)(1, \langle \text{ran}, 3\rangle)\} *$$
$$\{(z_2, \langle i, 2\rangle)(1, \langle i, 3\rangle)(x, \langle \text{ran}, 1\rangle)(y, \langle \text{ran}, 2\rangle)(i, \langle i, 5\rangle)(q, \langle i, 6\rangle)\}$$
$$\Rightarrow \{(f_3, \langle i, 1\rangle)(2, \langle \text{ran}, 3\rangle)\},$$
$$f_3 = 2z_2((x + iy) \bmod q);$$

$$\theta_4: \{(0, \langle i, 4\rangle)(2, \langle \text{ran}, 3\rangle)\} * \{(z_1, \langle i, 1\rangle)(1, \langle i, 3\rangle)(q, \langle i, 6\rangle)\}$$
$$\Rightarrow \{(f_4, \langle i, 4\rangle)(3, \langle \text{ran}, 3\rangle)\},$$
$$f_4 = 1, \text{ if } z_1 \le q;$$

$$\theta_5: \{(1, \langle i, 4\rangle)(3, \langle \text{ran}, 3\rangle)\} * \{(z_2, \langle i, 2\rangle)(z_2^l, \langle j_l^{(i)}, 2\rangle)(1, \langle j_l^{(i)}, 4\rangle)\}$$
$$\Rightarrow \{(f_5, \langle i, 4\rangle)(4, \langle \text{ran}, 3\rangle)\}, \quad l = 1, \ldots, k,$$
$$f_5 = 0, \text{ if } z_2 \le z_2^l;$$

$$\theta_6: \{(1, \langle i, 4\rangle)(1, \langle i, 3\rangle)(-, \langle j_1^{(i)}, 3\rangle) \ldots (-, \langle j_k^{(i)}, 3\rangle)(4, \langle \text{ran}, 3\rangle)\}$$
$$\Rightarrow \{(2, \langle i, 4\rangle)(0, \langle i, 3\rangle)(0, \langle j_1^{(i)}, 3\rangle) \ldots (0, \langle j_k^{(i)}, 3\rangle)(0, \langle \text{ran}, 3\rangle)\}.$$

The initial states of the cells $\langle i, 1\rangle$, $\langle i, 2\rangle$, $\langle i, 4\rangle$ of the node i, $i = 1, \ldots, n$, and of the cell $\langle \text{ran}, 3\rangle$ are equal to 0. The initial state of the cell $\langle i, 3\rangle$, $i = 1, \ldots, n$, is equal to 1, what corresponds to $G' := G$. The initial states of the cells $\langle \text{ran}, 1\rangle$ and $\langle \text{ran}, 2\rangle$ are specified by a method of generating random numbers. Its algorithm terminates work when all cells $\langle i, 3\rangle$, $i = 1, \ldots, n$, turn into the state 0. The algorithm returns a MIS specified with those cells $\langle i, 4\rangle$, $i = 1, \ldots, n$, which have the state 2.

In each iteration step, firstly the substitution θ_1 and the substitution θ_2 in all nodes i, $i = 1, \ldots, n$, are performed in parallel. The substitution θ_1 generates two random numbers. The substitution θ_2 computes the current degrees of the vertices of the subgraph G'. Then, in each node i, $i = 1, \ldots, n$, the substitution θ_3 calculates random numbers by the above rule. Further, the substitution θ_4 being executed in all nodes i in parallel builds a set X of vertices – pretenders to I. Following, for each pair of the adjacent

