International Journal of Innovative Computing, Information and Control
Volume **10**, Number **1**, February **2014**

ICIC International ©2014 ISSN 1349-4198
pp. 101–113

EFFECTIVE GRAPH REPRESENTATION SUPPORTING MULTI-AGENT DISTRIBUTED COMPUTING

Adam Sędziwy

Department of Applied Computer Science
AGH University of Science and Technology
al. Mickiewicza 30, 30-059 Kraków, Poland
sedziwy@agh.edu.pl

Received December 2012; revised April 2013

Abstract. *The parallel processing is an effective approach to solving those high complexity problems which may be represented as a set of independent or loosely coupled subproblems. In the latter case, however, the critical factor for a computation time is an overhead generated by communication among particular subtasks. The decomposition of a graph-based computational problem allows transforming it into a set of subproblems to be processed in parallel. A decomposition method should guarantee a good performance of parallel computations with respect to communication and synchronization among agents managing a distributed representation of a considered system. In this paper we present the novel method of a decomposition, reducing coupling among subproblems and thus minimizing a required cooperation among agents. Comparison and performance tests are also included.*

Keywords: Graph, Slashed form, Distributed computing, Multi-agent system, Lighting computations

1. **Introduction.**

Application of multi-agent systems in such domains as smart grid systems or distributed information processing enforces using solutions based on scalable, dependable and computationally effective formal models. Such models have to support decomposability (required for achieving the scalability) and, on the other side, efficient cooperation of agents.

Graph-based formal models are appropriate for such purposes and provide a flexible and widely used modeling framework for solving various types of problems, either dynamic or static, in such areas as system specification, software generation or task allocation control [7, 13]. The limitation for their applicability is the time complexity of parsing or membership problems. It may be overcome, however, by using distributed computational paradigm or by decreasing an expressive power of a graph grammar if possible. GRADIS (GRAph DIStributed) multi-agent environment designed to perform such distributed graph transformations was presented in [8, 11].

Multi-agent systems are used in computer aided design (CAD) related problems in the automotive industry [2], in constructional tasks [18], collaborative systems [12] or the adaptive design [10]. The domain lying on the border of CAD and control problems is large-scale intelligent lighting [14]. Large-scale intelligent lighting (abbrev. LaSIL) problem consists of two subproblems. The first one is designing a distribution of lighting points in an urban area in such a way that values of given lighting parameters (luminance, threshold increment and so on) meet compulsory standards and, on the other side, exploitation costs generated by a power consumption are minimized [15]. The second subproblem is designing a smart, adaptive lighting control system supported by distributed sensors, working in the mode of a weak real time system. Such a system may be a part of a smart grid solution (see Green AGH Campus [16]). The similar problem, namely computer simulation of an adaptive illumination facility was solved using environment descriptions based on cellular structures (Situated Cellular Agents and Dissipative Multilayered Automata Network) [1]. The weakness of this approach, however, is the limited applicability to problems formulated above.

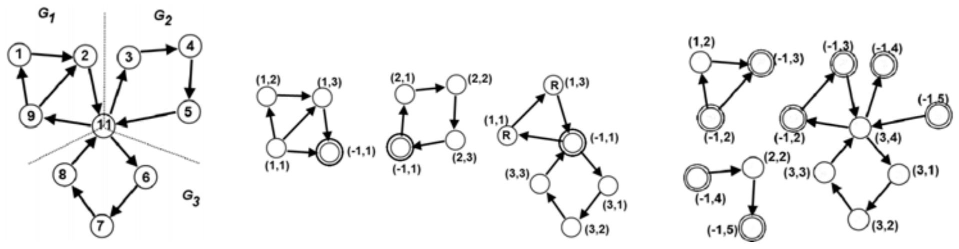
LaSIL problems are characterized by the high computational complexity of underlying computations. Solving the first LaSIL subproblem (i.e., setting up an optimal distribution of lamps) is a compound of multiple nonlinear multidimensional and multiobjective optimization tasks. In the second phase (i.e., lighting control) optimizations which follow environmental changes are performed. The crucial constraint of this phase is a limited system adaptation time. Note that the second phase is perpetual.

As the lighting computations may be broken into separate (or weakly coupled in overlapping areas) local tasks, the first step to be done is assigning an appropriate representation to a problem. The graph formalism seems to be the most convenient one for solving LaSIL due to its correspondence with the problem structure and the mentioned capability of modeling system architecture and dynamics implied by changes of an actual environmental state. Fragmentation of a problem's graph representation into a set of smaller subtasks precedes a multi-agent system deployment, which enables a parallel problem solving.

In the paper we introduce the *slashed representation* model which replaces the formalism of RCGs (Replicated Complementary Graphs) [9] used in the GRADIS environment. Such replacement is possible because the specificity of LaSIL problem eliminates the need for replicating fragments of subgraphs. The *slashed representation*, unlike RCG approach, does not support replication but reduces and simplifies a cooperation between agents and thereby improves a system performance and dependability. The novelty of the approach is that each shared element (so called *dummy node*) is common for exactly two subgraphs and does not depend on a decomposition structure. The consequences of this fact are discussed in detail in Section 6.

The article is organized as follows. In Section 2 the concept of replicated complementary graphs is sketched. Section 3 introduces the notion of the *slashed form* of a graph and the related algorithms. Section 4 describes Incorporate procedure for a slashed representation: execution time of this procedure may generate significant contribution to the total computation time. Section 5 presents the performance comparison for RCG and slashed representations, and the analysis of the performance of the FIPA (Foundation for Intelligent Physical Agents) compliant multi-agent system performing exemplary photometric computations. Section 6 contains the discussion on an effectiveness issue. Final conclusions are presented in Section 7.

