Graph Matching for Context Recognition[1]

Adrian Dobrescu
Computer Science Department
University Politehnica of Bucharest
313 Splaiul Independentei
060042 Bucharest, Romania
Email: marius.dobrescu@cti.pub.ro

Andrei Olaru
Computer Science Department
University Politehnica of Bucharest
313 Splaiul Independentei
060042 Bucharest, Romania
Email: cs@andreiolaru.ro

***Abstract*—In the software implementation of a general Ambient Intelligence (AmI) system, there are two major issues, on which depend the flexibility and the performance of the project. One is the implementation paradigm – how the various entities are organized and how they interact; the other is the management of context information, and how context-awareness is integrated as a first-class element in the implementation. This paper is framed in a research effort to develop an agent-based platform for AmI applications. While in previous research we have already argued in favor of using an agent-oriented paradigm for the implementation, and we have already introduced the concept of context graphs and context patterns, it is in this paper that we argue that matching context patters against context graphs is a valid method for detecting the user's situation and acting upon the user's context. In support of this, we analyze several algorithms for graph matching, adapted to our problem, and compare their performance on specific examples of context matching.**

I. Introduction

Ambient Intelligence – or AmI, for short – is the vision of a future where people will be surrounded by an electronic environment – consisting of a large number of sensors, actuators, smart appliances and devices – that sense the user's context and act in order to improve his or her experience. AmI consists of devices and services that are personalized for each user and that collaborate in order to be adaptive and anticipatory, taking action in a proactive but non-intrusive manner.

So far, a certain number of scenarios have been developed, that describe how Ambient Intelligence should act in a large, unified environment [1]–[3]. Implemented applications have focused on specific environments and situations (e.g. smart homes, various aspects of AAL, museum assistance) but few implementations deal with general platforms for AmI, and few use generic and flexible reasoning in their actions [4], [5].

This research is framed by a larger initiative (AmIciTy – Ambient Intelligence for the Collaborative Integration of Tasks) to build an agent-based environment for AmI applications in which context is a first-class element. At the core of the platform is the context-aware transfer of information between the agents, as well as a generic class of context-aware actions [6], [7]. This context-aware behavior of agents relies on the use of context graphs and context patterns – simple, flexible, yet powerful representations for the user's context and for known situations. Matching context patterns against the user's context graph, an agent can detect the user's situation and act accordingly, by detecting incomplete matches of context patterns in the context graph and suggesting missing edges [8]. The communication between agents is based on the principle that every agent sends the information that it deems interesting to the agents that share context with it and that may be interested in that piece of information [9]. "Interesting" here means that it matches a pattern from the agent's set of patterns. In fact, most of the generic, non application-specific activity of the context-aware agent is based on context matching (matching context patterns against the agent's context graph). It is used to detect if the information from other agents is relevant to the agent; it is used to detect the user's situation and propose possible actions; and it is used to detect information that may be interesting to neighbor (context-wise) agents. For implementation we use the agent-based tATAmI platform (towards Agent Technologies for Ambient Intelligence) [10].

Let us have a short example to show how this works, and see why context matching is important: Emily is an elderly woman that lives alone. Since she is old of age, she rarely goes out of her home, and when she does, it is usually to go shopping at the nearby store. At the store she needs her shopping bag and her wallet. She always needs to take her keys when going out. But she is forgetful and sometimes forgets one of the things, which is painful for her because she needs to go back. But since AmIciTy is here to help her, her clothes, shopping bag, wallet and keys all have RFID tags, and they can be traced by several detectors in the house. When her personal agent detects that she is near the door and dressed to go outside, therefore likely to leave, a pattern is activated that indicates that she needs to get her keys. Another pattern is activated saying that she may need her wallet and a bag that qualifies as a shopping bag. A non-intrusive reminder is activated. Moreover, if she has already taken, for instance, her wallet, the match with the second pattern is stronger and the alert should be stronger as well. A visual representation of these patterns can be seen in Figure 1.

While the argument about the using context graphs and

[1]The final publication can be found on IEEE Xplore
Dobrescu, A. and Olaru, A., *Graph Matching for Context Recognition* in Dumitrache, I., Florea, A. M. and Pop, F. (eds.), Proceedings of CSCS 19, the 19th International Conference on Control Systems and Computer Science, May 29-13, Bucharest, Romania, IEEE Xplore, 2013, pages 479-486
https://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6569308

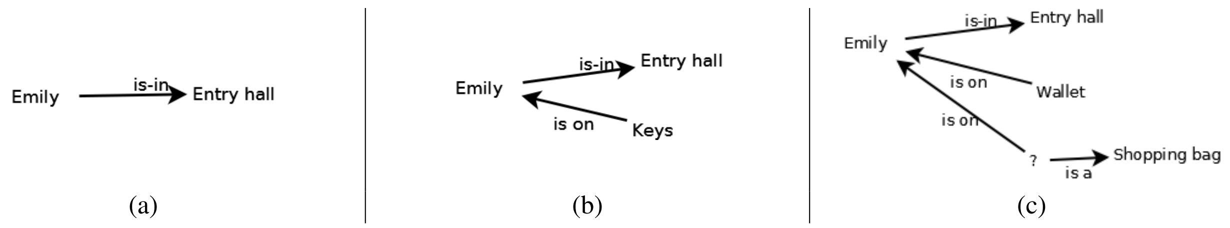


Fig. 1. Example context graph (a) and two context patterns (b, c).

patterns for AmI applications has already been done in our previous work, we cannot forget that a usable AmI system needs to have a certain degree of performance, in order to respond promptly to the user's needs and act with anticipation should the situation require it. The question is therefore how the matching of context patterns against graphs (context matching) can be done by agents, and if it can be done in adequate time by the whole range of agents (large and small) and with satisfactory results. This is the focus of this paper.