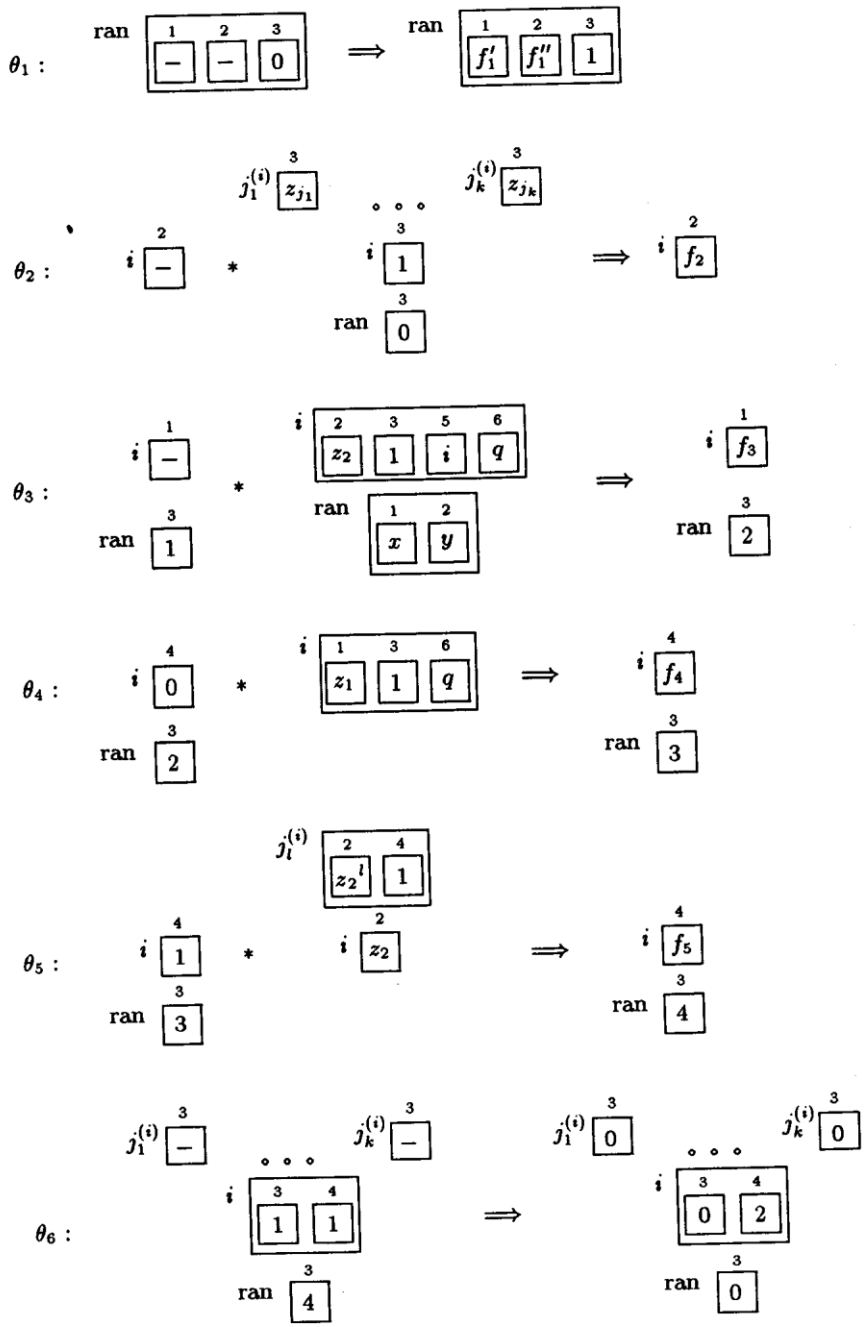
θ1 :
ran
1 2 3
– – 0
⟹
ran
1 2 3
f1' f1'' 1
θ2 :
j1(i) 3 z_j1
jk(i) 3 z_jk
i 2 –
*
i 3 1
ran 3 0
⟹
i 2 f2
θ3 :
i 1 –
*
i 2 3 5 6 z2 1 i q
ran 1 2 x y
ran 3 1
⟹
i 1 f3
ran 3 2
θ4 :
i 4 0
*
i 1 3 6 z1 1 q
ran 3 2
⟹
i 4 f4
ran 3 3
θ5 :
j1(i) 2 4 z2' 1
i 4 1
*
i 2 z2
ran 3 3
⟹
i 4 f5
ran 3 4
θ6 :
j1(i) 3 –
jk(i) 3 –
i 3 4 1 1
ran 3 4
⟹
j1(i) 3 0
jk(i) 3 0
i 3 4 0 2
ran 3 0

Figure 2

nodes i and j such that the vertices $i, j \in X$, the substitution θ_5 is executed which adds one of the vertices i and j to I. Finally, the substitution θ_6 being executed in all nodes i, $i = 1, \ldots, n$, in parallel deletes all vertices which have been put in I and all adjacent to them. The order in which the substitutions are executed is determined by the state of the cell $\langle \mathrm{ran}, 3 \rangle$ and the contexts of the substitutions. The substitutions are non-contradictory. The algorithm comes to a halt (no substitution is applicable) when the states of the cells $\langle i, 3 \rangle$, $i = 1, \ldots, n$, are equal to 0 and the state of the cell $\langle \mathrm{ran}, 3 \rangle$ is equal to 1.

