Computing and Drawing Isomorphic Subgraphs

Sabine Bachl[1] and Franz-Josef Brandenburg[2]

[1] sd&m AG, Thomas Dehler Str. 27, 81737 München, Germany
sabine.bachl@sdm.de
[2] University of Passau, 94030 Passau, Germany
brandenb@informatik.uni-passau.de

Abstract. The isomorphic subgraph problem is finding two disjoint subgraphs of a graph which coincide on at least k edges. Then the graph partitions into a large subgraph, its copy and a remainder. The problem resembles the NP-hard largest common subgraph problem. In [1,2] it has been shown that the isomorphic subgraph problem is NP-hard, even for restricted instances. In this paper we present a greedy heuristic for the approximation of large isomorphic subgraphs and introduce a spring algorithm which preserves isomorphic subgraphs and displays them as copies of each other. The heuristic has been tested extensively on four independent test suites. The drawing algorithm yields nice drawings which cannot be obtained by standard spring algorithms.

1 Introduction

Graph drawing is concerned with the problem of displaying graphs nicely. There is a wide spectrum of approaches and algorithms [4]. Nice drawings help in understanding the structural relation modelled by the graph. Bad drawings are misleading. This has been evaluated by HCI experiments [25,26], which have shown that symmetry is an important factor in the understanding of graph drawings and the underlying relational structure. Drawings of graphs in textbooks [27] and also the winners of the annual Graph Drawing Competitions and the logos of the symposia on Graph Drawing are often symmetric. In [5] symmetry is used for nice drawings without defining nice.

Symmetry has two directions: geometric and graph theoretic. A graph has a geometric symmetry, if it has a drawing with a rotational or a reflectional invariant. This has first been studied by Manning [22,23], and is NP-hard for general graphs. For some subclasses of planar graphs there are polynomial time solutions [18,22]. As a pre-requisite such graphs need a non-trivial automorphism. This is the graph theoretic access to symmetry. It is less restrictive than the geometric symmetry, and is isomorphism complete. However, almost all graphs from applications have only a trivial automorphism. Hence, more flexibility is needed, and the notions of geometric symmetry and automorphism must be relaxed and generalized. This is achieved by restricting symmetry to subgraphs only. From another viewpoint there is the cut&copy operation, which first selects a portion of a graph, then creates a new copy thereof, attaches it to the original graph,

M.T. Goodrich and S.G. Kobourov (Eds.): GD 2002, LNCS 2528, pp. 74–85, 2002.

and connects the copy and the remainder by some edges. The reverse operation leads to the notion of isomorphic subgraphs, which has been investigated in [1,2]. The isomorphic subgraph problem is finding two large disjoint subgraphs of a given graph, such that one subgraph is a copy of the other.

The generalization from graph automorphism to isomorphic subgraphs parallels the generalization from graph isomorphism to largest common subgraph. The difference between these problems lies in the number of input graphs. The coincidence or similarity of the subgraphs is measured in the number of common edges, which must be 100% for the automorphism and isomorphism problems, and which may be less for the more flexible generalizations. A related approach on symmetry by Chen et al. [7] reduces a given graph to a subgraph with a geometric symmetry by node and edge deletions and by edge contractions.

The isomorphic subgraph problem and the largest common subgraph problem are NP-hard, even when restricted to forests, connected outerplanar graphs and 2-connected planar graphs [1,2,16]. Both problems are tractable, if the graphs have tree-width k and the graphs and the isomorphic subgraphs are k-connected [6]. In particular, isomorphic subtrees [1,2] and the largest common subtree of two rooted trees can be computed in linear time. For arbitrary graphs the isomorphic subgraph problem seems harder than the largest common subgraph problem. For the largest common subgraph problem we must find a matching between pairs of nodes which induce an isomorphism. For the isomorphic subgraph problem we must first partition the graph into two subgraphs and a remainder and then find the isomorphism preserving matching between the subgraphs. Recall that the graph isomorphism problem can be solved in linear time for almost all graphs [3]. The algorithm uses the degree of the nodes as an invariant and distinguishes nodes by their degree sequences. For subgraph isomorphism Ullmann's algorithm [31] is well-known and is often used in computational biology for the search of molecular structures. Another method uses Levi's transformation into a clique problem, which is also used for the largest common subgraph problem [19,20].

Our approach to the computation of large isomorphic subgraphs is a greedy heuristic. In fact, there is a collection of algorithms with different weighting parameters. So we capture both, the partition and the isomorphism problems. Our experiments are directed towards the appropriateness of the approach and the effectiveness of the chosen parameters. The measurements are based on more than 100.000 computations conducted by Bachl for her dissertation [2].

