

Graph Pattern Matching using Constraint Satisfaction

Javier Larrosa* and Gabriel Valiente**

Department of Software, Technical University of Catalonia, Barcelona, Spain,
valiente@lsi.upc.es

Abstract. Graph pattern matching is a central problem in many application fields. Since it is NP-complete, algorithms with a good worst-case performance cannot be expected to be found. However, there is still room for general procedures with a good average performance. Using the constraint satisfaction framework, a new algorithm is presented which is superior to previous approaches. The new algorithm relies on neighborhood constraints, a constraint not used before. It is shown theoretically that the new algorithm cannot do worse than previous approaches in terms of number of visited nodes. However, since it performs more work per node, this result does not ensure that the additional effort will pay off in practice. An additional contribution is the introduction of a new benchmark for testing algorithms in this domain. It is formed by a large set of well-defined graphs of very diverse nature. In this benchmark, the new algorithm also outperforms previous approaches, while still leaving many problem instances unsolved. The use of this challenging benchmark is encouraged for future algorithms evaluation.

1 Introduction

Graph pattern matching is the problem of finding an isomorphic image of a given graph in another graph, and it is also known as the subgraph isomorphism problem. It is a central problem in several fields, among which graph transformation. As a matter of fact, there is an increasing interest in the field of graph transformation in finding better subgraph isomorphism algorithms [2, 14, 17].

Since the subgraph isomorphism problem is NP-complete [5], all known algorithms have an exponential worst-case behavior. Thus, the development of a general algorithm able to solve medium and large problem instances within a practical amount of time is out of reach with current technology. There is, however, still room for algorithms that exhibit good average performance and with which particular problem instances can be solved in a reasonable amount of time. One way to assess these algorithms is by means of a benchmark. If the benchmark is formed by a large set of problem instances of different nature and one algorithm typically performs better than the rest, it can be expected that algorithm to outperform the others in a broader spectrum of problem instances.

* Partially supported by the Spanish CICYT project TIC96-0721-C02-02.

** Partially supported by the EC TMR Network GETGRATS and by the ESPRIT Basic Research Working Group APPLIGRAPH through the University of Bremen, by the Spanish DGES project PB96-0191-C02-02 and CICYT project TIC98-0949-C02-01 HEMOSS.

Constraint Reasoning is a flourishing field in Computer Science that provides a common framework for representing many combinatorial problems and a collection of techniques for efficiently solving them. In particular, graph pattern matching can be easily expressed as a constraint satisfaction problem.

In this paper, two contributions are presented toward the development of practical, general algorithms for subgraph isomorphism. On the one hand, a new benchmark for testing subgraph isomorphism algorithms is proposed. On the other hand, a new algorithm is presented that improves previous approaches. Although the amount of work per node in the search space is higher with the new algorithm, however, experimental results show that it pays off and by large.

1.1 Preliminaries

A *constraint satisfaction problem* (CSP) is defined by an ordered set of n variables $X = (1, 2, \dots, n)$, a finite domain D_i of possible values for each variable i , and a set C of *constraints* among variables. A constraint R_{j_1, \dots, j_r} on the ordered set of variables (j_1, \dots, j_r) is a subset of $D_{j_1} \times \dots \times D_{j_r}$ which only contains the *allowed* combinations of values for variables j_1, \dots, j_r .

An assignment of values to variables is *complete* if it includes every variable in X . A (possibly incomplete) assignment is *consistent* if it satisfies every constraint it is involved with. A *solution* for a constraint satisfaction problem is a consistent and complete assignment (that is, it satisfies every constraint).

Typical tasks of interest in constraint satisfaction are: finding one solution, finding all solutions, and finding the best solution under some preference criterion. In this paper, the problem of finding one solution is dealt with (although the new algorithm can also be used to solve the task of finding all solutions).

Most algorithms for constraint satisfaction problems start with an empty (trivially consistent) assignment and attempt to extend it by adding one variable at a time, while keeping the assignment consistent. When no extension is possible, the algorithm backtracks and changes a previous decision. The algorithm stops when a total assignment has been computed, or when the whole search space has been unsuccessfully traversed. During search, assigned variables are called *past* variables, unassigned variables are called *future* variables, and the variable under consideration is called the *current* variable.