3. A parallel substitution algorithm for the minimum weighted vertex cover problem

For any minimum vertex cover and any maximum edge packing

$$\sum_{i \in C} w(i) \geq \sum_{(i,j) \in E} w(i,j),$$

where $C \subseteq V$ is a vertex cover, $w(i)$ is the weight of a vertex i, $w(i,j)$ is the weight of an edge (i,j). Assuming that for any vertex $i \in C$ and a set $\mathrm{Inc}(i)$ of edges incident to the vertex i

$$\sum_{(i,j) \in \mathrm{Inc}(i)} w(i,j) \geq (1-\varepsilon) w(i),$$

where $\varepsilon \in (0,1)$, we obtain

$$(1-\varepsilon) \sum_{i \in C} w(i) \leq \sum_{i \in C} \sum_{(i,j) \in \mathrm{Inc}(i)} w(i,j) \leq 2 \sum_{(i,j) \in \mathrm{Inc}(i)} w(i,j).$$

From this follows that approximate solutions of the above dual problems differ by the factor $2/(1-\varepsilon)$ from the optimum solutions. Thus, the problem on the vertex cover reduces to the problem on the edge packing. In each iteration, the algorithm PD (Primal–Dual) starting from the zero packing increases the edge weights at the expense of lowering the weights of the incident vertices until (after a number of iterations) has formed a cover C such that $w'(i) \leq \varepsilon w(i)$ for each vertex $i \in C$. To ensure that the condition "a sum of the weights of the edges incident to a vertex i does not exceed $w(i)$" is met, in each iteration, an edge weight $w(i,j)$ may be increased only by the value

$$\delta(i,j) = \min\left\{ \frac{w'(i)}{d'(i)}, \frac{w'(j)}{d'(j)} \right\},$$

where $d'(i)$ is the current degree of a vertex i in $G' \subseteq G$, $w'(i)$ is the current weight of a vertex i. In [6] it is proved that the algorithm takes $O(\log m)$ iteration steps to return a vertex cover of weight at most $2/(1-\varepsilon)$ times the minimum.

Algorithm PD

```
begin
   G'(V', E') := G(V, E);
   C := ∅;
   in parallel for all i ∈ V'
      w'(i) := w(i);
   while G' ≠ ∅, do
      begin
         in parallel for all i ∈ V'
            compute d'(i);
         in parallel for all (i, j) ∈ E'
            δ(i, j) = min{w'(i)/d'(i), w'(j)/d'(j)};
         in parallel for all i ∈ V'
            w'(i) := w'(i) − Σ_{(i,j)∈Inc(i)} δ(i, j);
            if w'(i) ≤ εw(i), then
               begin
                  C := C ∪ i;
                  V' := V' \ i;
                  E' := E' \ {(i, j) ∈ Inc(i)};
               end
      end
end
```

The algorithm PD can be implemented in a distributed architecture of the following kind. Let there be $m+n$ nodes, n of which corresponds to the vertices, m – to the edges of the graph in hand. The architecture topology is similar to the graph. A node corresponding to a vertex i has a name i, a node corresponding to an edge (i, j) has a name $[i, j]$. Each node i is