As a second step and an application we present a spring algorithm which preserves isomorphic subgraphs and displays them as copies of each other. The algorithm is an extension of the Fruchterman and Reingold approach [15]. Each node and its copy are treated identically by averaging the forces and moves. An additional force is used to separate the two copies of the subgraphs. With a proper setting of the parameters of the forces this gives rather pleasing pictures.

2 Isomorphic Subgraphs

We consider undirected graphs $G = (V, E)$ with a set of nodes V and a set of edges E. Edges are denoted by pairs $e = (u, v)$. The set of *neighbours* of a node v is denoted by $N(v)$. For convenience, we exclude self-loops and multiple edges. In applications graphs are generally attributed graphs with labels at the vertices and at the edges. This is discarded here, although labels may be a great help in the selection and association of nodes and edges for the isomorphic subgraph problem.

Definition 1 (node- and edge-induced subgraphs). *Let $G = (V, E)$ be a graph and let $V' \subseteq V$ and $E' \subseteq E$ be subsets of nodes and edges. The node-induced subgraph $G[V'] = (V', E'')$ consists of the nodes of V' and the edges of E between nodes of V' such that $E'' = E \cap V' \times V'$. The edge-induced subgraph $G[E'] = (V'', E')$ consists of the edges of E' and the incident nodes V'' such that $V'' = \{u, v \in V | (u, v) \in E'\}$.*

An *isomorphism* between two graphs is a bijection on the sets of nodes that preserves adjacency. A *common subgraph* of two graphs G_1 and G_2 is a graph H that is isomorphic to $G_1[E_1]$ and $G_2[E_2]$. More precisely, H is an edge-induced common subgraph of size k, where $k = |E_1|$ is the number of edges. Accordingly, one may consider large node-induced common subgraphs.

Definition 2 (isomorphic node-induced subgraph, INS).
Instance: A graph $G = (V, E)$ and an integer k.
Question: Does G contain two disjoint node-induced common subgraphs H_1 and H_2 with at least k edges?

Definition 3 (isomorphic edge-induced subgraph, IES).
Instance: A graph $G = (V, E)$ and an integer k.
Question: Does G contain two disjoint edge-induced common subgraphs H_1 and H_2 with at least k edges?

For INS and IES the graph G partitions into H_1, H_2, and a remainder R. Let V, V_1, V_2 and V_R denote the sets of nodes of G, H_1, H_2 and R, respectively. Then $V_1 \cap V_2 = \emptyset$ and V_R contains all nodes from $V \setminus (V_1 \cup V_2)$. In the node-induced case, INS, the remainder contains all edges between the nodes of H_1 and H_2 and the edges with at least one endnode in R. In the edge-induced case, IES, R may also contain some edges between nodes from V_1 or from V_2. IES is our most flexible version.

Notice that also in the node-induced case the size of the common subgraphs must be measured in terms of the common edges. Otherwise, the problem may become meaningless. As an example consider an $n \times m$ grid graph with n, m even. If V_1 consists of all even nodes in the odd rows and of all odd nodes in the even rows, and $V_2 = V \setminus V_1$, then V_1 and V_2 have maximal size. However, their induced subgraphs are discrete with the empty set of edges.

The isomorphic subgraph problems INS and IES have been investigated by Bachl [1,2]. These problems are NP-hard, in general. This has been shown by a directed reduction of 3-PARTITION. At present, there are no direct reductions between the isomorphic subgraph and the largest common subgraph problems, although these problems look similar.

When restricted to trees, the isomorphic subtree problem is solvable in linear time. However, the isomorphic subtree problem requires trees as common subgraphs of a tree. This is not the restriction of the isomorphic subgraph problem to a tree. If the isomorphic subgraphs of a tree may be disconnected, then these instances of INS and IES become NP-hard. These results are summarized in Theorem 1. In [7] there are quite similar results for a related problem.

Theorem 1. *The isomorphic subgraph problems INS and IES are NP-hard. These problems remain NP-hard, if the graph G is*

- *a tree and the subgraphs are forests* [6]
- *outerplanar and the subgraphs are connected* [1,2]
- *2-connected outerplanar and the subgraphs are not 2-connected* [6].

3 Computing Isomorphic Subgraphs

The isomorphic subgraph problem is NP-hard almost everywhere. The size of the largest isomorphic subgraphs can be regarded as a measure for the self-similarity of a graph in the same way as the size of the largest common subgraph is regarded as a measure for the similarity of two graphs.