It is well known in the constraint satisfaction area that *look-ahead* algorithms are the best choice for non-trivial problems. Each time the current variable is assigned, these algorithms remove values from future variable domains that cannot participate in a solution extending the current assignment. Surviving values are often called *feasible*.

An isomorphism of a graph $G_1 = (V_1, E_1)$ with a subgraph of a graph $G_2 = (V_2, E_2)$ is equivalent to the following constraint satisfaction problem. A variable i is associated with each vertex $v_i \in V_1$, and all variables take values on domain V_2 . Let n be the cardinality of V_1 . Finding a subgraph isomorphism is then equivalent to finding a complete assignment satisfying the following *structure constraint*:

$$R_{i,j} = \{(v_a, v_b) \in V_2 \times V_2 \mid v_a \neq v_b \wedge \text{edge}(G_1, i, j) \implies \text{edge}(G_2, v_a, v_b)\}$$

for all $i, j = 1, \dots, n$ with $i \neq j$.

For the sake of clarity, the rest of this paper deals with undirected graphs only, although the concepts and methods can be extended in a straightforward way to directed graphs. Experimental results for directed graphs are available from the authors.

1.2 Related work

Usually, [16] is considered to be the first algorithmic approach to the subgraph isomorphism problem. Although presented as a domain-specific algorithm, the method used therein can be described in terms of general constraint satisfaction techniques. His algorithm was later coined as *Really Full Look-Ahead* (RFLA) [12] in the constraint satisfaction community, a backtracking procedure that enforces *arc consistency* at each subproblem (arc consistency is attained when for every variable each domain value has at least one permitted value in every other variable). The filtering technique used for maintaining arc consistency was AC-1 [10], the only one available at that time. In this work, the following *degree constraint* was added to the CSP formulation:

$$R_i = \{v_a \in V_2 \mid \deg(G_1, i) \leq \deg(G_2, v_a)\}$$

for all $i = 1, \dots, n$. This unary constraint expresses the obvious necessary condition for a vertex (variable) of G_1 being *mappable* to a vertex (value) of G_2 . Such constraints are used in a preprocessing step in which values not satisfying the constraint are pruned, thus reducing the search space.

The first formulation of subgraph isomorphism as a constraint satisfaction problem dates back to [11]. The algorithm proposed in this work was later coined as Forward Checking (FC) [7]. Although RFLA requires to visit lesser nodes than FC, empirical results indicated that FC outperforms RFLA in terms of CPU time.

In a more recent work [14], unary constraints on the attributes of vertices and arcs are dealt with in a preprocessing step (enforcing node consistency) and structure constraints are dealt with using Back Jumping (BJ) [4]. It is known, however, that BJ cannot improve on FC [9], since FC never visits more nodes than BJ.

Finally, [13] introduced the *all-diff* constraint, a single n -ary constraint expressing that the value assigned to each variable must be different from any other value. He showed that this constraint is stronger than expressing the same with binary constraints, and proposed an algorithm to exploit that fact.

2 A New Algorithm for Subgraph Isomorphism

The new algorithm is based on the addition of the *neighborhood constraint*, which expresses the fact that a vertex (variable) $i \in V_1$ can only be mapped to another vertex (value) $v_a \in V_2$ if all vertices in the neighborhood of i can be mapped to some vertices in the neighborhood of v_a .

The neighborhood constraint is defined as follows:

$$R_{i,p,\dots,q} = \left\{ \begin{array}{l} (v_{a_1}, \dots, v_{a_r}) \in V_2 \times \dots \times V_2 \mid \\ v_{a_k} \neq v_{a_l} \quad \forall k, l = 1, \dots, r \quad (k \neq l) \\ \wedge \text{edge}(G_2, v_{a_1}, v_{a_k}) \quad \forall k = 2, \dots, r \end{array} \right\}$$

where $\{p, \dots, q\}$ is the neighborhood of vertex i in graph G_1 .