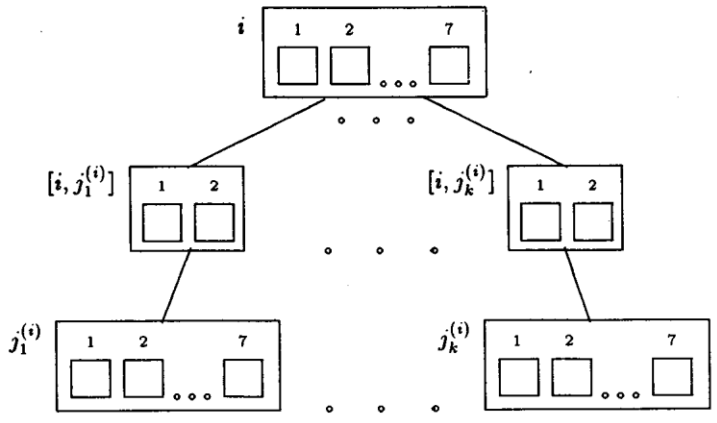

Figure 3

connected with the nodes $[i, j_1^{(i)}], \ldots, [i, j_k^{(i)}]$, $k = d(i)$. A node i contains seven cells $\langle i, 1\rangle, \ldots, \langle i, 7\rangle$, two of which $\langle i, 5\rangle$ and $\langle i, 6\rangle$ are memory cells for the constants $w(i)$ and ε, $\langle i, 7\rangle$ works as a counter. A node $[i, j]$ contains two cells $\langle [i, j], 1\rangle$, $\langle [i, j], 2\rangle$. A fragment of the architecture is shown in Figure 3.

The cells $\langle i, 1\rangle$ and $\langle i, 2\rangle$ calculate the current weight $w'(i)$ and degree $d'(i)$ of a vertex i, respectively. The cells $\langle i, 3\rangle$ and $\langle i, 4\rangle$ can be in the state 1 or 0. The state 1 of the cell $\langle i, 3\rangle$ means that a vertex i belongs to G', the state 0 – it does not. The state 1 of the cell $\langle i, 4\rangle$ means that a vertex i has been put in the cover C, the state 0 – it has not been. A cell $\langle [i, j], 1\rangle$ calculates $\delta(i, j)$. A cell $\langle [i, j], 2\rangle$ being in the state 1 indicates that an edge (i, j) belongs to G', in the state 0 – the converse. A parallel substitution algorithm realizing PD in the distributed architecture comprises five substitutions $\theta_1, \ldots, \theta_5$ (their visual representation is shown on Figure 4):

$$\theta_1: \{(-, \langle i, 2\rangle)(0, \langle i, 7\rangle)\} *$$
$$\{(1, \langle i, 3\rangle)(z_{j_1}, \langle [i, j_1^{(i)}], 2\rangle) \ldots (z_{j_k}, \langle [i, j_k^{(i)}], 2\rangle)\}$$
$$\Rightarrow \{(f_1, \langle i, 2\rangle)(1, \langle i, 7\rangle)\},$$
$$f_1 = z_{j_1} + \ldots + z_{j_k};$$

$$\theta_2: \{(-, \langle [i, j], 1\rangle)(1, \langle i, 7\rangle)(1, \langle j, 7\rangle)\} *$$
$$\{(z_{i1}, \langle i, 1\rangle)(z_{i2}, \langle i, 2\rangle)(z_{j1}, \langle j, 1\rangle)(z_{j2}, \langle i, 2\rangle)(1, \langle i, 3\rangle)(1, \langle j, 3\rangle)\}$$
$$\Rightarrow \{(f_2, \langle [i, j], 1\rangle)\,(2, \langle i, 7\rangle)\,(2, \langle j, 7\rangle)\},$$
$$f_2 = \min\{z_{i1}/z_{i2}, z_{j1}/z_{j2}\};$$

$$\theta_3: \{(z_1, \langle i, 1\rangle)(2, \langle i, 7\rangle)\} *$$
$$\{(1, \langle i, 3\rangle)(z_{j_1}, \langle [i, j_1^{(i)}], 1\rangle) \ldots (z_{j_k}, \langle [i, j_k^{(i)}], 1\rangle)\}$$
$$\Rightarrow \{(f_3, \langle i, 1\rangle)(3, \langle i, 7\rangle)\},$$
$$f_3 = z_1 - [z_{j_1} + \ldots + z_{j_k}];$$

$$\theta_4: \{(0, \langle i, 4\rangle)(3, \langle i, 7\rangle)\} *$$
$$\{(z_1, \langle i, 1\rangle)(1, \langle i, 3\rangle)(z_5, \langle i, 5\rangle)(z_6, \langle i, 6\rangle)\}$$
$$\Rightarrow \{(f_4, \langle i, 4\rangle)(4, \langle i, 7\rangle)\},$$
$$f_4 = 1, \text{ if } z_1 \leq z_5 z_6;$$

$$\theta_5: \{(1, \langle i, 3\rangle)(4, \langle i, 7\rangle)(-, \langle [i, j_1^{(i)}], 2\rangle) \ldots (-, \langle [i, j_k^{(i)}], 2\rangle)\} *$$
$$\{(1, \langle i, 4\rangle)\}$$
$$\Rightarrow \{(0, \langle i, 3\rangle)(0, \langle i, 7\rangle)(0, \langle [i, j_1^{(i)}], 2\rangle) \ldots (0, \langle [i, j_k^{(i)}], 2\rangle)\}.$$