The isomorphic subgraph problem is different from the isomorphism, subgraph isomorphism and largest common subgraph problems. There is only a single graph, and it is a major task to locate and separate the two isomorphic subgraphs. Moreover, graph invariants such as the degree and adjacencies of isomorphic nodes are not necessarily preserved, since edges may end in the remainder or in the other subgraph or may be discarded, if they belong to only one of the edge-induced isomorphic subgraphs.

Our approach is a greedy algorithm. The core are weighted graph parameters. If the algorithm has identified a pair of nodes as a copy of each other it finds new pairs among their neighbours. The correspondence between the neighbours is computed by a weighted bipartite matching. Let $G = (V, E)$ be the given graph, and suppose that the subgraphs H_1 and H_2 have been computed as copies of each other. Consider a pair of nodes (v_1, v_2) which are copies of each other. For each pair the following graph parameters are taken into account:

- $w_1 = degree(v_1) + degree(v_2)$
- $w_2 = |degree(v_1) - degree(v_2)|$
- $w_3 =$ the number of common neighbours
- $w_4 =$ the number of new neighbours of v_1 and v_2 which are not in $H_1 \cup H_2$
- $w_5 =$ the graph-theoretical distance between v_1 and v_2
- $w_6 =$ the number of new isomorphic edges in $(H_1 \cup v_1, H_2 \cup v_2)$.

```
initialize(P);
    while (P not empty) do
        (v1, v2) = next(P); delete (v1, v2) from P;
        N1 = new neighbours of v1;
        N2 = new neighbours of v2;
        M = optimal weighted bipartite matching over N1 and N2;
        forall_edges(u1, u2) of M) do
            if (G[V1 ∪ {u1}] is isomorphic to G[V2 ∪ {u2}]) then
                P = P ∪ {(u1, u2)};
                V1 = V1 ∪ {u1}; V2 = V2 ∪ {u2};
```

Fig. 1. Heuristic for isomorphic node-induced subgraphs.

These parameters can be combined arbitrarily to a weight $W(v_1, v_2)$. The weights w_2 and w_3 are taken negative. Let $w_0 = \sum w_i$.

The algorithm iteratively selects a pair of nodes (v_1, v_2) at random, which are identified by the isomorphism, and then matches their new neighbours in the bipartite graph with the sets of nodes $N(v_1)$ and $N(v_2)$. For each pair of neighbours (u_1, u_2) let $W(u_1, u_2)$ be the weight of the edge (u_1, u_2). Then compute an optimal weighted bipartite matching. The pairs of matched nodes are new pairs of identified nodes in V_1 and V_2, provided that they pass the isomorphism test. The optimal weighted bipartite matching is computed by the Kuhn-Munkres algorithm (Hungarian method) [8], with a running time of $O(n^2 m)$ in our implementation. This can be improved, but it does not really matter, since these graphs are generally small.

Figure 1 shows the algorithm for the computation of node-induced subgraphs. P consists of the pairs of nodes which have already been identified by the isomorphism. The isomorphism test is only local, since each time only one new node is appended to each subgraph. The matching edges are sorted by decreasing weights and are checked in this order. In the node-induced case, a matched pair of neighbours (w_1, w_2) is skipped, if their addition to the intermediate subgraphs does not yield isomorphic subgraphs. In the case of edge-induced isomorphic subgraphs, the isomorphism test adds all pairs of matched nodes and all common edges. For each pair of starting nodes the algorithm runs in linear time except for the Kuhn-Munkres algorithm, which is used as a subroutine on bipartite graphs, whose nodes are neighbours of single nodes.

In our experiments a single parameter w_i or the sum w_0 is taken as a weight. The unnormalized sum worked out because the parameters had similar values.

The initialization of the algorithm is a critical point. What is the best pair of nodes to start with? We ran our experiments with three cases:

- repeatedly for all $O(n^2)$ pairs of nodes
- with the best pair of nodes according to the weighting function and
- repeatedly for all pairs of nodes, whose weighting function is at least 90% of the weight of the best pair.

With repetitions the largest subgraphs are kept. Clearly, the all pairs version has a $O(n^2)$ higher running time than the single pair version. The 90% best weighted pairs is a good compromise between the ellapsed time and the size of the computed subgraphs.
The experiments were performed on four independent test suites [24].

