Structural Approach to the Designing of the Complementary Graphs in the GRADIS Agent Framework

Leszek Kotulski and Adam Sędziwy

AGH University of Sciences and Technology, Institute of Automatics,
al. Mickiewicza 30, 30-059 Kraków, Poland, {kotulski,sedziwy}@agh.edu.pl

Abstract
The parallel execution of the programs increases their effectiveness, but designing such the algorithms is more difficult compared to the sequential ones. Thus any form of automatic translation of a sequential program structure into a distributed hardware environment is desired. In the paper the problem of an effective implementation of the complementary graphs concept performed by an agent environment is considered. An inter-agent cooperation leading to removing from the partial graphs the undesired substructures making the parallel graph transformations ineffective is presented.

Keywords: distributed graph transformations, agents systems, graph grammars

1 Introduction
The quantity of information being processed by the computer systems grows continuously. This causes that sequential computing systems become ineffective even if using the polynomial time algorithms. New paradigms such as agent systems which locally act and process an information and which cooperate to achieve a common goal, seem to be very promising in the field of the parallel computations. Let us note that parallel computations paradigm enables an effective solving of the NP-complete problems, when the formalism describing such a problem really support parallelism and is implemented effectively. Kreowski shows (Kreowski and Kluske, 2008) that the problem described with the help of Graph Multiset Transformation can be solved in a polynomial time (i.e. $NP_{GraphTransf} = P_{ParallelGraphTransf}$). On the other hand, a designing of a distributed application is more difficult, because we should consider such problems as data replication and proper synchronization of access to them to avoid time dependent errors or deadlocks. Thereby the most desired solution would be designing a system in a centralized form and next, its automatic translation to the distributed environment (with assurance an explicit synchronization mechanism). Such a solution is proposed in the graph transformation, where the centralized graph can be expressed in a complementary form i.e. as a set of partial graphs (Kotulski and Sędziwy, 2008) and some rules that allow to apply the "old" productions (designed for the centralized solution) in a distributed environment. This solution, called GRADIS agent framework, will
be reviewed briefly in the Section 2. In the paper we consider in a more detail the way of exchanging (by the cooperating agents) an information about the local graph structures. This way stays in a compliance with the complementary graph definition and allows to obtain some properties of the local graphs. A inter-agent cooperation leading to removing from the partial graphs undesired substructures is presented in Section 3. It appears to be more effective than the method presented in Kotulski and Sędziwy (2008).

2 Partial graphs

Definition 2.1 A \((\Sigma^v, \Sigma^e)\)-graph is a triple \((V, E, \varphi)\) where \(V\) is a nonempty set, \(E\) is a subset of \(V \times \Sigma^e \times V\), and \(\varphi : V \longrightarrow \Sigma^v\). \(V\) is set of nodes, \(E\) is set of edges and \(\varphi\) is node labeling function. We denote the family of \((\Sigma^v, \Sigma^e)\)-graphs as \(G\).

Definition 2.2 The set of graphs \(G_i = (V_i, E_i, \varphi_i)\), for \(i = 1, 2, \ldots k\), is a decomposition of graph \(G\) to the complementary form iff there exists a set of injective homomorphisms \(s_i : G_i \longrightarrow G\) such that:

1. \(\bigcup_{i=1}^{k} s_i(G_i) = G\)
2. \(\forall i, j \in \{1, \ldots k\} : s_i(V_i) \cap s_j(V_j) = s_i(Border(G_i)) \cap s_j(Border(G_j))\)
3. \(\forall w \in V_i \forall v \in V_j : \exists p = \text{PathS}(G, w, v) \Rightarrow \exists b \in \text{Border}(G_i) : s_i(b) \in p\)
4. \(\forall i \in \{1, \ldots k\} : v \in \text{Border}(G_i) \Leftrightarrow (\exists w \in G_i : w \text{ is connected with } v) \text{ or } G_i = \{v\},\)

where the function \(\text{PathS}(H, v, w)\) returns a set of the nodes belonging to the edges creating any acyclic connection between \(v\) and \(w\) in \(H\) and \(\text{Border}(G_i)\) denotes a set of all border nodes of the graph \(G_i\).