2. **Related Work – Replicated Complementary Graphs.** The Replicated Complementary Graphs (RCG) concept enables a decomposition of a centralized graph model of a system into a set of subgraphs (so called *complementary graphs*) with replication of some fragments of those graphs. Decomposition is based on a recursive splitting of a centralized graph. Split procedure may be described as follows. A given graph G is divided into two subgraphs G_1, G_2 in such a way that nodes which are shared by G_1 and G_2 (so called *border nodes*) are replicated together with edges connecting them with other border nodes common for both subgraphs. Borders between RCGs obtained by a decomposition of a centralized graph may change in a result of *incorporating* one or more border nodes by agents maintaining particular complementary graphs. The detailed description of Incorporate procedure for that approach may be found in [11]. In certain circumstances the replication of a given fragment may be performed instead of incorporating it, for example



(a) (b) (c)

FIGURE 1. (a) G in the centralized form. The dotted line marks planned borders of decomposition, (b) $\{G_1, G_2, G_3\}$ – the complementary form of G, (c) the complementary form of G after incorporating $(-1, 1)$ into G_3.

in the case when an agent needs to know a neighborhood of a given vertex and no changes are made on that neighborhood. That results in a lower complexity expressed in a number of exchanged messages.

The exemplary graph and its replicated complementary form are shown in Figure 1. The RCG form of G has only one border node shared by all subgraphs (node indexed with $(-1, 1)$ in Figure 1(b), marked with the double circle). Nodes $(1, 1)$, $(1, 3)$ belonging to G_1 are replicated and attached to G_3 together with the connecting edge and two edges incident to the vertex indexed with $(-1, 1)$. They are labeled with R in G_3.

Figure 1(c) presents the complementary form of G (shown in Figure 1(b)) after incorporating $(-1, 1)$ into G_3. Performing that operation required locking replicas of $(-1, 1)$ together with adjacent nodes: $(1, 1)$, $(1, 3)$, $(2, 1)$, $(2, 3)$, $(3, 1)$, $(3, 3)$. After completing the operation the reindexation of corresponding nodes was made.

3. **Slashed Form of Centralized Graph.**

The concept of the slashed form of a centralized graph aims at reducing coupling among subgraphs (generated by border nodes) in a distributed representation and thereby simplifying operations performed by maintaining agents in a distributed environment. The basic idea of that approach is splitting edges rather than the multiple replication of existing nodes of a centralized graph as it was made in RCG environment. The formal definitions are presented below.

Definition 3.1. *(Σ^v, Σ^e, A)-graph is a triple $G = (V, E, \varphi)$ where V is nonempty set of nodes, $E \subseteq V \times (\Sigma^e \times A) \times V$ is a set of directed edges, $\varphi : V \longrightarrow \Sigma^v$ is a labeling function, Σ^v and Σ^e denote sets of node and edge labels respectively and A is a set of edge attributes. We denote the family of (Σ^v, Σ^e, A)-graphs as $\mathcal{G}$.*

Definition 3.1 modifies the (Σ^v, Σ^e)-graph notion (see [9]): we change the edge structure from $V \times \Sigma^e \times V$ to $V \times (\Sigma^e \times A) \times V$ in order to store all required data in edge attributes. These data include slashing details (e.g., geometric coordinates) but also problem specific information (e.g., architectural details of adjacent buildings). (Σ^v, Σ^e)-graph definition can be also extended, e.g., by introducing an attributing function for nodes, but such an extension does not impact further considerations so it will not be considered here.

Definition 3.2 (Slashed form of G)**.** *Let $G = (V, E, \varphi) \in \mathcal{G}$. A set $\{G_i\}$ of graphs is defined as follows.*

- *$G_i = (V_i, E_i, \varphi_i) \in \mathcal{G}$ and $V_i = C_i \cup D_i$, $C_i \cap D_i = \emptyset$, where C_i is a set of **core nodes**, D_i denotes a set of **dummy nodes** and $\varphi_i \equiv \varphi|_{V_i}$,*
- *$\bigcup_i C_i = V$ where C_i, C_j are mutually disjoint for $i \neq j$,*
- *$\forall v \in D_i \exists! v' \in D_j$ $(i \neq j)$ such that v' is the replica of v; $\forall v \in D_i \deg(v) = 1$,*

- *$\forall e \in E_i : e$ is incident to at last one dummy node.*

An edge incident to a dummy node is called a ***border edge****. The set of all border edges in G_i is denoted as E_i^b. A set $E_i^c = E_i - E_i^b$ is referred to as a set of* ***core edges*** *of G_i. Let $M = \Sigma^e \times A$, then a set $\{G_i\}$ as defined above is referred to as a* ***slashed form*** *of G, and denoted $\not{G}$, iff following conditions are satisfied.*

1. *$\forall G_i^c = (C_i, E_i^c, \varphi_i|_{C_i}), \exists H_i \subset G : H_i \overset{\alpha}{\simeq} G_i^c$ (α denotes an isomorphic mapping between graphs) and H_i, H_j are disjoint for $i \neq j$.*
2. *$\exists f : M^2 \to M$ – a bijective mapping $\forall (e, e') \in E_i^b \times E_j^b$ $(i \neq j)$ such that (i) $e = (x_c, m, v) \in C_i \times M \times D_i$, $e' = (v', m', y_c) \in D_j \times M \times C_j$, (ii) v' is a replica of $v : \exists! e_{ij} = (x, m_e, y) \in E$ such that $x_c = \alpha(x)$, $y_c = \alpha(y)$ and $f(m, m') = m_e$. e_{ij} is called a* ***slashed edge*** *associated with replicated dummy nodes v, v'.*
3. *$\forall e = (x, m, y) \in E$: (i) $\exists! e_c \in E_i^c$ for some i, such that $e_c = \alpha(e)$ or (ii) $\exists!(v, v') \in D_i \times D_j$ for some i, j, such that e is a slashed edge associated with v and v'.*

$G_i \in \not{G}$ is called a ***slashed component*** *of G.*

Note that f mapping recovers labeling/attributing data of a slashed edge basing on a labeling/attributing of given border edges.