- 2215 random graphs with a maximum of 10 nodes and 14 edges
- 243 graphs generated by hand and
- 1464 graphs selected from the Roma graph library [28]
- 740 dense graphs.

These graphs where chosen by the following reasoning. For each graph from the first test suite the optimal result has been computed by an exhaustive search algorithm. This is doable only for small graphs.

The graphs from the second test suite were constructed by taking a single graph from the Roma library, making a copy and adding some nodes and edges at random. The *threshold* is the size of the original graph. It is a good bound for the size of the edge-induced isomorphic subgraphs. The goal is to detect and reconstruct the original graph. For INS, the threshold is a weaker estimate because the few added edges may destroy node-induced subgraph isomorphism.

Next, 1464 graphs from the Roma graph library were selected and inspected for isomorphic subgraphs with 10 to 60 nodes. These graphs are sparse with $|E| \sim 1.3|V|$.

The experiments of the first three test suites give a uniform picture. The heuristic finds the optimal subgraphs in the first test suite. In the second test suite the original graphs and their copies are detected almost completely when starting from the 90% best weighted pairs of nodes, see Figure 2.

The weights w_2, w_3 and w_6 produce the best results. The total running time is in proportion to the number of pairs of starting nodes. For the 90% best weighted pairs of nodes it is less than two seconds for the largest graphs with 100 nodes and w_0, w_1, w_4 and w_5, 30 seconds for w_2 and 100 seconds for w_3 and w_6, since in the latter cases the algorithm iterates over many to almost all pairs of starting nodes. In summary, the difference of degree w_2 gives the best performance.

For the third test suite the heuristic detects very large isomorphic subgraphs. Almost all nodes are contained in one of the node- or edge-induced isomorphic subgraphs which are almost trees. This surprising result is due to the sparsity of the test graphs. Again weight w_2 performs best. [2] has the complete list of test results with more than 100 diagrams.

Finally, we ran the heuristic on dense graphs with the 90% best weighted pairs of starting nodes. The dense graphs were generated at random. They have $10, 15, \ldots, 50$ nodes and $25, 50, \ldots, 0.4 \cdot n^2$ edges.

For IES the heuristic finds isomorphic subgraphs which together contain almost all nodes and more than 1/3 of the edges. For INS each subgraph contains about 1/4 of the nodes and 1/20 of the edges, see Figure 3.

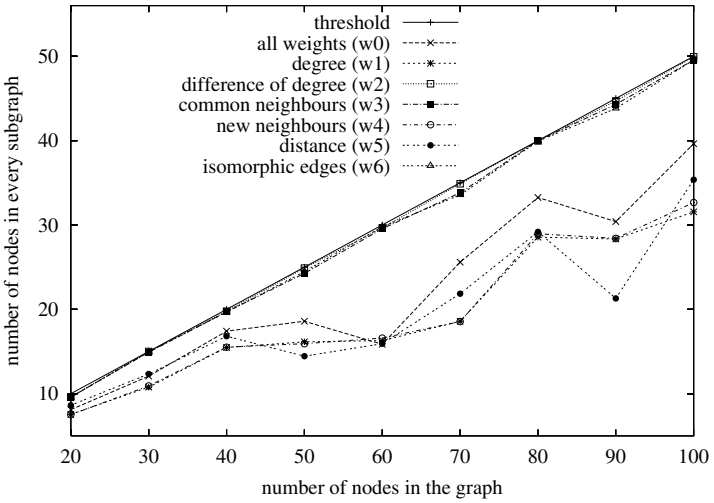

Fig. 2. Isomorphic edge-induced subgraphs of the second test suite for the 90% best weighted pairs of starting nodes.

These results are plausible, since for IES the heuristic adds pairs of matching nodes and ignores only isomorphism violating edges. For INS the isomorphic subgraphs turn out to be sparse.