Fig. 1 outlines the algorithm. Procedure *solve* receives as parameters the set of future variables and their current domains, and it has to be initially called with V_1 as the set of variables and V_2 as the domain of each variable. First of all, neighborhood constraints are used to prune values that cannot satisfy them. If during this process a domain has become empty, there is no possible extension and the algorithm backtracks. Otherwise, a future variable is selected and its domain values are sequentially attempted. For each value, a forward checking-like look-ahead is performed using structure constraints. If the look-ahead does not cause an empty domain, the solver is recursively called with the current subproblem. The *propagate neighborhood constraints* procedure removes value v_a of variable i if it cannot satisfy neighborhood constraint $R_{i,p,\dots,q}$. Since the removal of one value may produce that another value becomes unfeasible, this filtering procedure is repeated until a fixed point is reached. Notice that by removing the call to this procedure, the algorithm coincides with FC.

```

procedure solve ( $X, D$ )
  if  $X$  is empty then a solution to the problem has been found
  else
     $D' \leftarrow$  propagate neighborhood constraints ( $X, D$ )
    if all domains in  $D'$  are not empty then
      select a variable  $i$  from  $X$ 
      for all  $v_a$  in  $D_i$  do
         $D'' \leftarrow$  look ahead ( $i, v_a, D'$ )
        if all domains in  $D''$  are not empty then solve ( $X - \{i\}, D''$ )
procedure propagate neighborhood constraints ( $X, D$ )
   $changes \leftarrow true$ 
  while  $changes$  do
     $changes \leftarrow false$ 
    for all  $i$  in  $X$  do
      for all  $v_a$  in  $D_i$  do
        let  $p, \dots, q$  be the neighborhood of  $i$  in  $X$ 
        if  $(v_{a_1}, \dots, v_{a_r}) \notin R_{i,p,\dots,q}$  for all  $v_{a_1}, \dots, v_{a_r} \in D_p \times \dots \times D_q$  then
           $D_i \leftarrow D_i - \{v_{a_1}\}$ 
           $changes \leftarrow true$ 
  return  $D$ 

```

Fig. 1. Outline of the new algorithm for subgraph isomorphism.

The new algorithm is more efficient than the algorithms described in [11, 16] for the subgraph isomorphism problem, because it prunes more values and thus visits lesser nodes of the search space, as established by the following result.

Proposition 1. *The new algorithm for subgraph isomorphism never visits more nodes than RFLA and than FC using degree constraints and structure constraints.*

Proof. It has to be shown that the new algorithm never keeps values that RFLA or FC would have removed. Since RFLA has a stronger pruning procedure than FC, it only has to be shown with respect to RFLA.

In the following, it will be shown that if value v_a of future variable i remains in the current domain D_i with the new algorithm at an arbitrary search node, it would also remain in the domain if RFLA had visited that node. Let P and F be the set of past and future variables, respectively. From the algorithmic description of RFLA¹ and the definition of degree and structure constraints, it is known that value v_a of variable i remains in D_i if and only if:

- a) $\deg(G_1, i) \leq \deg(G_2, v_a)$ (due to the initial enforcement of node consistency),
- b) $\forall j \in P, \text{edge}(G_1, j, i) \implies \text{edge}(G_2, v_j, v_a)$, being v_j the value assigned to j (due to the propagation of each assignment), and
- c) $\forall j \in F, \exists v_j \in D_j$ such that $\text{edge}(G_1, j, i) \implies \text{edge}(G_2, v_j, v_a)$ (due to the enforcement of arc consistency).

With the new algorithm, if a value remains feasible, then it also satisfies these three conditions. They are shown one after the other:

1. If value $v_a \in D_i$ does not satisfy condition (a) it will be pruned the first time *propagate neighborhood constraints* is called, because a set of different values in the neighborhood of v_a cannot be found that be large enough as to map into the set of variables in the neighborhood of i .
2. Those values not satisfying condition (b) will be pruned during the look-ahead process, because the look-ahead process in the new algorithm and in RFLA coincide.
3. If value $v_a \in D_i$ stops satisfying condition (c) during search, it will be pruned in the next call to *propagate neighborhood constraints*, because since there is a variable in the neighborhood of i not having any value in the neighborhood of v_a , $R_{i,p,\dots,q}$ cannot be satisfied.