Remarks. (i) To preserve the clarity of images we neglect the attributing/labeling of graph edges in figures. (ii) A border edge incident with a dummy vertex v will be denoted as e_v. Similarly, a border edge incident with a dummy node indexed with an index id will be denoted as e_{id}.

One can slash a core edge $e = (x, m, y) \in G$ to a pair of border ones $e_1 = (x, m_1, d)$, $e_2 = (d, m_2, y)$ in such a way that a dummy node d refers to the same physical entity as x does. In particular when xG and d represent points of $\mathbb{R}^2$ or $\mathbb{R}^3$ space. Thus, using a slashed representation does not generate additional geometric data related to a slashing point.

Example 3.1. *Let us consider the street area $\mathcal{S}$ (Figure 2) in which traffic is described by the cellular automaton (CA). In a graph representation a street is modeled by an edge while from a CA perspective one views $\mathcal{S}$ as a rectangular grid ([17]). To ensure consistency of those two representations one has to encode a geometric structure of $\mathcal{S}$ as a graph edge attribute. Assuming that the considered street has the rectangular shape with no holes, and that coordinates of corner cells are given by vectors $\mathbf{P}_i$ $(i = 1, \ldots, 4)$, an edge $e = (x, m, y)$ describing $\mathcal{S}$ may be characterized by*

$$x = \frac{1}{2}(\mathbf{P}_1 + \mathbf{P}_4), \quad y = \frac{1}{2}(\mathbf{P}_2 + \mathbf{P}_3), \quad \Sigma^e \times A \ni m = (\sigma_e, \{\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3, \mathbf{P}_4\}),$$

where σ_e is some edge label. To parallelize computations performed on CA we will decompose grid by slashing it together with its "dual" (i.e., graph) representation. To do this we select two points $(\mathbf{S}_1, \mathbf{S}_2)$ corresponding to the related dummy node. Thus we obtain the following border edges associated with respectively CA_1 and CA_2 which replace CA:

$$e_1 = (x, m_1, d), \quad e_2 = (d, m_2, y),$$

where $d = \frac{1}{2}(\mathbf{S}_1 + \mathbf{S}_2), m_1 = (\sigma_1, \{\mathbf{P}_1, \mathbf{S}_1, \mathbf{S}_2, \mathbf{P}_4\}), m_2 = (\sigma_2, \{\mathbf{S}_1, \mathbf{P}_2, \mathbf{P}_3, \mathbf{S}_2\})$ and σ_1, σ_2 are some arbitrary edge labels.

In Figure 3 the centralized and the slashed form of the given graph G are shown. Core nodes are marked as circles and dummy ones as squares. The following indexing convention is used for slashed components (see Figure 3(b)). A core node index has the form (i, k) where i is a unique, within $\not{G}$, identifier of a slashed component G_i and k is a unique, within G_i, index of this node. A dummy node index has the form $(-1, k)_r$ where k is a globally unique identifier of a node. Additionally, a subscript r denotes a reference

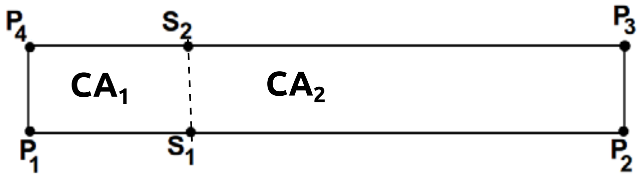


FIGURE 2. The rectangular street area $\mathcal{S}$ and slashing points $\mathbf{S}_1, \mathbf{S}_2$. CA$_1$ and CA$_2$ denote cellular automata ascribed to relevant areas.

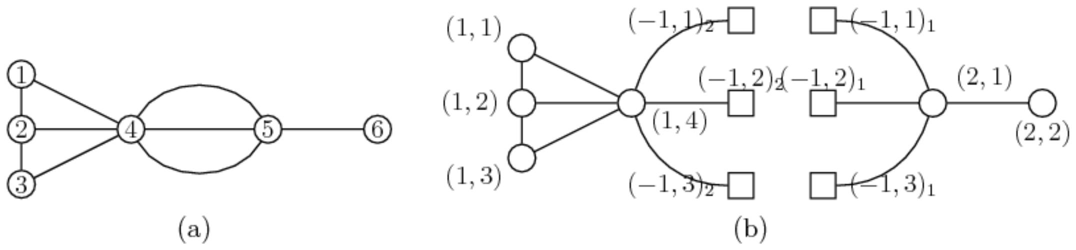


FIGURE 3. (a) Graph G, (b) $\not{\mathcal{G}}$ representation

to a slashed component (or its maintaining agent) hosting a replica of a given dummy node. Using such a subscript allows for immediate localization of a replica. To simplify the notation subscripts will be neglected within the text, unless needed. Note that a dummy vertex and its replica share a common index and differ in reference subscripts only: $(-1,k)_{r1}$, $(-1,k)_{r2}$.

Definition 3.2 introduces the formal background for decomposition of a graph belonging to $\mathcal{G}$ but it is poorly applicable in a practical use. Switching between centralized and slashed representation is enabled by two algorithms, Split and Merge, which are introduced below.