The general problem of graph matching, appearing usually in image recognition [11], has been well researched and several algorithms for matching graphs have been developed. However, the majority of them is focused on matching undirected, unlabeled graphs. An additional difficulty is that the general problem of graph matching is NP-hard.

In this context, the purpose of this paper is to review existing graph matching algorithms that are adequate for the problem at hand, to see how the algorithms should be modified in order to solve the problem, and what algorithms are able to offer adequate performance considering the constraints of Ambient Intelligence systems.

The following section presents some related work in the fields of context-awareness and graph matching. Section III details the concept of context graphs and patterns, to prepare the introduction of context matching algorithms in Section IV. Matching of edges labeled with regular expressions is detailed in Section V. Experimental results are given in Section VI and the last section draws the conclusions.

II. Related Work

A. Graphs and Patterns for Context-Awareness

In infrastructures for processing context information there are usually several layers that are proposed, going from sensors to the pre-processing of the perceived information, the layer for its storage and management, and finally to the application that uses the context information [12]. Infrastructures are also usually centralized, using context servers that are queried to obtain relevant or useful context information. In our approach [6], we attempt to build an agent-based infrastructure that is decentralized, in which each agent has knowledge about the context of its user, and the main aspect of context-awareness is based on associations between different pieces of context information. This makes the system able to use context-information that is not only consuming context information, but also creating context information and disseminating it through the system.

Modeling of context information uses representations that range from tuples to logical, case-based and ontological representations [13]. Henricksen et al use several types of associations as well as rule-based reasoning to take context-aware decisions [14]. Using graphs and patterns [8] leads to more flexibility and a more simple basic mechanism, that is more adequate to a constantly changing dynamic context.

B. Algorithms for Graph Matching

The idea of matching graphs has begun to gain momentum at the end of the 70s. In that time, it has been introduced as a powerful tool for solving pattern recognition (PR) problems [15]. After this period, the interest for using graphs in PR problems has lowered, due to the large computational effort needed to approach most of the algorithms used to determine similarities between graphs. This is because the graph matching problem is NP-complete.

However, recently the interest in using graphs for pattern recognition has grown back up [15]. Although the computational effort has remained significant, optimizations and most importantly much increased computational power enabled an approachable solution to matching graphs. This lead to using graphs in a wider range of domains. Graph matching is a process that can benefit applications in image (static and video) analysis, document processing, biometric analysis, and biomedical applications. Now, these domains include the one of Ambient Intelligence. Graphs can give a visual representation for agent networks, ontologies and other ways of knowledge representation (e.g. RDF).

We may classify graph matching algorithms in 2 major categories:

- Exact matching, when the structures must be identical;
- Inexact matching, when a match might be valid even if the 2 entities are different to a certain extent.

Among the most important algorithms for matching of unlabeled graphs are tree-search algorithms [16] and algorithms for the matching of a graph against a library of graphs [17]. Some algorithms, especially those for inexact matching [18], are based on powerful mathematical instruments – like expectation maximization [19], graduated assignment [20], and learning of assignment coefficients [21].

Ontology or schema alignment (or mapping), on the other hand, use labeled graphs in which nodes represent the elements of the schema and edges represent the relations. Most times it is nodes – i.e. concepts – that must be matched, in order to find the equivalence between concepts from different vocabularies or ontologies [22], [23]. While the fact that nodes and edges are attributed brings this type of matching closer to our work, the purpose of these algorithms is to match different vocabularies; our purpose is to match graph patterns against graphs using the same vocabulary, which in a way is closer to the purpose of image recognition.

The algorithms that we have focused on in this research are algorithms that can be adapted to the problem of context matching: they rely on label comparison and can be adapted to deal with generic edges and nodes. The algorithms presented use mainly two techniques:

- incremental matching by exploring the entire state space. (e.g. McGregor's algorithm [24]);
- the equivalence between finding a maximal clique and finding the maximum common subgraph (e.g. Bron-Kerbosch [25], Durand-Pasari [26], Akkoyunlu [27], Balas-Yu [28]);
- the equivalence with the maximal clique, but considering an extended modular product of the edges, not of the nodes (e.g. the Koch algorithm [29]).

There are other algorithms that perform graph matching very well but they are more suitable for particular graphs and those are hard to adapt to the context matching problem. Such an algorithm is Nauty, which is a special algorithm not related to more "traditional" graph matching algorithms [30].

We will further discuss graph matching algorithms and their adaptation to our particular problem in Section IV.

