AUTOMATYKA • 2009 • Tom 13 • Zeszyt 3

Adam Sędziwy*

Collective Agent Strategies in the GRADIS Environment

1. Introduction

Graph representation is a common method used for a description of large systems having graphwise structure and/or behavior. The problem that one faces when trying to use such a representation, is a time complexity of the basic algorithms applied to the used structures, e.g. graph grammar parsing. There are known several types of graph grammars with polynomial parsing time, like ETPL(k) grammar [1, 2], which may be used to model a system evolution or changes, in a case of dynamic systems, and time non-dependent systems as well, e.g. for the syntactic pattern recognition problems. The cost paid for using polynomial graph grammars is their relatively weak descriptive power.

Another way to make the graph description applicable for practical use is applying the parallel computations. NP-complete problems like the Hamiltonian path problem or graph k-colorability can be computed in a polynomial time when using parallel Turing machine. It was proven by Kreowski [6] that the parallel graph multiset transformation reduces a complexity of the Hamiltonian paths problem to the polynomial time.

For the huge size problems even using the polynomial graph grammars may be not sufficient to retain the model efficiency, e.g. for a graph grammar with quadratic parsing time and graph with 100.000 nodes the number of operations made in parsing task is $\approx 10^{10}$. In those cases it's necessary to combine the polynomial graph grammars with a parallel computations. GRADIS platform introduced by Kotulski [3] is a computational framework that enables to fulfill both requirements.

GRADIS is a multiagent environment prepared for modeling the complex systems by means of a distributed graph representation and the agents operating on it. The primary (initial) goal of the agents is self-organizing in aim to prepare an efficient environment for performing distributed graph transformations. This environment consists of the set of subgraphs which describe the system. The questions which system parameters (properties) have to be optimized to improve model performance and what is an "efficient environment", are the problem specific ones. In some cases, to achieve a load balancing, we want to decompose a graph representation into the equally sized fragments [2]. Other times, a key issue is

* Department of Automatics, AGH University of Science and Technology, Krakow

decreasing inter-agent communication overhead. Using a proper strategy, common for all the agents, may help to achieve those goals.

There are various methods of achieving a good-quality environment. One of them, discussed in [4], is tuning an agent's cost function. In this paper the other approach is presented. It's based on the collective behavior of the agents which execute their tasks in a given order.

In the subsection 3.1 various agent strategies producing the desirable system properties are presented. In the subsection 3.3 we discuss the results of performed tests.

2. Distributed graph transformations

The concept of complementary graphs introduced by Kotulski [3] is an useful formalism for modeling the complex and distributed systems which evolve in parallel manner. The presented analysis is made for the graphs generated by random graph generator (see [5]) producing digraphs as made by polynimial parsing time graph grammar ETPL(k) [1].

Figure 1 shows the centralized graph representation of a system, G (reflecting a centralized schema of its managing and processing) (Fig. 1a), which is decomposed into a set of the *partial graphs* (Fig. 1b) managed by the autonomous agents. Such a set constitutes a decentralized description of a system from now on, called a *complementary form* of G. In a result of this decomposition the nodes indexation, which was global previously, changes as follows: 0→(0,1), 1→(3,1), 2→(0,2), 3→(3,2), 4→(1,1), 5→(0,3), 6→(1,2), 7→(2,1), 8→(2,2). The detailed discussion concerning the indexation of partial graphs nodes is made at the end of this section.