Figures 1 and 2 show the sample \(G\) and one of its complementary form respectively. The dotted lines in Fig. 1 mark the scopes of the partial graphs \(G_1, G_2, G_3\).
Figure 2: The sample $G$ in its complementary form

The intersections of the areas determine the border nodes, which are marked with the double circles in Fig.2. The indexing convention for complementary form of a graph assumes that an index of each non-border node consists of two numbers: the number of its parent partial graph and its unique local number within this graph. For the border nodes the index is a pair of the form $((-1, j))_{[A_{i_1}, A_{i_2}, \ldots]}$ where $j$ is global border node index and the subscript list $[A_{i_1}, A_{i_2}, \ldots]$ contains the names of agents holding the partial graphs where the replicas of $(-1, j)$ are stored. Thus each border node holds an information about all its replicas so that their localizations in a distributed environment can be determined immediately.

The preliminary step in preparing an agent environment for distributed graph transformations is decomposing a graph representing given problem into the set of complementary graphs (called also partial graphs). Here we use the algorithm introduced in Kotulski (2008).

2.1 Algorithm of decomposition of centralized graph. Let $H$ be a subgraph of $G$, two partial graphs $H'$ and $H''$ are created in the following steps:

1. Initially $H' = H$ and $H'' = G \setminus H$.
2. For every $v \in H$ such that there exists a node $w \in G \setminus H$ which is a neighbor of $v$ in $G$: we replicate $v$ (denoting a copy of $v$ as $v'$) and:
   - we keep $v$ in $H'$ and mark it as a border node
   - we attach node $v'$, to $H''$ with all edges connecting $v$ with $G \setminus H$ and also mark it as a border node
• proper references are added $v$ and $v'$, they allows us to determine the set of all replicas of $v$ (iff $v$ has been a border node before this operation these references are also updated in all its previous replicas).

3. Optionally, some reindexation of $V_{H'}$ and $V_{H''}$ can be made for optimizing local transformations. The unique addressing of nodes in the glued graph (after 3rd step of the algorithm or any sequence of local graph transformations) is guaranteed by the remembering their indices as a pair (local_graph_id, local_index). We also assume that, marking of the node as a border one is associated with ascribing to it of an unique index in the border nodes set (border_id is equal to -1).

Decomposition of the graph into a set of complementary graphs can be made by the interactive executions of the the above algorithm on the previously obtained graphs.

After initial graph decomposition and assigning the agents to the particular partial graphs, the agents start to cooperate and can move borderline between complementary graphs executing Incorporate procedure. In Fig.5 the result of Incorporate($(-1,1),3$) for the partial graphs shown in Fig.2 is presented. Note that the node $(-1,1)$ becomes a regular (non-border) node and neighbors of its all replicas get border nodes or (if they already were) their lists of replicas are updated:

- in $G_1$: to the node’s $(-1,2)_{[A_2,A_3]}$ list the agent $A_3$ is added; node $(1,3)$ becomes a border node, its indexation changes into $(-1,1)_{[A_3]}$ ($(-1,1)$ index was released and may be reused now) and the $A_3$ is added to the list of replicas,
- in $G_2$: to the node’s $(-1,2)_{[A_1,A_3]}$ list the agent $A_3$ is added; for $(-1,3)_{[A_3]}$ no update is required,
- in $G_3$: node $(-1,1)$ becomes a non-border node (changes indexation for the local one, namely $(3,5)$); its two adjacent border nodes with their replicas lists $(-1,1)_{[A_1]},(-1,2)_{[A_1,A_2]}$ and the connecting edges are added.

In the task of a good decomposition of a main, centralized graph we often consider the problem of loadbalancing distributed agents workload. Using attributed graphs with each a node we can associate (e.g. adding special attribute) a timestamp of the request evaluation and with each an edge we can associate the attributes holding the data describing an average/maximal/total size of the exchanged messages in the time unit. For the simplicity of the presentation we assume that all these values are equal. Thus the first described optimization task is reduced to the decomposition of $G$ into the set of equally sized partial graphs. Such a problem has been considered in Kotulski and Sędziwy (2008). Let $|V_i|$ denotes a number of nodes of the graph $G_i$. A local cost function $C$ for $G_i$ is defined as follows:

$$C(G_i) = \begin{cases} 
(N_t - |V_i|)^2, & |V_i| < N_t; \\
|V_i| - N_t, & \text{otherwise,}
\end{cases}$$

where $N_t$ is a partial graph target size. Each an agent may incorporate some border node, say $b$ (executing Incorporate procedure) if a value of its local cost function $C$ is less than for other agents holding the replicas of $b$. The above cooperation
Structural Approach to the Designing ...