III. Using Graph Patterns to Detect Situation

In an Ambient Intelligence system, each agent should have a representation of the information that is interesting to it, and also the means of detecting what information is interesting to it from the stream of information that it receives. Moreover, it should have a representation of other agents' interests, in order to know whom to inform of potentially interesting information, out of all the agents that share some context with it.

Inside the agent, context information should be represented in a powerful, yet flexible manner, so that the same representation can be used on both capable and less capable devices. To support decentralization, agents should not rely strongly on centralized components, and must be able to use context information even in the lack of contact with the centralized components. An additional requirement is that agents should be able to easily aggregate information they receive, and that is interesting to them, with information already in their knowledge bases.

These are the reasons why we chose a graph-based, RDF-like representation for context information. Moreover, we introduced the notion of context patterns to define the interests of an agent and to help the agent detect the information that is relevant to its activity.

Each agent A has a *Context Graph* $CG_A = (V, E)$ that contains the information that is currently relevant to its function. Considering a global set of $Concepts$ and a global set of $Relations$, we have:

$CG_A = (V, E)$, where $V \subset Concepts$ and $E = \{edge(from,\ to,\ value) \mid ,\ from, to \in V,\ value \in Relations$.

The elements of $Concepts$ and $Relations$ are strings or URIs; $Relations$ also contains the empty string, for unnamed relations.

An agent also has a set of context $Patterns$:

$Patterns$ = $\{(G^P_s) \mid \ s \in PatternNames$, with G^P_s a graph pattern.

A context pattern s contains a graph $G^P_s = (V^P_s, E^P_s)$ and some other properties that are not relevant for the research question here (see our previous work for details on pattern relevance and persistence [6]). The graph has some special properties[1], i.e. can have question marks instead of vertex labels, and can have regular expressions as labels for edges.

A match i between a context pattern G^P_s and the context graph CG_A of an agent A is defined[2] as [8]:

$M_{A\text{-}si}(G'_A, G^P_m, G^P_x, f, k_f)$.

G'_A, G^P_m, G^P_x are graphs: $G'_A \subset CG_A$ is the subgraph matched by the pattern, $G^P_m = (V^P_m, E^P_m)$ is the part of the pattern that matches G'_A (or solved part), and $G^P_x = (V^P_x, E^P_x)$ is the rest of the pattern, which is unmatched. There is no intersection (common nodes or edges) between G^P_m and G^P_x.

The pattern G^P_s *matches* the subgraph $G'_A = (V', E')$, *iff* there exists an injective function $f : V^P_s \to V'$, so that

(1) $\forall v^P_i \in V^P_s, v^P_i = ?$ or $v^P_i = f(v^P_i)$ $(same\ value)$

and

(2) $\forall e^P \in E^P_s, e^P = (v^P_i, v^P_j, value)$ we have:

if $value$ is a string or an URI, then the edge $(f(v^P_i), f(v^P_j), value)$ is in E'

if $value$ is a regular expression (having $Relations$ as alphabet), then it matches the values $value_0$, $value_1$,... , $value_p$ of a series of edges $e_0, e_1, ..., e_p \in E'$, where $e_0 = (f(v^P_i),\ v_{a_0},\ value_0)$, $e_k = (v_{a_{k-1}},\ v_{a_k},\ value_1)$ $k = \overline{1, p-1}$, $e_p = (v_{a_{p-1}},\ f(v^P_j),\ value_p)$, $v_{a_l} \in V'$.

That is, very non-? vertex from the solved part matches (same label) a different vertex from G'_A, every non-*RegExp* edge from the solved part matches (same label for the edge and vertices) an edge from G'_A, and every *RegExp* edge from the solved part matches a series of edges from G'_A. Subgraph G' should be minimal.

A pattern G^P_s *k-matches* (matches except for *k* edges) a subgraph G' of G, if condition (2) above is fulfilled for $m-k$ edges in E^P_s, $k \in [1, m-1]$, $m = ||E^P_s||$ and G' remains connected and minimal.

For instance, in Figure 1, the first context pattern (Figure 1 (b)) 1-matches the context graph (Figure 1 (a)). The second pattern (Figure 1 (c)) only 3-matches the context graph, but with adequate values for characteristic and actionable, it is usable [8].

[1]We will mark with " ^{P} " graphs and elements that contain ? nodes, regular expressions, and other generic features.

[2]There may be multiple matches between the same pattern and the same graph.

IV. Adapting Algorithms for Graph Pattern Matching

While the problem of context matching does fall into the larger class of graph matching problems, it has a number of specific properties. First, most matching problems imply graphs with unlabeled nodes and edges. Context graphs and patterns are mostly labeled. Special cases are generic nodes and edges, which have unspecified labels, or (in the case of edges), are labeled with a regular expression.

The graph matching algorithms have a high complexity when the graphs contain multiple nodes with the same label. But in the context graph, it makes no sense to use an exponential algorithm when the nodes are unique. The problem can easily become one with multiple nodes with the same label, because of the generic nodes in the pattern, labeled with ? (in case they cannot be solved using a constraints satisfaction algorithm, which can directly deduce which are the labels that correspond to these nodes).