□

3 A Benchmark for Subgraph Isomorphism

The experimental comparison of algorithms for subgraph isomorphism, as it is found for instance in [1, 3, 6, 11], has been often based on the results reported in [16]. The experiments described therein used both random and highly regular graphs. However, a very particular model of random graphs was used and the given description does not allow reproducing the experiments.

As a matter of fact, no benchmark has been developed yet for the subgraph isomorphism problem. In order to facilitate experimental comparison of different approaches and algorithms for subgraph isomorphism, a benchmark is proposed in this paper that is based on the Stanford GraphBase.

The Stanford GraphBase [8] provides a large library of well-defined graphs, comprising both irregular graphs derived from cultural artifacts and highly regular graphs,

¹ For a detailed description of RFLA and arc consistency, the reader is referred to [15].

for comparing combinatorial algorithms and for evaluating methods of combinatorial computing. It presents the advantages of being stable, machine-independent, and freely available. A machine-independent random number generator is included in the Stanford GraphBase, providing thus a way to construct random graphs that can be reconstructed elsewhere.

The benchmark proposed in this paper consists of all pairs of graphs (G_1, G_2) generated using the example graph parameters from [8, App. C], with G_1 being not larger (in number of vertices) than G_2 . The undirected graphs range from 10 to 16796 vertices and from 0 to 328960 edges. Since there are 113 undirected graphs, there are $(113 \cdot 114)/2 = 6441$ such problem instances available.

The experiments reported in this paper have been focused to a subset of 1769 problem instances, namely all pairs (G_i, G_j) of undirected graphs from the benchmark with $3 \leq i < j \leq 62$. In these experiments, 234 instances have been found to have at least one solution and 1175 have been found to have no solution. The remaining 360 instances remain open. These undirected graphs range from 10 to 377 vertices and from 15 to 4950 edges.

4 Experimental Results

Table 1. Number of problem instances in which no solution has been found, in which a solution has been found, and that could not be solved within a fixed time limit.

	Forward checking	Neighborhood constraint
No solution found	851	1175
Solution found	183	234
Not solved	735	360

The set of 1769 proposed problem instances has been attacked with both FC and the new algorithm². Table 1 shows results for the number of subgraph isomorphism problem instances in which no solution has been found, in which a solution has been found, and which could not be solved within a fixed time limit of 300 seconds. It can be observed that the new algorithm can either find a solution or prove unsolvability in more problem instances than FC. As a matter of fact, the new algorithm has proved the unsolvability of 957 instances without visiting any node at all, that is, by means of the first propagation of neighborhood constraints.

The number of nodes visited and the execution time for these experiments, both for FC and for the new algorithm, are illustrated in Fig. 2. It shows the expected behavior: while the new algorithm requires to visit much lesser nodes, it requires a larger amount of time, even for the simpler instances.

² Both algorithms have been implemented in C, sharing code and data structures. The experiments have been performed on a Sun Ultra 1 running Solaris 5.5.1 at 143 MHz and with 32 MB of main memory.

5 Conclusion

A new algorithm for the subgraph isomorphism problem has been introduced in this paper. It uses a constraint satisfaction formulation of the problem and relies on the exploitation of neighborhood constraints for domain-filtering purposes. Neighborhood constraints, which express a necessary condition for a tuple being in a solution, have never been used before. It has been shown that the new algorithm will never visit more nodes than the method used in two previous approaches [11, 16].

There is no standard benchmark for graph pattern matching algorithms. In this work, the use of the Stanford GraphBase as a source of instances to empirically evaluate algorithms is suggested. In this benchmark, it is shown that the additional effort per node that the new algorithm makes pays off by large, because many more instances can be solved. Nevertheless, the benchmark has many problem instances that the new algorithm cannot solve (and do not seem to be easily solved with current technology). For this reason, other researchers are encouraged to use this benchmark to evaluate their algorithms.