Algorithm 1: Split(G, V_c)

```
input : $G = (C \cup D, E, \varphi)$ – graph to be decomposed, $V_c \subset C$ – a subset of the set
        of core nodes
output: $\not{\mathcal{G}} = \{G_1, G_2\}$ where $G_i = (C_i \cup D_i, E_i, \varphi_i)$
1  begin
2      $C_1 \leftarrow V_c$; $D_1 \leftarrow$ all dummy nodes from $D$, adjacent to $V_c$ in $G$;
3      $E_1 \leftarrow$ all edges connecting nodes from $C_1 \cup D_1$ in $G$;
4      $\varphi_1 \leftarrow \varphi|_{C_1 \cup D_1}$;
5      $C_2 \leftarrow V - V_c$; $D_2 \leftarrow$ all dummy nodes from $D$, adjacent to $C - V_c$ in $G$;
6      $E_2 \leftarrow$ all edges connecting nodes from $C_2 \cup D_2$ in $G$;
7      $\varphi_2 \leftarrow \varphi|_{C_2 \cup D_2}$;
8      $E_{conn} \leftarrow$ all edges from $E$ connecting $V_c$ and $C - V_c$ in $G$;
9      foreach Edge $e = (x, m_e, y) \in E_{conn}$ do
10         Create dummy node $v$ and its replica $v'$; Set $\varphi_1(v)$, $\varphi_2(v')$;
11         $(m, m') \leftarrow f^{-1}(m_e)$; $e_v \leftarrow (x, m, v)$, $e_{v'} \leftarrow (v', m', y)$ [1];
12         $D_1 \leftarrow D_1 \cup \{v\}$; $E_1 \leftarrow E_1 \cup \{e_v\}$; $D_2 \leftarrow D_2 \cup \{v'\}$; $E_2 \leftarrow E_2 \cup \{e_{v'}\}$;
13 end
14 return $\{G_1, G_2\}$
```

Split. It is assumed that $G = (C \cup D, E, \varphi) \in \mathcal{G}$ is given, where C is a set of core nodes, D denotes a set of dummy ones and $V_c \subset C$. Initially, for G representing a main graph, $D = \emptyset$. Algorithm 1 presents splitting $G \in \mathcal{G}$ into two slashed components G_1, G_2 (i.e., $\not{G} = \{G_1, G_2\}$) according to the given set V_c of core nodes. To obtain a deeper decomposition, the Split procedure has to be applied recursively on G. The time complexity of Algorithm 1 is $\mathcal{O}(|E|) = \mathcal{O}(|C \cup D|^2)$. In a result of performing the Split procedure on G_i a new component is produced. Replicas of dummy vertices which have been moved from G_i to this new component have to be requested to change references of hosting components in their indices.

Merge. Having a slashed form of G, $\not{G} = \{G_1, G_2, \ldots, G_n\}$, one can reassemble a centralized representation, G. It may be accomplished by iterative calls of the Merge procedure described by Algorithm 2 which matches replicas of particular dummy nodes and corresponding border edges. The time complexity of Algorithm 2 is $\mathcal{O}(|E|) = \mathcal{O}(|C \cup D|^2)$.

```
Algorithm 2: Merge($G_i, G_j$)
   input : $G_i, G_j$ – slashed components $G_i = (C_i \cup D_i, E_i, \varphi_i)$ to be merged
   output: $G_{ij}$ – graph obtained as a result of merging input ones
 1 begin
 2     $D_i^{common} \leftarrow \{v \in D_i : \exists v' \in D_j : v'$ is the replica of $v\}$;
 3     $D_j^{common} \leftarrow$ replicas of $D_i^{common}$ in $G_j$;
 4     $C_{ij} \leftarrow C_i \cup C_j$; $D_{ij} \leftarrow (D_i - D_i^{common}) \cup (D_j - D_j^{common})$;
 5     $E_{ij} \leftarrow (E_i - \{$Edges of $G_i$ incident to $D_i^{common}\}) \cup (E_j -$
       $\{$Edges of $G_j$ incident to $D_j^{common}\})$;
 6     $\varphi_{ij} \leftarrow (\varphi_i \cup \varphi_j)|_{C_{ij} \cup D_{ij}}$;
 7     for $e_v = (x, m, v) \in E_i, e_{v'} = (v', m', y) \in E_j$ where $v'$ is the replica of $v$ [1] do
 8         $m_e \leftarrow f(m, m')$; $E_{ij} \leftarrow E_{ij} \cup \{(x, m_e, y)\}$;
 9 end
10 return $G_{ij} = (C_{ij} \cup D_{ij}, E_{ij}, \varphi_{ij})$
```

4. **Incorporate Procedure for Slashed Representation.** Incorporate procedure is the most frequently performed operation in an agent system being considered. For this reason its efficiency impacts strongly on the overall system performance. A slashed component's border consisting of all its dummy nodes, may be shifted by calling Incorporate on a given border edge e_v incident with a dummy vertex v. It is accomplished by matching v and its replica v' being an endpoint of a border edge $e_{v'}$ in another component and recovering an underlying edge form e_v and $e_{v'}$. Thus exactly one core node and possibly some dummy ones are attached to a given component. Figure 4 presents the slashed form of some G and the change in $\not{G}$ implied by incorporation of the border edge $e_{(-1,1)}$ by G_1 which will be called an initiating component. Let us consider this example in detail. The operation Incorporate(G_i, e_v) for G_1 and the border edge e_v incident with the dummy node $v = (-1, 1)_2$ consists of following steps:

1. Get the replica of v, namely $v' = (-1, 1)_1$, localized in G_2 (as referenced by the subscript of the v's index).
2. Attach the core node u indexed by $(2, 1)$, neighboring v', to G_1 and reindex u to $(1, 3)$ (for the compliance with the indexation in G_1).
3. Attach to G_1 all border edges incident to u. Let us assume WLOG that a border edge $e_d = (u, m, d)$. Two scenarios are possible. In the first one a dummy vertex d (in the example d indexed with $(-1, 2)$) has a replica d' in an initiating graph. In that case

[1] $e_v = (v, m, x)$ and $e_{v'} = (y, m', v')$ if e is directed inversely.