In each node i, $i = 1, \ldots, n$, the initial state of the cell $\langle i, 1\rangle$ is equal to $w(i)$, the initial state of the cell $\langle i, 3\rangle$ is equal to 1 (what corresponds to $V' := V$), and the initial states of the cells $\langle i, 2\rangle$, $\langle i, 4\rangle$, $\langle i, 7\rangle$ are equal to 0. In each node $[i, j]$, the cell $\langle [i, j], 1\rangle$ has the initial state 0, the cell $\langle [i, j], 2\rangle$ has

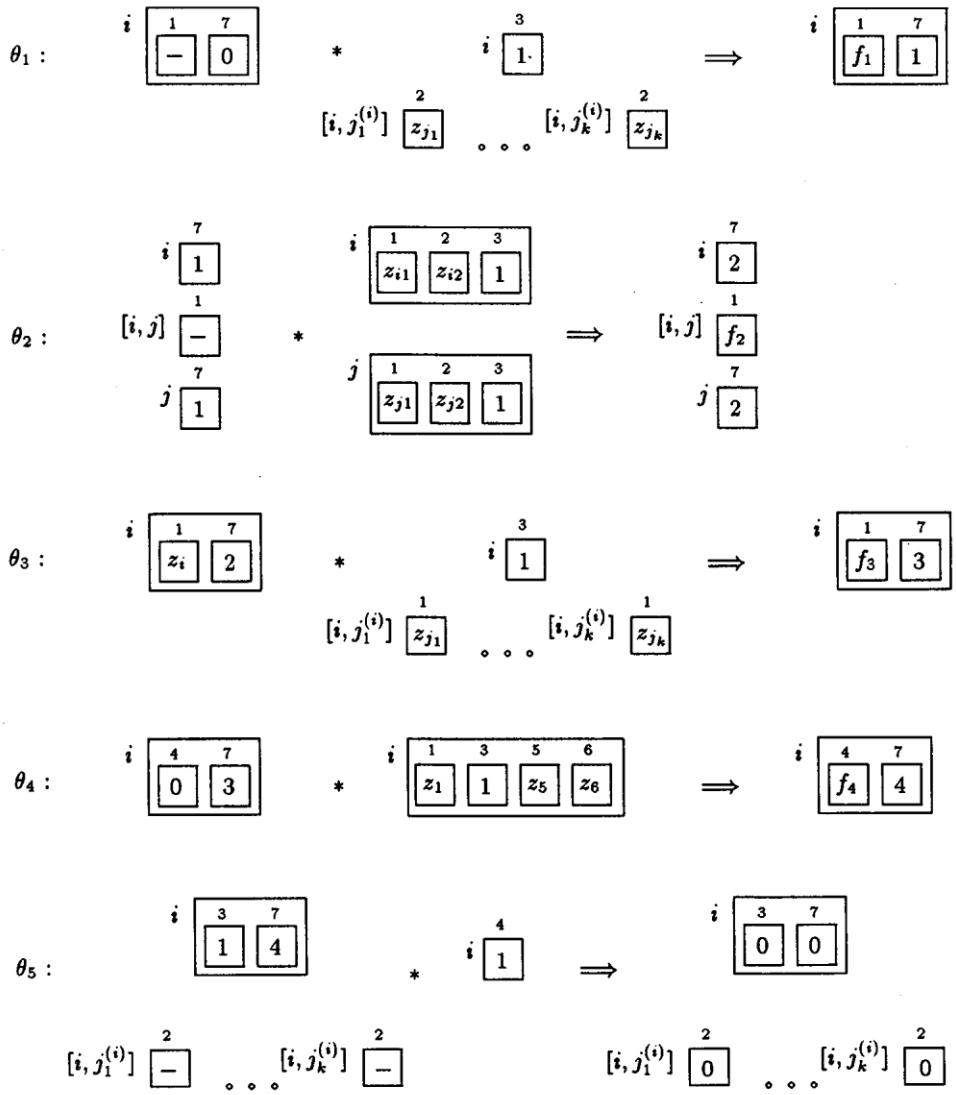

Figure 4

the initial state 1 (what corresponds to $E' := E$). The algorithm terminates its work when all cells $\langle i, 3\rangle$, $i = 1, \ldots, n$, obtain the state 0. The algorithm returns a vertex cover specified with those cells $\langle i, 4\rangle$, $i = 1, \ldots, n$, which have the state 1.