An appropriate solution would be based on backtracking, but not all algorithms permit this kind of labeling generic nodes on the fly. The other variant is expanding every node v_i, and replacing it with a set of nodes $v_j \ldots v_k$, where $v_j \ldots v_k \in CG_A$ and no nodes in G_s^P have the same label as $v_j \ldots v_k$.

For solving this type of graph, we can use algorithms that directly compare labels, such as McGregor's algorithm [24], or we can use algorithms that are based on the correspondence between the maximal clique in the associations graph and the largest common subgraph.

Another approach for this problem would be reducing the search space in the asymptotic limits given by the biggest clique in the chordal graph, derived from the initial graph (or triangulated, where every cycle of length greater than 4 has at least one chord). The dimension of this clique represents the inferior asymptote. The superior asymptote is given by the chromatic number of the graph [31].

A. The McGregor Algorithm

If we consider CG and G^P as the context graph and the context pattern, the McGregor algorithm [24] can be described as a representation is the state space. A state is defined as $S = (V, E, dim)$, where V is the set of all vertices and E the set of all edges in the current common subgraph; dim is the number of vertices, $dim = card(V)$. V and E represent a connected graph which is a subgraph of both CG and G^P. This subgraph can be a possible partial solution for the match. The solution to the global is, of course, the maximum common subgraph. The algorithm follows the following steps:

- *Step 1*: The initial state of the algorithm is empty (contains no vertices or edges, and the dimension is 0).
- *Step 2*: We choose a pair of vertices $(v1, v2) \in CG \times G^P$. A new node v' is created in V (the current state) with $label(v') = label(v1) = label(v2)$. If at least one edge $e'(v', v'')$ exists such that $v'' \in V$, the solution can be extended with v'.
- *Step 3*: If the solution is better than a previous maximum, then we keep this solution, being the best until the current step.
- *Step 4*: A new state is generated, corresponding to the current one, and a backtracking strategy is employed, with a depth first search, until we meet a leaf state (that we cannot expand anymore). In this moment, the previous state is restored and the new state is generated by expanding a valid vertex. If the leaf state is not useful anymore (i.e. maximal), it is discarded.

The complexity of the algorithm, in the worst case, is equal with the size of the cartesian product of the vertices in both graphs.

To adapt this algorithm to the context graphs problem, a few transformations are needed:

- *Step 1*: The nodes labeled with ? (generic nodes) are expanded. The number of vertices in the actual problem will grow, in the worst case, with the product between the number of unknown vertices and the number of vertices that are not present in the graph pattern.
- *Step 2*: The edges that do not have any correspondent in the contextual graph are removed. Same procedure is applied for the edges that lose their connectivity as a result of removing other edges, before. The complexity of this step is $O(E' \times E)$, where E is the set of the edges in the context graph and E' are the edges of the extended graph.
- *Step 3*: The resulting graph is verified again and only the maximal connected component is kept.
- *Step 4*: Apply the McGregor algorithm, presented above.

The advantage of this algorithm is in its simplicity. This approach is suitable for a whole range of applications, in which the agents recognize a relative small context (less than 15 independent nodes). For the majority of situations, this size is fair enough. In the worst case, the algorithm is exploring a big number of states equal with the cartesian product between vertices. If the size of the two graphs is growing or there are a bunch of vertices labeled with ?, the time of execution for this algorithm is very big and is not feasible for large scale applications.

B. The Larrosa algorithm

This algorithm was devised by Javier Larrosa and Gabriel Valiente [32]. It is a backtracking approach with constraint satisfaction.

A constraint satisfaction problem (CSP) is defined as an ordered set with N variables. An assignment of values for the variables is complete if it includes each variable. An assignment of values is consistent if it satisfies any constraint. A particular CSP can require an exhaustive generation of all solutions for this problem, a single solution or finding the best solution, given some circumstances.

In our case, we are interested in finding a single solution. This solution is found when all generic nodes have a correspondent in the graph. If we know what kind of labels are

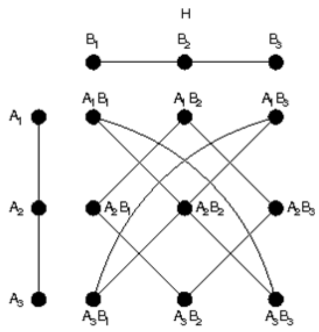


Fig. 2. Example of association graph obtained after applying the modular product [33].

present in the pattern graph, finding the maximum common subgraph is trivial because the vertices are unique and can be determined in polynomial time.

To adapt the Larrosa algorithm to our problem, the following difficulties need to be overcome:

The main disadvantage in being applied to context graphs is that this algorithm can find only a complete isomorphism. To translate the complete isomorphism between the two graphs in a partial isomorphism between any two subgraphs, we can verify if each new node that is added to the partial solution, generates the best solution to the present.

Another obstacle is that some vertices labeled with ? in the graph pattern have no correspondent. This detail has a very big impact on the backtracking mechanics because the validation of the neighbors cannot be done because they may not be a part of the solution.

For these reasons, the Larrosa algorithm cannot solve the context graph problem because we cannot take benefit of its advantages. Instead, this algorithm is very suitable for two types of individual subproblems – when all the generic nodes are included in the maximum common subgraph, and when the agent is allowed to identify a non-optimal solution, but that can be found very quickly.