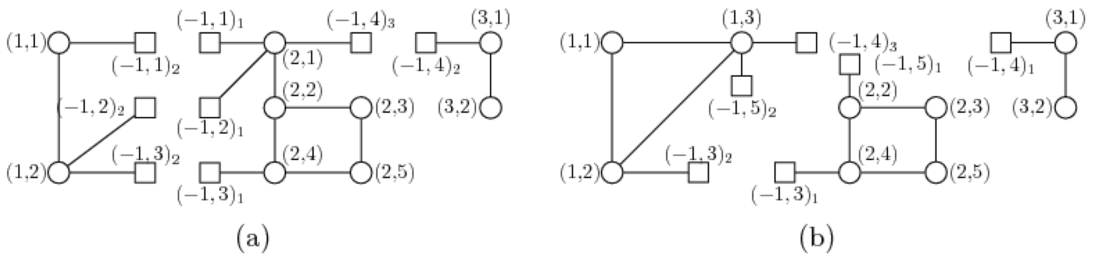


FIGURE 4. (a) $\mathcal{G} = \{G_1, G_2, G_3\}$, (b) $\mathcal{G}$ after incorporating $e_{(-1,1)}$ by G_1

d, d' are removed and a corresponding slashed edge e is recovered (in Figure 4(b) $e = ((1,3), m_e, (1,2)))$. In the second scenario d does not match any dummy vertex in an initiating graph and it is attached together with e_d to an initiating graph. Note that e_d may not be split because an edge belonging to a slashed component may be incident with at last one dummy node.

4. Split all edges connecting u with other core nodes (see splitting in Algorithm 1, line 9 and next). Resultant dummy vertices are attached to the initiating graph together with incident edges. The edge $e = ((2,2), m_e, (2,1))$ shown in Figure 4(a) has been split, resultant $e_{(-1,5)} = ((-1,5), m, (1,3))$ has been attached to G_1 (note that u was reindexed previously) and $e'_{(-1,5)} = ((2,2), m', (-1,5))$ remains in G_2.

Incorporate(G_0, e_v) procedure execution has one phase only. Let us assume that v' is a replica of v and c is a core node being the neighbor of v'. To perform Incorporate(G_0, e_v) an initiator agent (say A_0) sends a `commit` request to a responder agent (say A_1) to gather c vertex and its neighborhood (which may contain newly created dummy nodes). A_1 may either supply requested nodes/edges or reject a request (a response is `locked`) if one or more of requested nodes/edges are already locked by some other initiators. If A_0 was replied with a requested subgraph then it sends `update` requests to agents maintaining replicas of all dummy vertices which just have been attached to G_0. A_0 requests to update references for replicas of those vertices. Note that references to those agents are known to A_0 as they are included in dummy nodes' indices. It is assumed that each agent keeps the information where his particular dummy nodes were moved to. Hence, if an agent being a recipient of an `update` message has not a dummy vertex replica u' to be updated, because u' node was previously moved to some other slashed component G_k, then it forwards `update` request to the corresponding agent A_k and informs A_0 that a replica location changed. In the case when a response from A_1 is `locked` A_0 repeats the operation with a random delay.

The Incorporate protocol described above is more efficient compared with its form used in the case RCG representations which acts according to the 2PC protocol semantics.

Figure 4(b) demonstrates an index update: after incorporating $e_{(-1,1)}$ by G_1 the index $(-1,4)_2$ in slashed component G_3 has changed to $(-1,4)_1$.

5. **Performance Tests.** To compare the effectiveness of RCG and slashed representations we conducted two tests. In the first one we compared both the number of messages sent by agents during the operation of an optimization of a set of subgraphs and the final number of produced subgraphs. For the better efficiency of a multi-agent system populated on this set and solving a LaSIL problem, a number of included subgraphs should be low. In the second test we compared a convergence of the relaxation algorithm for both approaches.

The goal of the third test was investigating effectiveness of the computations based on slashed representation with respect to CPU resources.

Multi-agent system architecture. The multi-agent system used in tests consisted of two types of agents. *Operational Agent* (OA) was responsible for recursive decomposition of a problem into smaller subproblems and, when the required subproblem size had been reached, making the photometric calculations. Algorithm 3 presents a life cycle of an OA. It may be noted that each OA is capable of creating new OAs when it finds that its graph order exceeds some arbitrary value. A decomposition method used in the algorithm is the simple bisection but other schemes may be also used.

```
Algorithm 3: OALifeCycle($G$)
input: $G$ – graph maintained by OA
1  begin
2      while Number of core vertices in $G >$ maximum acceptable number of core
       vertices in a problem do
3          $N_c \leftarrow$ number of core vertices in $G$;
4          $k = \lfloor \frac{N_c}{2} \rfloor$;
5          $H \leftarrow$ subgraph (slashed component) of $G$ such that $H$ has at last $k$ core
           vertices;
6          $G \leftarrow G - H$;
7          Create new Operational Agent on $H$;        // starting OALifeCycle($H$)
                                                       // on newly created OA
8      Compute the average illuminance on each street represented in $G$;
9      Report results to the Registry Agent;
10     Terminate yourself;
11 end
```

Registry Agent's (RA) goal was gathering the results received successively from particular OAs. The supplementary capability which enables an RA to resolve a problem of missing results (caused by OAs' failures) may be easily added to a Registry Agent. It may be achieved by creating extra OAs assigned to subgraphs for which result data were lost.