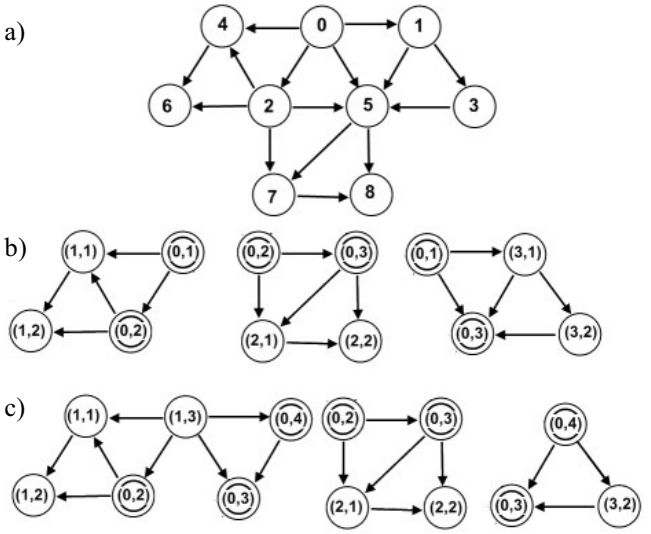

Fig. 1. Centralized and complementary form of a graph: a) Centralized graph G; b) Complementary form of G. Nodes indexed according to the indexation rules; c) Set of partial graphs after Incorporate((0,1)) call performed by the agent maintaining partial graph PG1

Some of the nodes, being common for the neighboring partial graphs (*border nodes*, marked with a double circle), are replicated to preserve a system description consistency (e.g. see nodes (0,1), (0,2), (0,3) in Fig. 1b). The borders between the partial graphs may be shifted by performing Incorporate procedure which is the atomic operation made by the agents and impacting a system. The details of Incorporate are presented in the subsection 2.1. Thus the partial graphs sizes may change (compare Fig. 1b and 1c).

There is following way of a node indexation in a set of the partial graphs: All regular nodes (i.e. non-border ones) are indexed with a pair of the form (I_{PG}, Id_{loc}) where I_{PG} denotes an index of a given partial graph PG, and Id_{loc} is a local node index, unique inside PG. An index for the borer nodes has a form (0, Id_{BN}) where Id_{BN} is a global border node index.

2.1. Incorporate Procedure

To incorporate given border node, say B, belonging to its partial graph, the agent sends a query to all other agents to discover the partial graphs containing other replicas of B, denoted as B′. Next, B′s are removed from the corresponding partial graphs together with the incident edges and neighbor nodes of B′s are supplied to the agent together with the edges connecting them with B′. Those nodes become the border ones from now on and on the other hand, B becomes non-border node in the partial graph to which B was incorporated. Figure 1c presents the resultant set of partial graphs, obtained after incorporating node (0,1) into the partial graph PG1. The node (0,1) has two replicas, in PG1 and PG2 respectively. The replica of (0,1) is removed form PG2; its neighbor nodes, namely (3,1) and (0,3) are replicated to PG1 together with the connecting edges; (3,1) becomes the border node and changes its indexation to (0,4); (0,3) remains unchanged; the node (0,1) becomes the „internal" one inside PG1 and gets indexed as (1,3).

2.2. Cost Function

The primary, global goal of the agents is to obtain an optimal partitioning, fulfilling given criteria. Such a criterion may be a partition with equally sized partial graphs (for balancing an agent workload) or minimal number of border nodes (for minimizing communication overhead).

The strategy of generating equally sized partition was discussed in [4]. It was cost function oriented strategy. The agents achieve that common goal by satisfying their individual, local conditions i.e. by minimizing cost function C.

Where σ_{opt} is a target partial graph size to be reached by the agents, $\varepsilon > 0$ is a partial graph size tolerance, $|V(PG)|$ denotes a number of border nodes in PG and a, b are the positive constants. The sample form of C is given below (see also Fig. 2).

After achieving an optimal partition the agents may start applying the graph grammar productions reflecting the system changes. This activity remains beyond the scope of this paper. In this paper we present a method of decomposition leading to a partial graphs set having assumed properties (equally sized partial graphs or minimized inter-partial graphs connections), based on agent activation order.

$$C(PG) = \begin{cases} 0, \text{ if } \| V(PG) | - \sigma_{opt} | \leq \varepsilon, \\ a(| V(PG) | - \sigma_{opt} - \varepsilon), \text{ if } | V(PG) | > \sigma_{opt} + \varepsilon | \\ b(- | V(PG) | + \sigma_{opt} - \varepsilon), \text{if } | V(PG) | < \sigma_{opt} - \varepsilon \end{cases}$$