C. The Bron-Kerbrosch Algorithm

The implementation of the Bron-kerbrosch algorithm [25] is reduced to finding a maximal clique in the associations graph [34], resulted by computing the modular product between the context graph and the pattern (see an example association graph in Figure 2). By applying the modular product, we are obtaining a graph $AG(AV, AE)$ with the following properties:

$AV = \{(u, v) | u \in CG, v \in G^P\}$

$AE = \{(v1, v2) | v1, v2 \in AV, compatiblenodes\}$

Two nodes $v1 = (u, v)$ and $v2 = (u', v')$ are adjacent *iff* 1) edges $(u, u', label) \in E$ and $(v, v', label) \in E^P$ have the same label, or 2) neither (u, u), nor (v, v) are adjacent.

The modifications needed to adapt this algorithm to our problem are:

- *Step 1*: CG and G^P are converted into undirected graphs such that any directed edge will become an undirected one.
- *Step 2*: The generic nodes in the pattern graph are expanded, in the same way as we did in the McGregor algorithm (Section IV-A. This step can also be done before *Step 1*, increasing the probability to remove edges that have the same label but other orientation.
- *Step 3*: The association graph is built using the modular product, as presented above.
- *Step 4*: The Bron-Kerbrosch algorithm is then applied in the association graph and the maximal cliques are identified.
- *Step 5*: A validation process takes place in which we filter the maximal cliques identified and we remove the edges that have the wrong orientation.

This algorithm works on 3 sets, R, P, and X. It finds the maximal cliques that include all of the vertices in R, some of the vertices in P, and none of the vertices in X. It can be upgraded through introduction of a pivot u from the set P. Any maximal clique must include u or his neighbors. Therefore, for a node v that is added to the set P, it is necessary only to test the node u and the nodes that aren't neighbors of u. Another approach would be to sort all the nodes in a convenient way. In the worst case, the complexity for the presented algorithm is $O(3^{n/3})$.

D. Other algorithms

There are some other algorithms which are variations of the ones in the previous sections, that we will present in short in this section.

The *Akkoyunlu* algorithm is a contemporary version [27] of the Bron-Kerbosch algorithm. It uses the optimizations of pivoting and vertex ordering and is presented in different terms, but it generates the same recursive search tree.

The *Durand-Pasari* algorithm [26] is also based on the equivalence between the maximal common subgraph and the maximal clique in the association graph. This approach is different than of Bron-Kerbosch in that it introduces supplementary validations on each step while adding vertices to the partial solution. Because the main approach is the same as of Bron-Kerbrosch and consists in finding cliques, the modifications needed are the same. The advantage is that the solution works like a stack (insertions and deletions are done only at one end), therefore we can optimize the space used to $O(card(V))$.

The *Balas-Yu* algorithm [28] relies as well on the equivalence between the maximal common subgraph and the maximal clique in the associations graph obtained from modular product. It uses a more complex heuristic for pruning the subtrees in the recursive search space. The heuristic is based on two concepts: triangulation of the graph, and coloring. A graph is considered to be triangulated if any cycle of more than 4 edges has at least one chord. The coloring procedure finds an association of colors to vertices in such a way that no two adjacent vertices have the same color. The algorithm's heuristic is based on the property that the size of the maximal clique is bounded by the largest triangulated subgraph and the minimum number of colors. This way, we can limit the size of the clique we are searching for.

The *Koch* algorithm [29] uses the equivalence between the maximal clique and the maximum common subgraph, but the modular product graph is being built by considering the edges and not the vertices. This approach might be better when the graphs have many vertices but fewer edges.

V. Using Regular Expressions

In order to make context patterns more powerful, it is allowed to have edges in the pattern labeled with regular expressions. This allows a single edge in the pattern to match a series of edges in the context graph (see Section III), edges whose labels match the regular expression (note that the regular expression has the values of the edges as alphabet).

For instance, in the pattern in Figure 1 (c), instead of the edge $? \xrightarrow{is\text{-}a} Shopping\ bag$[3], we could make the pattern closer to reality with specifying that something which can be used as a shopping bag is a bag that can, among other relations be used for shopping: $?(\xrightarrow{is\text{-}a^\star} Bag) \xrightarrow{is\text{-}a^\star for^+} Shopping.$

The algorithm adapted for using regular expressions follows the backtracking scheme of the Larrosa algorithm for generating the different ways to label the vertices. This doesn't benefit from the Larrosa optimizations because we want to find a partial match. On top of this procedure, we generate the paths that can match the regular expression.

However, including this feature in the matching of graphs greatly increases the difficulty of the problem. Some research has been done on this domain, from different points of view [35], [36]. However, since context patterns also contain generic (unknown) nodes, we cannot directly adapt a proposed solution. We have devised an algorithm with the following steps:

- Obtaining all the correct possible matching edges for an edge that is part of the pattern graph. If the edge has a simple label (not a regular expression), we should find all the edges in the context graph that have the same label and the vertices that represent the source and the destination also have to match. If one vertex is unknown, there may be multiple solutions for that edge, with each distinct label assigned to the vertices.
- If the edge is a regular expression then find all the possible paths between the source node and the destination node that match the label and also match the regular expression. (To optimize this step, all the paths from one node to another node that are visited in the recursive search, are being stored and reused in different traversals of the graph.)
- Choose one edge from the list or a path, instead, if the edge is labeled with a regular expression.
- Remove the chosen nodes from the available nodes.
- Continue this process until the solution cannot be extended anymore and test if the current one is better than a previous found solution.