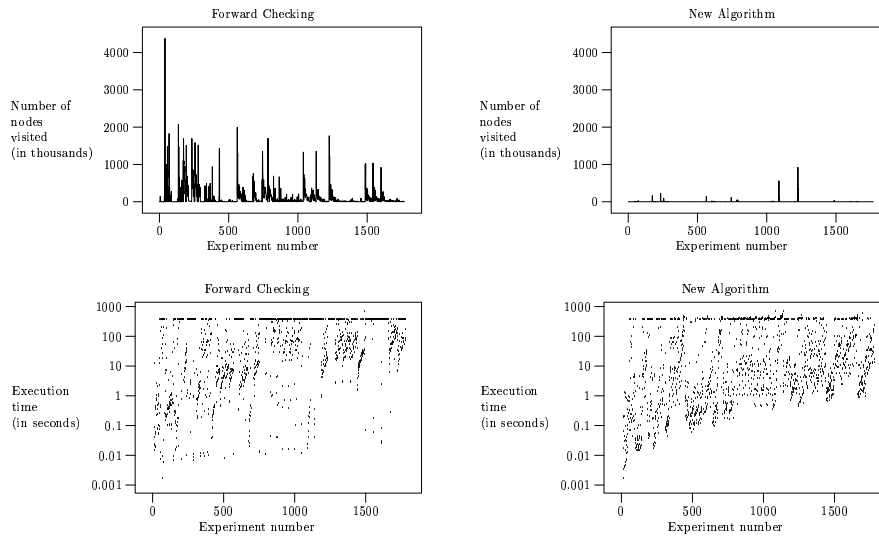


Fig. 2. Number of nodes visited and execution time for subgraph isomorphism experiments on undirected graphs.

References

1. F. A. Akinniyi, A. K. C. Wong, and D. A. Stacey. A new algorithm for graph monomorphism based on the projections of the product graph. *IEEE Trans. on Systems, Man, and Cybernetics*, 16(5):740–751, 1986.

2. H. Bunke, T. Glauser, and T.-H. Tran. An efficient implementation of graph grammars based on the RETE matching algorithm. In *Proc. 4th Int. Workshop on Graph-Grammars and their Application to Computer Science*, volume 532 of *Lecture Notes in Computer Science*, pages 174–189. Springer-Verlag, 1991.
3. J. K. Cheng and T. S. Huang. A subgraph isomorphism algorithm using resolution. *Pattern Recognition*, 13(5):371–379, 1981.
4. R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, 1990.
5. M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to NP-completeness*. Freeman, 1979.
6. D. E. Ghahraman, A. K. C. Wong, and T. Au. Graph monomorphism algorithms. *IEEE Trans. on Systems, Man, and Cybernetics*, 10(4):189–197, 1980.
7. R. M. Haralick and G. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263–313, 1980.
8. D. E. Knuth. *The Stanford GraphBase: A Platform for Combinatorial Computing*. ACM Press, 1993.
9. G. Kondrak and P. V. Beek. A theoretical evaluation of selected backtracking algorithms. *Artificial Intelligence*, 89(1–2):365–387, 1997.
10. A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.
11. J. J. McGregor. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Information Sciences*, 19:229–250, 1979.
12. B. Nudel. Tree search and arc consistency in constraint satisfaction algorithms. In L. N. Kanal and V. Kumar, editors, *Search in Artificial Intelligence*, pages 287–342. Springer-Verlag, 1988.
13. J.-C. Régin. A filtering algorithm for constraints of difference in constraint satisfaction problems. In *Proc. 12th Conf. American Assoc. Artificial Intelligence*, volume 1, pages 362–367, 1994.
14. M. Rudolf. Utilizing constraint satisfaction techniques for efficient graph pattern matching. In *Proc. 6th Int. Workshop on Theory and Application of Graph Transformation*, volume 1764 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000. To appear.
15. E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.
16. J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1):31–42, 1976.
17. A. Zündorf. Graph pattern matching in PROGRES. In *Proc. 5th Int. Workshop on Graph Grammars and their Application to Computer Science*, volume 1073 of *Lecture Notes in Computer Science*, pages 454–468. Springer-Verlag, 1996.