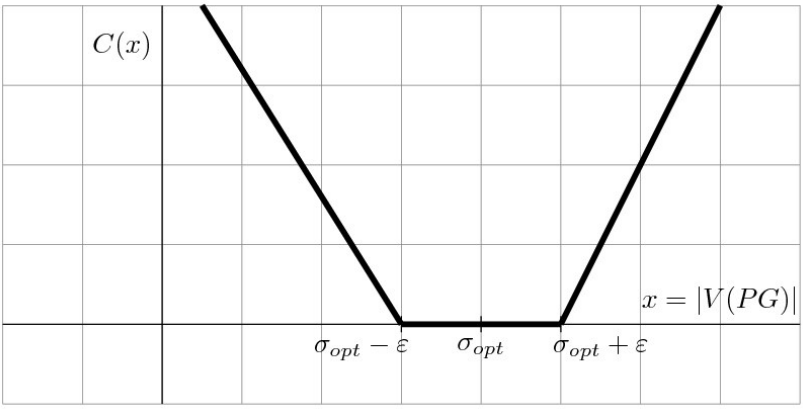

Fig. 2. The sample cost function

3. Tests

3.1. Agent activation schemas

An agent activation notion is referred to as the giving an exclusive right to performing some action to a particular agent, for example the Incorporate procedure.

The common feature of all the schemas presented below is the way a single agent acts. It tries to incorporate a single node from a neighbor agent having the worst (highest) cost function value. If all the neighbors are already optimized (cost function is equal to 0) then no action is performed.

In the schemas presented below all the agents are put in the array. The processing is performed as subsequent passes – iterations over all array elements. Thus in a single pass the agents are activated (i.e. act) accordingly to their occurrence in the array.

– **Sequential processing**. In the sequential processing an order of the elements in the array is determined by the initial centralized graph decomposition and doesn't change in the agent system lifecycle. Formally, for each two the agents X, Y: if the agent X is activated prior to the agent Y (denoted as $X \prec Y$) in *i*-th pass then $X \prec Y$ in each *j*-th pass ($j > i$), on condition that both X and Y exist in *j*-th pass. Note that the last assumption is necessary because the agents managing the graphs that became empty are removed from the array.

- **Random processing**. In that approach an order of elements in the array is randomly changed in the subsequent passes. It's achieved by making a random permutation of the agents stored in the array.
- **Sequential and random processing with perturbation**. These processing methods are the variants of the previous ones. The following modification is introduced to them. In the system lifecycle there are made two perturbations accomplished by enforcing each an agent to incorporate its randomly chosen border node. The purpose of the perturbations is making an environment escape from it's local minimum.

3.2. Experiment description

The goal of the experiment was comparison of the basic descriptors characterizing the set of the partial graphs, obtained for the different processing schemas. Those descriptors give the quantitative information how much a given partition differs from the optimal one. The following quantities were chosen:

N_{PG} – number of partial graphs. Basically this parameter depends on σ_{opt} value, but there is possible an undesired growth of N_{PG} being an artifact of the multi-agent system activity. N_{PG} value is expected to be low.

σ_{avg} – average partial graph size. Its value is expected to be close to σ_{opt} which was set to 20 in the tests.

N_{BN} – total number of border nodes in the environment. The border nodes are the interfaces between a given partial graph and the other ones. We want to reduce their number in a case when the communication between the agents increases operations costs. For that reason it's expected to be low in such the cases.

α – ratio of all border nodes to all nodes in the environment. This ratio gives a relative measure of border nodes number, that can be used to compare a quality of two graphs with different sizes. For the reasons discussed above it's expected to be low.