[3]We use the following notation for graphs written in text, using labeled (if the edge is labeled) right arrows, parentheses and stars ("$\star$"): a graph with three nodes A, B and C and two edges, from A to B, and from B to C, is written as $A \to B \to C$; a tree with the root A having two children B and C is written as $A(\to B) \to C$.

The syntax of the regular expressions is flexible and is depending on the language used and the support for regular expressions.

VI. Experiments and Results

This section will present a view on the performances of the algorithms presented in the sections above. Benchmarks have been performed on all algorithms, trying to balance the examples between the best case and the worst case of a particular algorithm. It must be noted that the performance of the algorithms varies strongly depending on the number of generic nodes in the graph pattern. For instance, the McGregor algorithm, for a graph with 12 vertices out of which 2 nodes labeled with "?", the time of execution is a trivial one; instead, for 10 vertices labeled with "?", the time is increasing with an exponential growth factor. In this case the algorithm is not feasible for real applications anymore.

For the graphs below we have chosen a suite of tests such that it can be illustrate the situations where some algorithms are weak and other algorithms are strong. The tests have been designed to be similar with realistic situations. In the majority of tests, the pattern graph is almost half the size of the context graph. The values presented in the tables and graphs are computed as an arithmetic mean of the results of two or more executions on representative graphs.

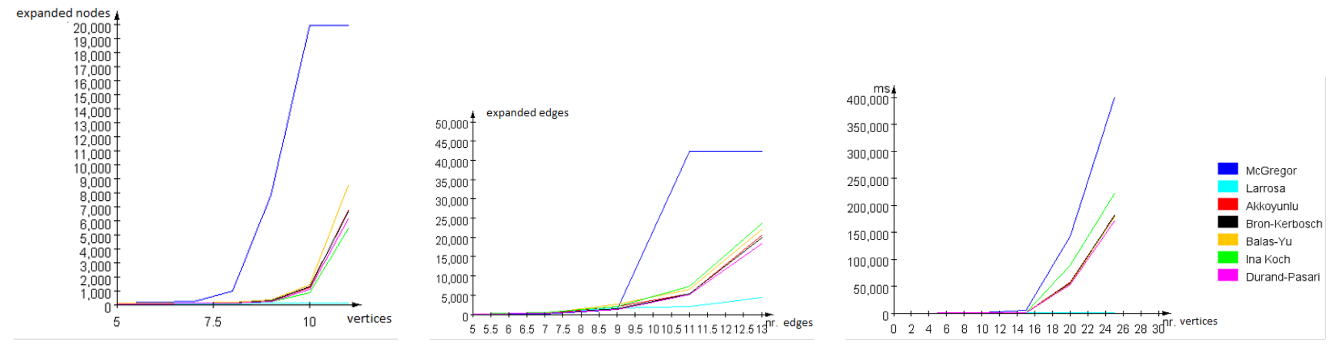
The results of the experiments – comparing the evolution of the implemented algorithms – are presented in Figure 3 (a)-(c) and Tables I, II and III.

Analysing the results after the execution of the algorithms, we can observe the following:

- The Larrosa algorithm has the best performances by far. This algorithm becomes infeasible only at around 40 nodes and performances begin to shrink. The only disadvantage of the Larrosa algorithm is that it cannot find partial solutions that are matching the two graphs.
- The McGregor algorithm, in the current approach has bad performances for a relatively small number of nodes. Its advantage consists, instead, in having a very easy implementation that is easy to adapt to our problem.
- Somewhere in between the two extremes is the Bron-Kerbrosch algorithm, together with its relatives. These algorithms are suited for our problem and have a less steep growth in complexity when the size of the graphs increases.

A few consideration on the complexity of graph matching (especially when using regular expressions) must be made. The problem of graph matching is NP-hard. While having most nodes and edges labeled is an advantage, generic / unknown nodes and regular expression edges add another layer of backtracking to matching algorithms.

Nevertheless, in advocating this approach for the implementation of context-awareness in AmI, we hold that adjusting



(a) (b) (c)

Fig. 3. Comparison of algorithm performance, as a function of a growing size of the graphs: the number of expanded nodes (a); the number of expanded edges (b); the execution time (c).

Vertices	McGregor	Larrosa	Akkoyunlu	Bron-Kerbosch	Balas-Yu	Koch	Durand-Pasari
5	124	10	51	31	72	30	32
6	130	15	47	45	84	38	41
7	223	43	105	91	122	71	84
8	992	56	134	123	183	101	115
9	7872	82	282	273	320	220	235
10	42100	105	1222	1255	1420	830	1100
11	524100	122	6723	6620	8522	5400	6113

TABLE I
COMPARISON OF THE NUMBER OF EXPANDED NODES.