In each iteration, starting with θ_1 all five substitutions are performed one after another. Such an order corresponds to the change of state of the cell $\langle i, 7\rangle$ from 0 to 4 and further on the cycle $0, 1, \ldots, 4$. The substitution θ_1 being executed in each node i, $i = 1, \ldots, n$, and its adjacent nodes $[i, j_l^{(i)}]$, $l = 1, \ldots, k$ $(k = d(i))$, computes the current degree $d'(i)$ of a vertex i. The

substitution θ_2 is applied to each node $[i, j]$ and its adjacent ones i and j and finds the suitable addition $\delta(i, j)$ to the current weight of an edge $(i, j) \in E'$. θ_3 works with each node i and its adjacent nodes $[i, j_l^{(i)}]$, $l = 1, \ldots, k$, and decreases the weight of the vertex i by the value of the addition calculated by θ_2. θ_4 is applied only to the nodes i, $i = 1, \ldots, n$, and reveals those vertices which may be put into C. Finally, θ_5 being applied to each node i and its adjacent ones $[i, j_l^{(i)}]$, $l = 1, \ldots, k$, deletes all edges incident to the vertices put to C.

4. Conclusion

The original model of distributed computations – the Parallel Substitution Algorithm – is applied for design of the distributed architectures implementing the fast parallel algorithms for two combinatorial optimization problems: on a maximal independent set and on a minimum weighted vertex cover. The approaches underlying these algorithms (one is based on the Monte Carlo strategy, the other – on duality of combinatorial optimization problems) share three interesting properties. The first consists in the fact that the initially parallel algorithms are designed on the basis of the approaches. The second consists in the fact that the structure of the algorithms allows us to implement them in the distributed architecture with the local connections between the processors. And the third is that the key component of the time complexity of the algorithms which is the number of required iteration steps is the binary logarithm of the input size of the problem under consideration. The parallel substitution algorithms realize these properties in the distributed architecture having the topology similar to the graph in hand.

References

[1] R. Carp and A. Wigderson, *A fast parallel algorithm for the maximal independent set problem*, J. of ACM, **32**, 1985, 762–773.

[2] M. Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput., **15**, No. 4, 1986, 1036–1053.

[3] A. Clementi, J.D.P. Rolim, and E. Urland, *Randomized parallel algorithms*, Lect. Notes in Comput. Sci., **1054**, 1996, 25–50.

[4] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization. Algorithms and Complexity*, Prentice–Hall, 1982.

[5] S. Khuller, U. Vishkin, and N. Young, *Primal–Dual Parallel Approximation Technique Applied to Weighted Set and Vertex Cover*, J. of Algorithms, 1994, 280–289.

[6] D.P. Bovet, A. Clementi, P. Crescenzi and R. Silvestri, *Parallel approximation of optimization problems*, Lect. Notes in Comput. Sci., **1054**, 1996, 7–24.

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

[8] S. Achasova, O. Bandman, V. Markova, and S. Piskunov, *Parallel Substitution Algorithm. Theory and Application*, World Scientific, 1994.

[9] Y. Pogudin, *Simulation of Fine-Grained Parallel Algorithms with the ALT (Animated Language Tools) System*, First International Workshop on Distributed Interactive Simulation and Real Time Applications (Proceedings), Eilat, Israel, Jan. 9–10, 1997, p. 22.

[10] Y. Pogudin, O. Bandman, *Simulating Cellular Computations with ALT. A Tutorial*, Lect. Notes in Comput. Sci., **1277**, 1997, 424–435.

[11] W. Feller, *An Introduction to Probability Theory and its Applications*, **1**, John Wiley, New York, 1968.