The single test execution has followed the steps:

1) (Initial) the centralized graph with 5.000 nodes is decomposed into the set of the partial graphs, S. None of these partial graphs is greater than 3 nodes,
2) (Intermediate) 200 passes are executed on the S; if the perturbations are present in the method, the first one is made in the middle of this step i.e. after 100 passes,
3) (Final) next 200 passes are executed on the S. In that step, each an agent removes all hanging border nodes after each 5 incorporatings; if the perturbations are present in the method, the second one is made in the middle of this step i.e. after its 100 passes.

To get a reliable output data test execution is repeated 100 times and the particular descriptors are averaged.

3.3. Results

The results of the tests are presented in the Table 1 showing the detailed data (averaged as mentioned above) from all three phases: initial, intermediate and final. Also the bar chart for the final data is presented in Figures 3–6.

Table 1
The results of the tests

Phase	N_{PG}	σ_{avg}	N_{BN}	α
Sequential processing				
Initial	3564	2.8	8279	81.5%
Intermediate	386	16.2	2392	38.2%
Final	347	16.8	1587	27.3%
Random processing				
Initial	3568	2.8	8300	81.6%
Intermediate	371	17.5	2813	43.3%
Final	337	17.7	1813	30.5%
Sequential processing with perturbation				
Initial	3567	2.8	8288	81.6%
Intermediate	368	17.1	2420	38.6%
Final	323	18	1553	26.7%
Random processing with perturbation				
Initial	3569	2.8	8298	81.6%
Intermediate	356	18.3	2828	43.4%
Final	316	18.7	1713	29.0%