Edges	McGregor	Larrosa	Akkoyunlu	Bron-Kerbosch	Balas-Yu	Koch	Durand-Pasari
5	42	46	124	120	135	140	119
7	115	391	330	321	372	420	310
9	1250	1699	1995	1464	2542	2115	1330
11	42343	2108	5431	5440	6423	7345	5219
13	1976312	4324	20470	19989	22170	23575	18322

TABLE II
COMPARISON OF THE NUMBER OF EXPANDED EDGES.

Vertices	McGregor	Larrosa	Akkoyunlu	Bron-Kerbosch	Balas-Yu	Koch	Durand-Pasari
5	14	1	9	7	23	10	9
10	82	11	22	34	42	36	30
15	5021	25	50	56	89	82	51
20	143023	36	56971	56392	54112	88523	52332
25	X	53	179434	181302	178342	221390	171000

TABLE III
COMPARISON OF THE EXECUTION TIME OF THE VARIOUS ALGORITHMS.

the size of context patterns and graphs to the computational power of the machine on which the agent is executing, the system can offer a good performance. Experiments show that for small graphs the algorithms offer a good execution time. Therefore, these sizes are adequate for agents running on small machines (small devices, sensors, Arduino boards, etc.), which is realistic, because smaller agents will have functions needing fewer information. In the whole system, agents will hold only the information which is relevant to their function.

VII. CONCLUSION

Implementing Ambient Intelligence applications is a complex task, due to the large number of elements that contribute to a successful AmI environment – hardware, coordination, power saving, knowledge representation, context-awareness, etc. In our research, we focus on building a context-aware multi-agent system that offers a generic layer of context recognition and context-aware action and information transfer. This layer has at its center context matching – matching context patterns against the agent's context graph.

Context matching is based on matching graphs, therefore in this paper we have analyzed several graph matching algorithms, detailing modifications that they needed for solving our specific problem, advantages and disadvantages that they have, and finally the experimental results of running the modified algorithm on context graphs of various sizes. We have identified three algorithms that are most likely candidates for the context matching problem.

As the problem of matching graphs is significantly difficult from the computational point of view, the greatest limitation of this approach lies within the reduced efficiency of matching algorithms, especially when the number of vertices in the graph increases beyond a certain (not very large) value.

As future work, our first priority is to optimize the matching to only match partial matches once, even if the context graph changes or new patterns are added. This would mean that matching can be done asynchronously and will be performed only when truly necessary.

Moreover, further testing needs to be performed to observe how the algorithms behave when varying the proportion of generic nodes in the patterns, the size of the pattern with respect to the size of the graph, and the complexity of regular expression edges. Further, context matching will be used in the simulation of realistic scenarios, under the constraints of devices of various sizes (e.g. Android smartphones).

References

[1] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J. Burgelman, "Scenarios for ambient intelligence in 2010," Office for Official Publications of the European Communities, Tech. Rep., February 2001.

[2] M. Satyanarayanan, "Pervasive computing: Vision and challenges," *IEEE Personal communications*, vol. 8, no. 4, pp. 10–17, 2001.

[3] G. Banavar and A. Bernstein, "Software infrastructure and design challenges for ubiquitous computing applications," *Communications of the ACM*, vol. 45, no. 12, pp. 92–96, 2002.

[4] D. Cook, J. Augusto, and V. Jakkula, "Ambient intelligence: Technologies, applications, and opportunities," *Pervasive and Mobile Computing*, vol. 5, no. 4, pp. 277–298, 2009.

[5] J. C. Augusto and P. J. McCullagh, "Ambient intelligence: Concepts and applications," *Computer Science and Information Systems (ComSIS)*, vol. 4, no. 1, pp. 1–27, 2007.

[6] A. Olaru, A. M. Florea, and A. El Fallah Seghrouchni, "A context-aware multi-agent system as a middleware for ambient intelligence," *Mobile Networks and Applications*, 2012. [Online]. Available: http://www.springerlink.com/content/6jj760k0774ju837

[7] A. Olaru, "A context-aware multi-agent system for AmI environments," Ph.D. Thesis, University Politehnica of Bucharest, Romania, University Pierre et Marie Curie (Paris 6), France, December 2011.

[8] A. Olaru, A. M. Florea, and A. El Fallah Seghrouchni, "Graphs and patterns for context-awareness," in *Ambient Intelligence - Software and Applications, 2nd International Symposium on Ambient Intelligence (ISAmI 2011), University of Salamanca (Spain) 6-8th April, 2011*, ser. Advances in Intelligent and Soft Computing, P. Novais, D. Preuveneers, and J. Corchado, Eds., vol. 92. Springer Berlin / Heidelberg, 2011, pp. 165–172. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-19937-0_21

[9] A. Olaru and C. Gratie, "Agent-based, context-aware information sharing for ambient intelligence," *International Journal on Artificial Intelligence Tools*, vol. 20, no. 6, pp. 985–1000, December 2011. [Online]. Available: http://www.worldscinet.com/ijait/20/2006/S0218213011000498.html

[10] V. Baljak, M. T. Benea, A. El Fallah Seghrouchni, C. Herpson, S. Honiden, T. T. N. Nguyen, A. Olaru, R. Shimizu, K. Tei, and S. Toriumi, "S-CLAIM: An agent-based programming language for AmI, a smart-room case study," in *Proceedings of ANT 2012, The 3rd International Conference on Ambient Systems, Networks and Technologies, August 27-29, Niagara Falls, Ontario, Canada*, ser. Procedia Computer Science, vol. 10. Elsevier, 2012, pp. 30–37. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050912003651

[11] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, "Face recognition: A literature survey," *Acm Computing Surveys (CSUR)*, vol. 35, no. 4, pp. 399–458, 2003.