Graph grammars provide a mechanism in which the local transformations of a graph can be modeled mathematically in a precise, well-defined way (Rozenberg, 1997). The main component of the graph grammars is a finite set of productions; a production, in general, is a triple \((L, R, E)\) where \(L\) and \(R\) (left- and right-hand side of a production, respectively) are the graphs and \(E\) is some embedding mechanism. Such a production can be applied to the given graph \(G\) whenever there is present a \(m\)-occurrence of \(L\) in \(G\). It is applied by removing \(m(L)\) from \(G\) and replacing it by (an isomorphic copy) of \(R\), and finally, using the embedding mechanism \(E\) to attach \(R\) to the remainder \(G \setminus m(L)\). Unfortunately, in the currently used grammars these productions are applied to the centralized graphs. Complementary graph concept and \texttt{Incorporate} procedure allow one to use these productions (designed for the centralized case) in a distributed environment (Kotulski, 2008). The general idea of the proposed application of the production \((L, R, E)\) over a distributed graph is as follows:

- an agent, say \(i\)-th, decides autonomously to apply this production when he can find the \(m\)-occurrence of \(L\) in \(G_i\), none of the border nodes is removed and all the edges being removed belong to \(G_i\).
- if \(m(L)\) match can not be found in \(G_i\), \(i\)-th agent needs to cooperate with the rest of the system. It is accomplished in three steps:
  
  \textbf{Step 1.} Gathering an information: for each node \(v \in \text{Border}(G_i) \cap V(m(L))\) we are looking for such \(k\) that the graph \(B\) denoted as \texttt{k-neighborhood}(\(v\)) covers \(L\) and all removed nodes and edges.

  \textbf{Step 2.} Preparing an environment: all nodes belonging to \(B \setminus G_i\) are incorporated to the graph \(G_i\) by the sequential execution of \texttt{Incorporate(...)} calls, made in the transactional mode.

  \textbf{Step 3.} Applying of the production: the production \((L, R, E)\) can be applied in a newly created graph \(G_i\) according to 1-st rule (local derivation).

The presented algorithm does not depend on the specific properties of a graph transformation mechanism e.g. NLC embedding transformation (in case of the algorithmic approach) (Engelfriet and Rozenberg, 1997) or single- and double-pushout (in the case of the algebraic approach) (Ehrig et al., 1997).

3 Structural methods of the complementary graph structure optimization

Agents based systems technology has generated a lot of excitement in recent years because of its promise of a new paradigm for conceptualizing, designing, and implementing software systems. The features of the multiagent systems are (Sycyra, 1998):

- the assumption that each an agent has incomplete information or capabilities for solving the problem and, thus, a limited point of view;

schema leads to the nearly equally sized partial graphs of size \(N_t\) after the tens of cooperation cycles (in a single cycle each an agent tries to execute single \texttt{Incorporate} procedure).
Figure 3: Undesired artifact in the partial graph structure.

- there is no global system control;
- a data is decentralized;
- computations are asynchronous.

An agent can use not only statistical methods for improving the complementary graph structure but it can analyze local graph structure and suggest the way of its modification as well. When focusing on a local complementary graph structure only we are not able to suggest any incorporating actions because we don’t known the structures of other partial graphs. Thus we can perform only the inverse action, i.e. we can try to relegate some border nodes from our structure. The implementation of the Relegate operation is quite simple: for a given border node \( b \) we ask any agent holding its replica to incorporate it. In this paper we consider two types of undesirable substructures that have to be removed from complementary graphs. We test how the relegation of those subgraphs improves the structure of an entire complementary graph. As the criteria we will consider:

- the difference between the average partial graph size and the target partial graph size (as in the previous section),
- the total number of border nodes. This criterion arises from the observation that we should minimize the effect of a replication of the global graph structure necessary for maintaining a distributed structure.

The first undesired structure is an orphan complementary graph, containing the isolated border nodes; in such a case we should relegate all its isolated border nodes. In extreme, such graphs will be totally incorporated into another partial graphs. The second undesired structure being considered, is presented in Fig.3. Two border nodes \( b_1, b_2 \) are connected with one internal node \( v \). Relegation of \( b_1, b_2 \) produces one border node (formerly internal, \( v \)). Such an action decrease locally a number of border nodes and should not increase its number in a global case either.