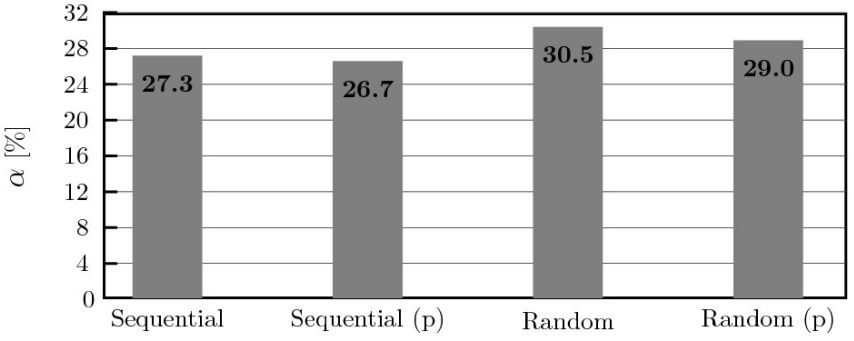

Fig. 3. Final border nodes ratio

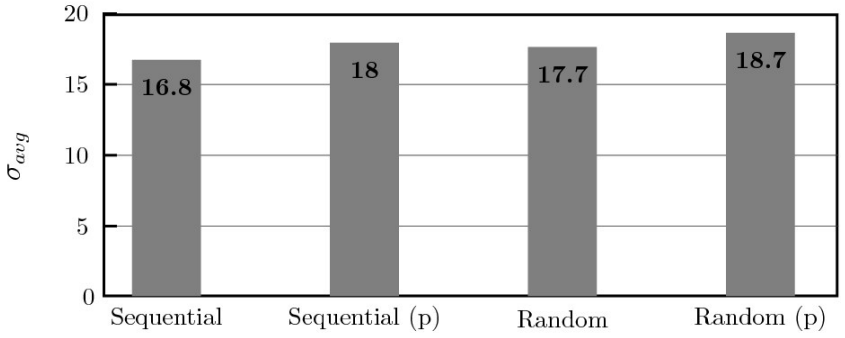

Fig. 4. Final average partial graph size

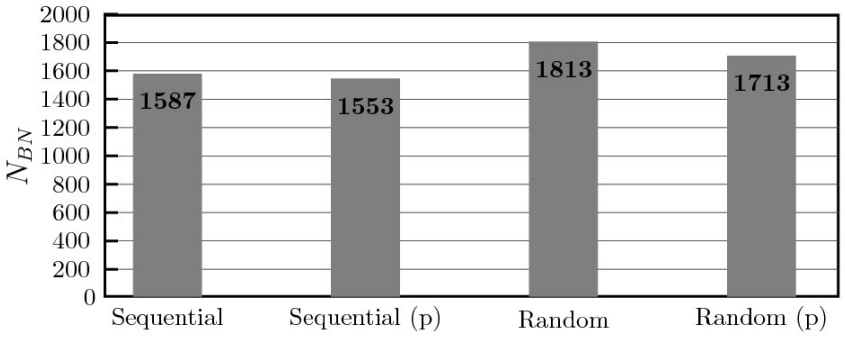

Fig. 5. Final border nodes count

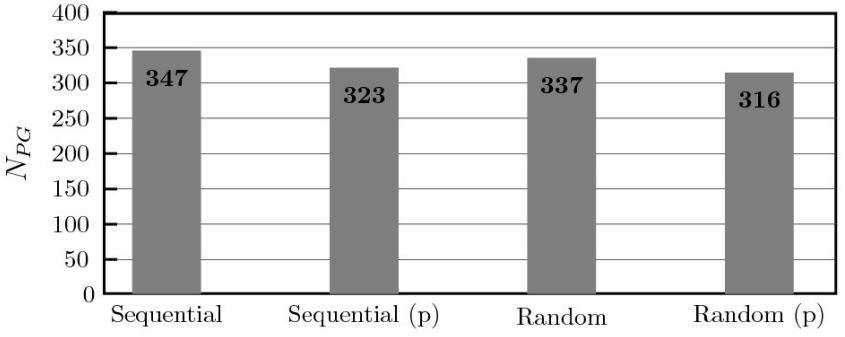

Fig. 6. Final number of partial graphs in the system

The bar charts presented below show the comparison of the descriptors obtained with the help of all presented agent strategies.

On the basis of the presented data one can make following observations.

– The models wit random partial graphs processing give better results if a criterion is the convergence to the optimal partial graph size (or in other words, minimizing $|\sigma_{avg} - \sigma_{opt}|$ value). Also procedure of perturbation improves the results.
– If a criterion is minimal total number of border nodes (N_{BN}) the sequential processing produces better results, although system perturbation may slightly reduce N_{BN}, in both cases (2.1–5.6%).

Note that the perturbations were provided to the system twice in each pass: in the middles of the intermediate and the final phase.

4. Conclusions

Common agent strategy impacts significantly a final state of a partial graphs set. Dependently on what properties of a partial graphs set we want to get, the different agent strategies have to be applied. Although the random order agent activation may slow the system convergence the random processing with perturbation should be applied to obtain the partition with partial graphs sizes being close to some optimal value (e.g. for ensuring load balancing). If the issue is the minimizing a number of border nodes (to decrease the communication overhead) the sequential processing with perturbation is the method leading to desired results. Note that the presence of perturbations improves the results in both cases.

References

[1] Flasiński M., *On the Parsing of Deterministic Graph Languages for Syntactic Pattern Recognition*. Pattern Recognition, vol. 26, 1993, 16–93.
[2] Flasiński M., *Power Properties of NCL Graph Grammars with a Polynomial Membership Problem*. Theoretical Computer Science, vol. 201, 1998, 189–231.
[3] Kotulski L., *Distributed Graphs Transformed by Multiagent System*. Artificial Intelligence and Soft Computing ICAISC, LNAI 5097, 2008, 1234–1242.
[4] Kotulski L., Sędziwy A., *Agent Framework For Decomposing a Graph Into the Equally Sized Subgraphs*. WORLDCOMP'08 Conference, Foundations of Computer Science, 2008, 2–250.
[5] Kotulski L., Sędziwy A., *Stochastyczne metody generacji IE-grafów*. Automatyka (półrocznik AGH), t. 12, z. 3, 2008, 853–861.
[6] Kreowski H.J., Kluske S., *Graph Multiset Transformation as a Framework for Massive Parallel Computation*. 4-th International Conference ICGT, LNCS 5214, 2008, 351–365.