[12] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263–277, 2007.

[13] M. Perttunen, J. Riekki, and O. Lassila, "Context representation and reasoning in pervasive computing: a review," *International Journal of Multimedia and Ubiquitous Engineering*, vol. 4, no. 4, pp. 1–28, October 2009.

[14] K. Henricksen and J. Indulska, "Developing context-aware pervasive computing applications: Models and approach," *Pervasive and Mobile Computing*, vol. 2, no. 1, pp. 37–64, 2006.

[15] D. Conte, P. Foggia, C. Sansone, and M. Vento, "Thirty years of graph matching in pattern recognition," *International journal of pattern recognition and artificial intelligence*, vol. 18, no. 3, pp. 265–298, 2004.

[16] L. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub) graph isomorphism algorithm for matching large graphs," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 10, pp. 1367–1372, 2004.

[17] B. Messmer and H. Bunke, "Efficient subgraph isomorphism detection: A decomposition approach," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 12, no. 2, pp. 307–323, 2000.

[18] E. Bengoetxea, P. Larrañaga, I. Bloch, A. Perchant, and C. Boeres, "Inexact graph matching by means of estimation of distribution algorithms," *Pattern Recognition*, vol. 35, no. 12, pp. 2867–2880, 2002.

[19] B. Luo and E. Hancock, "Structural graph matching using the EM algorithm and singular value decomposition," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, pp. 1120–1136, 2001.

[20] S. Gold and A. Rangarajan, "A graduated assignment algorithm for graph matching," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 18, no. 4, pp. 377–388, 1996.

[21] T. Caetano, J. McAuley, L. Cheng, Q. Le, and A. Smola, "Learning graph matching," *IEEE transactions on pattern analysis and machine intelligence*, pp. 1048–1058, 2009.

[22] D. Gašević and M. Hatala, "Searching web resources using ontology mappings," in *Integrating Ontologies Workshop Proceedings*. Citeseer, 2005, p. 33.

[23] W. Hu, N. Jian, Y. Qu, and Y. Wang, "GMO: A graph matching for ontologies," in *Integrating Ontologies Workshop Proceedings*. Citeseer, 2005, p. 41.

[24] J. J. McGregor, "Backtrack search algorithms and the maximal common subgraph problem," *Software: Practice and Experience*, vol. 12, no. 1, pp. 23–34, 1982.

[25] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[26] P. J. Durand, R. Pasari, J. W. Baker, and C.-c. Tsai, "An efficient algorithm for similarity analysis of molecules," *Internet Journal of Chemistry*, vol. 2, no. 17, pp. 1–16, 1999.

[27] E. Akkoyunlu, "The enumeration of maximal cliques of large graphs," *SIAM Journal on Computing*, vol. 2, no. 1, pp. 1–6, 1973.

[28] E. Balas and C. S. Yu, "Finding a maximum clique in an arbitrary graph," *SIAM Journal on Computing*, vol. 15, no. 4, pp. 1054–1068, 1986.

[29] I. Koch, "Enumerating all connected maximal common subgraphs in two graphs," *Theoretical Computer Science*, vol. 250, no. 1, pp. 1–30, 2001.

[30] B. D. McKay, "Practical graph isomorphism," *Congressus Numerantium*, vol. abs/1301.1493, no. 30, pp. 45–87, 1981.

[31] D. Conte, C. Guidobaldi, and C. Sansone, "A comparison of three maximum common subgraph algorithms on a large database of labeled graphs," *Graph Based Representations in Pattern Recognition*, pp. 589–607, 2003.

[32] J. Larrosa and G. Valiente, "Constraint satisfaction algorithms for graph pattern matching," *Mathematical structures in computer science*, vol. 12, no. 4, pp. 403–422, 2002.

[33] H. G. Barrow and R. M. Burstall, "Subgraph isomorphism, matching relational structures and maximal cliques," *Information Processing Letters*, vol. 4, no. 4, pp. 83–84, 1976.

[34] Y. Shinano, T. Fujie, Y. Ikebe, and R. Hirabayashi, "Solving the maximum clique problem using PUBB," in *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International... and Symposium on Parallel and Distributed Processing 1998*. IEEE, 1998, pp. 326–332.

[35] W. Fan, J. Li, S. Ma, N. Tang, and Y. Wu, "Adding regular expressions to graph reachability and pattern queries," *Frontiers of Computer Science*, vol. 6, no. 3, pp. 313–338, 2012.

[36] A. Jalali, A. Rensink, and A. H. Ghamarian, "Incremental pattern matching for regular expressions," *Electronic Communications of the EASST*, vol. 47, 2012.