Number of messages and subgraphs. In the first test, the optimization objective was achieving equally sized subgraphs (see [11]). In both cases the following *relaxation* method was applied.[1] If a given subgraph G_i has an optimal size then no action is undertaken. Otherwise some neighboring subgraph G_k is selected and incorporated into G_i and next, the resultant G'_i is split into G''_i, G'_k such that $|V(G''_i)| \approx |V(G'_k)|$ (Figure 5). Updating messages are sent by a maintaining agent while executing incorporate and split operations.

The technical details of the test are the following. First, the random 2000-node IE-graph G (see [4]) was generated as an input for the relaxation process performed for both approaches (i.e., for RCG and slashed forms). Next, for each representation, G was decomposed into subgraphs having not more than 3 nodes/core nodes and, in the sequel, the relaxation was performed. The subgraph target size for the system self-optimization process was set to 30 ± 5. The process was interrupted when the ratio of optimized subgraphs reached 90%. Then M_{tot} – a total number of messages sent by the agent system

[1] The *relaxation* described here is inspired by the numeric method of solving the BVP for the stationary temperature distribution $\nabla^2 T = 0$: in both cases we iteratively compute an average of values ascribed to a given element and neighboring ones.

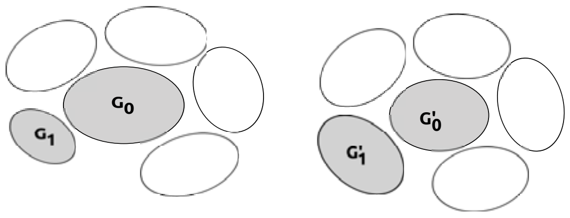
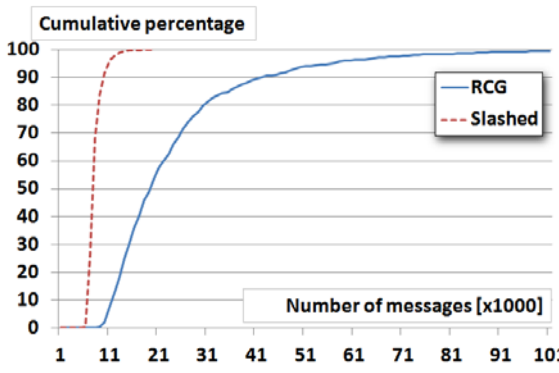


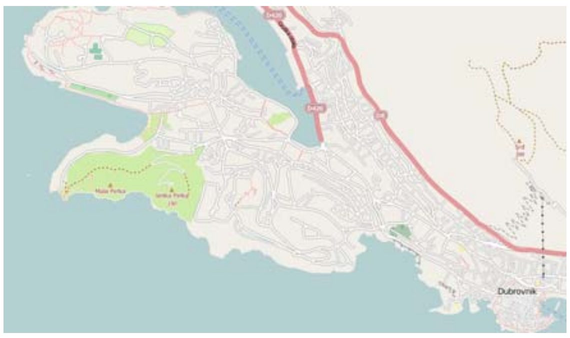
FIGURE 5. The relaxation method concept: an agent maintaining the graph G_0 selects G_1 (at left), then averages both sizes by moving some vertices from G_0 to G_1.

TABLE 1. Test results

Parameter	**RCG**	**Slashed**
Variability range (I_{var})	[8571,204274]	[5819,19165]
Averaged number of sent messages (M_{tot}^{avg})	24106	7902
Average final number of agents (Q^{avg})	84.4	37.7
Standard deviation for Q^{avg} (σ)	1.5	1.4



(a)



(b)

FIGURE 6. (a) Cumulative frequency distribution for total numbers of updating messages sent in optimization process, (b) OpenStreetMap of Dubrovnik, Croatia

was recorded. Such a single test pass was repeated 1000 times and following statistics were calculated: cumulative frequency distribution for M_{tot}, averaged number of sent messages $M_{tot}^{avg} = \frac{1}{N}\sum_{i=1}^{N} M_{tot,i}$, where $N = 1000$, and the variability range $I_{var} = [n_1, n_2]$ for M_{tot}, where $n_1 = \min_i M_{tot,i}$ and $n_2 = \max_i M_{tot,i}$. Table 1 shows results obtained for both representations. The cumulative frequency distributions (normalized to 100%) for M_{tot} generated for both RCG and slashed forms are presented in Figure 6(a). This chart shows for a given number of messages m, what percent of all test passes, produced a total number of messages less than or equal to m. The test described above shows that even for sparse graph structures like IE-graphs the average number of sent messages was reduced by 3.05.

The average final number of agents was computed as $Q^{avg} = \frac{1}{N}\sum_{i=1}^{N} Q_i$, where Q_i is a final number of agents for an ith test pass. Additionally standard deviation σ was calculated while testing each representation. The results given in Table 1 show that the number of agents in the RCG underlaid multi-agent system is 2.2 times higher than that for the slashed one.

Convergence. To compare the convergence of the relaxation algorithm for both graph representations, the Dubrovnik city map was used as the test case (Figure 6(b)). Similarly as in the first test, the goal was decomposing centralized graph G_{Dub} corresponding to the selected map into equally sized subgraphs, where the target size was set to 20 ± 4. The order of G_{Dub} was $|V| = 4105$ and the size $|E| = 4547$. Such a decomposition is the preliminary action for LaSIL problems, prior to the optimizing or controlling tasks which are performed on resultant subgraphs. The test showed that for the RCG data model the system cannot be optimized. After about 5 iterations the ratio of optimized subgraphs stabilizes around the average $\overline{s} = 55.4\%$ and fluctuates with the standard deviation $\sigma = 3.4$ (the acceptance threshold for the optimization was set to 90% for both representations). For the slashed form the ratio of optimized subgraphs reached 91.9% in 310 iterations.