In order to check our assumptions we prepare the following experiment. We generate 100 \( ETPL(k) \) graphs \( G \) (Flasiński, 1993; Kotulski and Sędziwy, 2008) of size \( |V| = 10,000 \) nodes. Those graphs represent a centralized solution. For each of them (denoted as \( G \)) we do the following:

- \( G \) is decomposed into the set of partial graphs of the size not greater than 3,
- we execute 200 passes of agents cooperations to obtain the set partial graphs having the size \( \sim N_t = 20 \) nodes. The cost function \( C \) that impacts an agent strategy is given.
Figure 4: The total number of border nodes in the partial graphs set during the agent cooperation.

- we execute next 200 passes of agents cooperations. In that case:
  - in each pass, an incorporation basing on cost function $C$ is performed;
  - after each 5 passes the relegations based on the mentioned structural criteria are made.

To describe quantitatively the results of the experiment we use following descriptors characterizing the set of all partial graphs (PGS) being a complementary form of centralized graph $G$:

- $n_{PG}$ - number of partial graphs in PGS,
- $n_{BN}$ - the total number of border nodes in PGS,
- $n_{BN}/n_N$ - the ratio of all border nodes in PGS to all PGS nodes, expressed in percents,
- $\sigma_{avg}$ - average (in PGS) partial graph size.

We expect that the agents should minimize $n_{BN}$ and $n_{BN}/n_N$. The $n_{PG}$ value depends also on the $N_t$, i.e. target partial graph size. On the other hand $\sigma_{avg}$ is expected to converge to this value. Fig.4 shows that the evolution of the second descriptor satisfies our expectations (see also Tab.1): the total number of border nodes reduces over 5 times. Also Table 1 demonstrates that the agent strategy based on relegating undesired substructures from the partial graph, leads to improving the results obtained in the first phase of the experiment (i.e. in the first 200 passes).
<table>
<thead>
<tr>
<th></th>
<th>$n_{PG}$</th>
<th>$n_{BN}$</th>
<th>$n_{BN}/n_N$</th>
<th>$\sigma_{avg}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initially</td>
<td>7159</td>
<td>16670</td>
<td>82%</td>
<td>2.8</td>
</tr>
<tr>
<td>After 1st phase</td>
<td>776</td>
<td>4765</td>
<td>38%</td>
<td>16.1</td>
</tr>
<tr>
<td>After 2nd phase</td>
<td>701</td>
<td>3110</td>
<td>27%</td>
<td>17.2</td>
</tr>
</tbody>
</table>

Table 1: The PGS descriptors.

![Figure 5: Incorporate( (-1, 1), 3) execution (see Fig.2)](image)

4 Conclusions

Complementary graphs concept (Kotulski, 2008) brings the possibility of distributing a graph describing a problem and transforming it in a parallel manner using the graph transformation rules that has been designed for the centralized solution. The key problem is however, the method of a graph distribution, mentioned above. In some cases (e.g. in adaptive design process (Kotulski and Strug, 2008)) the natural borders between the subsystems are created but more frequently, especially when we use parallel computation paradigm to make a computation faster, there are no natural borders. Designing an optimal distribution of complementary graphs is too complex problem. In Kotulski and Sędziwy (2008) the agent cooperation basing only on the size of the partial graphs, maintained by neighboring agents, enables us to produce a set of the partial graphs of a given size in tens of iterations. Here we show that the structural information stored inside a partial graph allows us to decrease the difference between a desired partial graph size and an average partial graph size computed for PGS. It is also useful in reducing a number of border nodes in a complementary graphs system, what can
be very important in practice: the growing number of nodes replicas impacts on the system performance, e.g. due to a distributed communication overhead. The patterns of structures presented in the paper (Section 3) are very simple but one can easily define more complex ones, especially when working in conjunction with the graph grammars helping to find the occurrences of those unwanted subpatterns (analogically as in Kotulski (2009)).

References


Kreowski H.J., Kluske S.: Graph Multiset Transformation as a Framework for Massie Parallel Computation. 4-th International Conference ICGT 2008, LNCS 5214, pp. 351–365


Kotulski L.: On efficient finding subpatterns with help ETPL(k) graph grammar. – in preparation