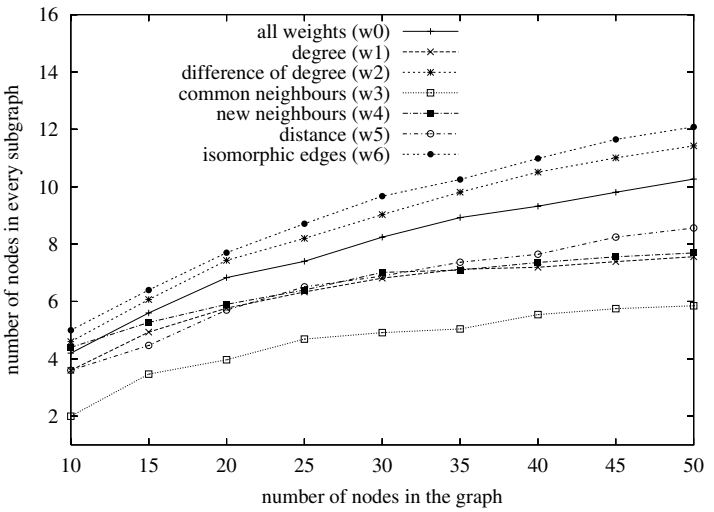

Fig. 3. Isomorphic node-induced subgraphs of dense graphs

The running time of one subgraph isomorphism detection from a single pair of nodes takes less than one second for our largest graphs. The total running time multiplies with the number of pairs of starting nodes, and is between two and 100 seconds for the largest graphs.

4 Drawing Isomorphic Subgraphs

In this section, we consider graphs with a pair of isomorphic subgraphs and their drawings. Sometimes, classical graph drawing algorithms preserve isomorphic subgraphs and display them symmetrically. The Reingold and Tilford algorithm [4,29] computes the tree drawing bottom-up, and so preserves isomorphic subtrees rooted at distinct nodes. It is readily seen that this is no more true for arbitrary subtrees. To stress this point, Supowit and Reingold [30] have introduced the notion of eumorphous tree drawings. However, eumorphous tree drawings of minimal width and integral coordinates are NP-hard. Also radial tree of ordered or embedded trees draw isomorphic rooted subtrees the same. The radial tree algorithm of Eades [10] squeezes a subtree into a sector with a wedge in proportion to the number of the leaves of the subtree. Hence, isomorphic subtrees get the same wedge and are drawn identically up to translation and rotation.

It is well-known that spring algorithms tend to display symmetric structures. This comes from the nature of spring methods, and has been emphasized by Eades and Lin [12]. For special classes of planar graphs there are efficient algorithms for the detection and display of symmetries of the whole graphs [18]. For hierarchical graphs (DAGs) symmetry preserving drawing algorithms are unknown.

This can be resolved by a *three-phase method*, which is generally applicable in this framework. If a graph G is partitioned into two node-induced isomorphic subgraphs H_1 and H_2 and a remainder R, first construct a drawing of H_1 and take a copy as a drawing of H_2. Then H_1 and H_2 are replaced by two large nodes and the remainder together with the two large nodes is drawn by some algorithm in phase two. This algorithm must take the area of the large nodes into account. And it should take care of the edges entering the graphs H_1 and H_2. Finally, the drawings of H_1 and H_2 are inserted at the large nodes and the edges to the nodes of H_1 and H_2 are locally routed on the area of the large nodes.

The drawback of this approach is the third phase, and the difficulties in routing edges to the nodes of H_1 and H_2. This may lead to bad drawings with node-edge crossings, parallel edges, and the destruction of symmetry.

The alternative to the three-phase method is an *integrated approach*. Here the preservation of isomorphic subgraphs is built into the drawing algorithms. As said before, this goes automatically for some tree drawing algorithms, and is solved here for a spring embedder. Spring algorithms have been introduced to graph drawing by Eades [11]. Fruchterman and Reingold [15] have simplifed the forces to improve the running time, which is a critical factor for spring algorithms. Our algorithm is based on the advanced USE algorithm [13], which

is included in the Graphlet system and is an extension of GEM [14]. In particular, USE supports individual distance parameters between any pair of nodes. Other spring algorithms allow only a uniform distance.

Our goal are identical drawings of isomorphic subgraphs. The input is an undirected graph G and two isomorphic subgraphs H_1 and H_2 of G together with the isomorphism ϕ from the nodes of H_1 to the nodes of H_2. If $v_2 = \phi(v_1)$ then v_2 is the copy of v_1.

In the initial phase, the nodes of H_1 and H_2 are placed identically on grid points (up to a translation). Thereafter the spring algorithm averages the forces imposed on each node and its copy and moves both simultaneously by the same vector. This guarantees that H_1 and H_2 are drawn identically.

Large isomorphic subgraph should be emphasized. They should be separated from each other and distinguished from the remainder. The distinction can be achieved by a node colouring. This does not help, if there is no geometric separation between H_1 and H_2, which can be enforced by imposing a stronger repelling force between the nodes of H_1 and H_2. We use three distance parameters, k_1 for the inner subgraph distance, k_2 between H_1 and H_2, and k_3 otherwise, which $k_1 < k_3 < k_2$. Additionaly, H_1 and H_2 are moved by move_subgraph(H, f). The force f is the sum of the forces acting on the subgraph H and an extra repelling force between the barycenters of H_1 and H_2.