Speedup. In this test we measure the total time of computations (i.e., computations for entire considered urban area) in dependence on the number of concurrently running threads (N_t). As the test case we use a multi-agent system making photometric computations. The detailed background of the test may be sketched by means of the following three logical layers.
Graph layer. As previously the area was given as the OSM map of an urban space (the city of Rome, Figure 7). The map (or precisely, its layer related to roads) was transformed to the graph (2631 nodes, 2888 edges) which was recursively decomposed by the multi-agent system into the slashed form. The recursion was stopped when a subgraph contained not more than 30 core vertices.
Agent layer. This tier was described at the beginning of this section. From a test perspective it should pointed out that only one thread is associated with each agent, either OA or RA.
Calculation layer. The goal of computations was calculating the average illuminance of each street belonging to the given area. Computations were made in compliance with relevant standards (European standards EN 13201:3).

The multi-agent system was implemented on JADE which is FIPA compliant agent framework [3, 6]. The speedup test was performed on the eight-core processor AMD X8 FX-8150 (one thread per core).

An input for a single test pass was the number N_t of threads which were allowed to run concurrently. An output value was the time $T(N_t)$ required to complete photometric computations for the given city area. To smooth the fluctuations of results related to non

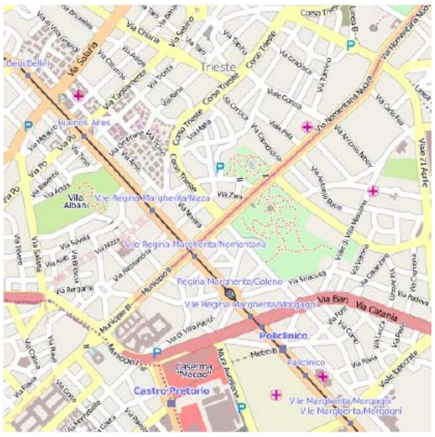


FIGURE 7. Part of Rome, Italy, whose graph representation was processed in tests (source: OpenStreetMap)

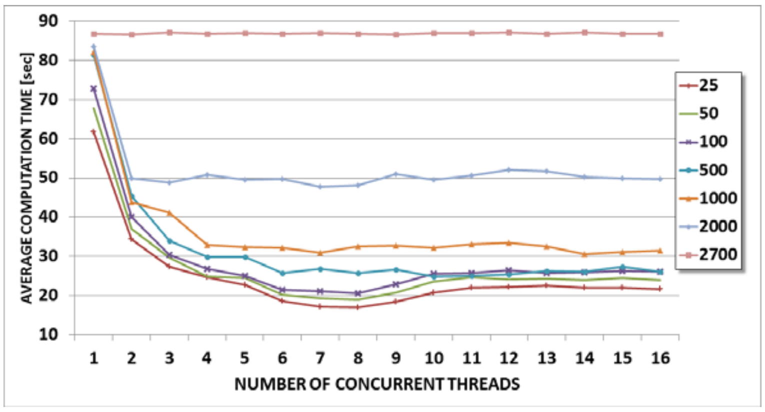


FIGURE 8. Computation time as the function of the number of concurrent threads. Particular series correspond to different N_c of decomposition.

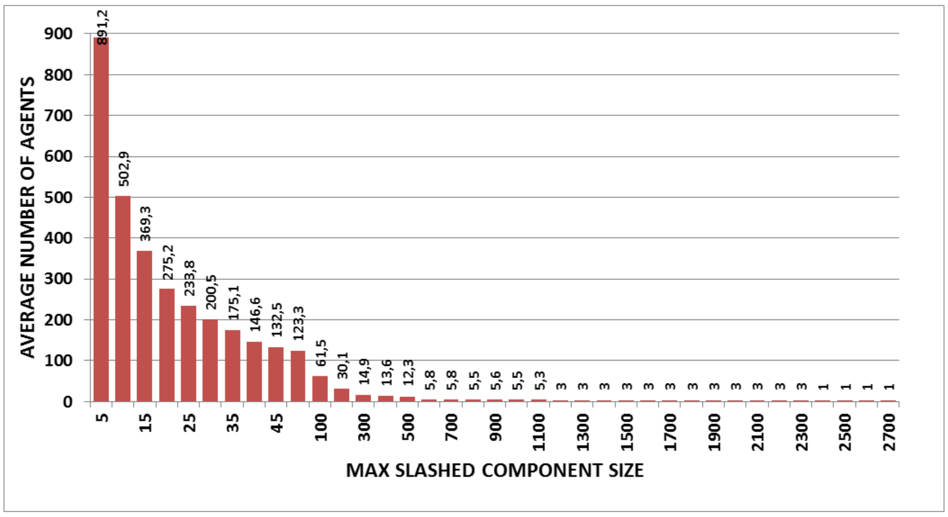


FIGURE 9. Average number of agents as the function of maximum subgraph size

deterministic system behavior such a single test pass was repeated $n = 5$ times and the average time $T_{avg}(N_t) = \frac{1}{n}\sum_{i=1}^{n} T_i(N_t)$ was taken as a final output for a particular N_t. This procedure was applied repeatedly for various values of the maximum subgraph order N_c (defined as a number of core vertices in a subgraph) assumed for a decomposition: the system behavior was tested for $N_c \in \{5, 10, \ldots, 50, 100, 200, \ldots, 2700\}$.

Figure 8 presents results obtained for selected N_c values. As it could be expected for small subgraphs, the minimum time was reached for the number of threads equal to the number of processor's cores. With increasing N_c a number of concurrent calculating agents (threads) decreas so speedup effect is either weak or absent. It is visible as flattening of corresponding charts (e.g., for $N_c = 2700$). The relation between N_c and the average number of agents is shown in Figure 9. It may be seen that for some $500 \le N_c^{threshold} \le 600$ the number of agents falls below 8. It means that for the decomposition with $N_c > N_c^{threshold}$ not all processor's cores are used so the speedup effect will be limited to the actual number of active ones.

6. **Discussion.**

Let us compare formally the complexities of Incorporate operations in both cases, for RCG and slashed representations. In RCG approach each border node is shared by at least two and at last k graphs, where k is the total number of complementary graphs (maintaining agents). For this reason an operation of incorporating a single border node implies cooperation among k agents. Note that such a cooperation requires simultaneous locking border nodes belonging to k (in the worst case) RCGs and their neighbor vertices. Together it makes $\mathcal{O}(d \cdot k)$ nodes in the worst case, where d is a maximum degree of a node belonging to G. If any node cannot be locked then the entire operation fails. A locking considered above is required for granting an exclusive access to those nodes to an incorporating agent. From the statistical point of view that requirement may get difficult to satisfy with an increasing number of agents, k. On the other side retrying a procedure execution generates a message exchange overhead in an agent system.

In the case of a slashed representation of a graph, a completion of Incorporate requires cooperation with one agent only and locking at last p dummy nodes and q core edges where $p+q \leq d$ and d is a maximum degree of a node belonging to G. In the other words it is constant with respect to a number of agents.

For the RCG approach an estimated number of updating messages sent by an incorporating agent to other agents is $\mathcal{O}(d \cdot k^2)$. In the slashed representation case it equals $\mathcal{O}(d)$, i.e., it is constant with respect to a number of agents.

Estimations presented above show that using the slashed representation of G simplifies and improves efficiency of the Incorporate operation by reducing a number of cooperating agents from $\mathcal{O}(k)$ to 2, a number of locked nodes/edges from $\mathcal{O}(d \cdot k)$ to $\mathcal{O}(d)$ and a number of updating messages from $\mathcal{O}(d \cdot k^2)$ to $\mathcal{O}(d)$ compared with RCG model.

7. Conclusions.

Applicability of an agent system for distributed processing, in particular in massive lighting computations is strongly dependent on its efficiency. The first contribution to the reduction of the overall computing time is brought by the processing parallelization. The second factor strongly impacting the performance of a multi-agent system is the inter-agent cooperation.

The approach introduced in this paper allows reducing the complexity of distributed operations based on cooperation among agents, preserving the sufficient expressive power for a problem description. It is achieved by changing the conception of graph borders and thereby reducing a number of exchanged messages. Also the complexity of Incorporate procedure has been reduced significantly. Such an approach does not support the replication but in the case of the LaSIL problem this feature may be neglected.

Acknowledgment. The paper is supported from the resources of Alive & KIC-ing project.

REFERENCES

[1] S. Bandini et al., Self-organization models for adaptive environments: Envisioning and evaluation of alternative approaches, *Simulation Modelling Practice and Theory*, vol.18, no.10, pp.1483-1492, 2010.
[2] S. Baumgart et al., PLUG: An agent based prototype validation of CAD-constructions, *The International Conference on Information and Knowledge Engineering*, 2006.
[3] *Foundation for Intelligent Physical Agents (FIPA)*, http://www.fipa.org.
[4] M. Flasinski, On the parsing of deterministic graph languages for syntactic pattern recognition, *Pattern Recognition*, vol.26, no.1, pp.1-16, 1993.
[5] L. De Floriani and B. Falcidieno, A hierarchical boundary model for solid object representation, *ACM Trans. on Graph.*, vol.7, no.1, pp.42-60, 1988.
[6] *Java Agent Development Framework (JADE)*, http://jade.tilab.com.
[7] L. Kotulski, Supporting software agents by the graph transformation systems, *Lecture Notes Computer Science*, vol.3993, pp.887-890, 2006.
[8] L. Kotulski, GRADIS – Multiagent environment supporting distributed graph transformations, *Lecture Notes in Computer Science*, vol.5103, pp.644-653, 2008.

[9] L. Kotulski, On the control complementary graph replication, *Models and Methodology of System Dependability, Monographs of System Dependability*, vol.1, pp.83-95, 2010.

[10] L. Kotulski and B. Strug, Distributed adaptive design with hierarchical autonomous graph transformation systems, *Lecture Notes in Computer Science*, vol.4488, pp.880-887, 2007.

[11] L. Kotulski and A. Sędziwy, GRADIS – The multiagent environment supported by graph transformations, *Simulation Modelling Practice and Theory*, vol.18, no.10, pp.1515-1525, 2010.

[12] L. Xue et al., Multi-agent architecture for collaborative CAD system, *ICCSIT*, pp.7-11, 2008.

[13] G. Rozenberg, *Handbook of Graph Grammars and Computing by Graph Transformation: Vol.I, Foundations*, World Scientific Publishing Co., 1997.

[14] A. Sędziwy and L. Kotulski, Solving large-scale multipoint lighting design problem using multi-agent environment, *Key Engineering Materials*, vol.486, 2011.

[15] A. Sędziwy and M. Kozień-Woźniak, Computational support for optimizing street lighting design, *Complex Systems and Dependability, Advances in Intelligent and Soft Computing*, pp.241-255, 2012.

[16] T. Szmuc, L. Kotulski, B. Wojszczyk and A. Sędziwy, Green AGH campus, *Proc. of SMARTGREENS'12*, pp.159-162, 2012.

[17] J. Wąs, Crowd dynamics modeling in the light of proxemic theories, *Proc. of the 10th International Conference on Artifical Intelligence and Soft Computing: Part II, ICAISC'10*, pp.683-688, 2010.

[18] N. Yabuki et al., A cooperative design environment using multi-agents and virtual reality, *Cooperative Design, Visualization, and Engineering, Lecture Notes in Computer Science*, vol.3190, pp.96-103, 2004.