In a round, all nodes of G are selected in random order and are moved according to the emanating forces. The algorithm stops, if a certain termination criterion is accomplished. This is a complex formula with a cooling schedule given by the USE algorithm, see [13,14]. The extra effort for the computation of forces between nodes and their copies leads to a slow down of the USE algorithm by a factor of about 1.5, if the symmetry of isomorphic subgraphs is enforced.

Input: a graph G and its partition into two isomorphic subgraphs H_1 and H_2, and a remainder R, and the isomorphism $\phi : H_1 \to H_2$
Initial_placement(H_1, H_2)
    while (termination is not yet accomplished) {
        forall_nodes_at_random $(v, V(G))$ {
            f = force(v);
            if $(v \in V(H_1) \cup V(H_2))$ then
                f' = force$(\phi(v))$;
                $f = \frac{f+f'}{2}$;
                move_node$(\phi[v], f)$;
            move_node(v, f);
        }
        move_subgraph(H_1, f_1); move_subgraph(H_2, f_2);
    }

Fig. 4. Isomorphism keeping spring embedder.

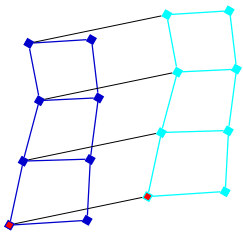

Fig. 5. 4×4-grid.

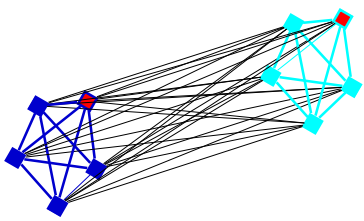

Fig. 6. Complete graph with 10 nodes.

Figure 4 describes the main steps of the isomorphism preserving spring algorithm. For each node v, *force(v)* is the sum of the forces acting on v and *move_node* applies the computed forces to a single node.

The examples shown below are computed by the heuristic and drawn by the spring algorithm. The most exciting drawings of graphs come from the second

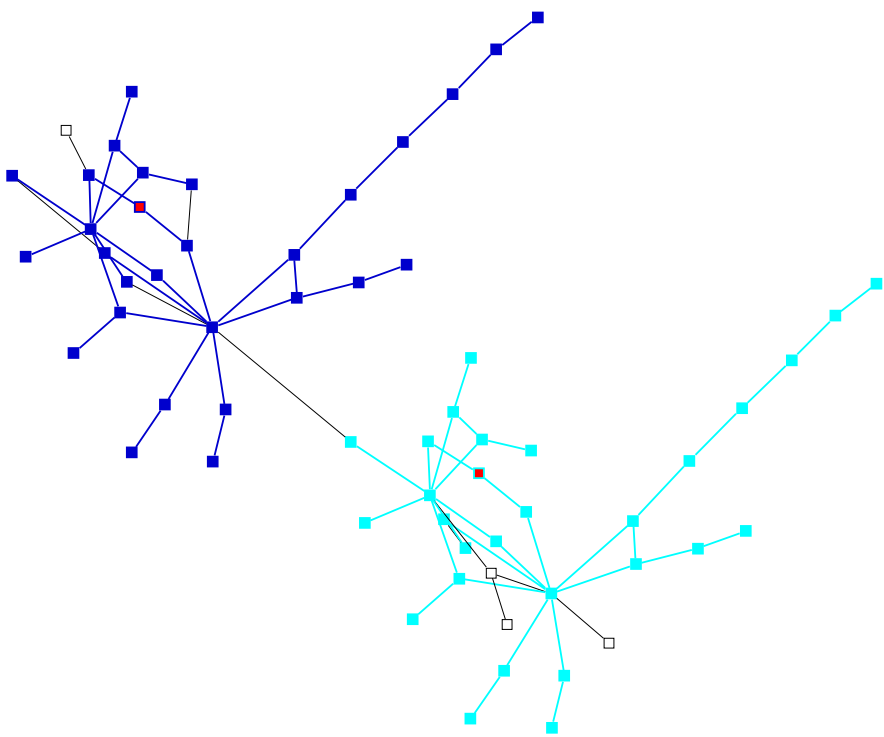

Fig. 7. Graph of test suite 2. The original graph with 30 vertices and 35 edges was copied and then 2 nodes and 4 edges were added. Our algorithm has re-computed the isomorphic edge-induced subgraphs. The open nodes belong to the remainder.

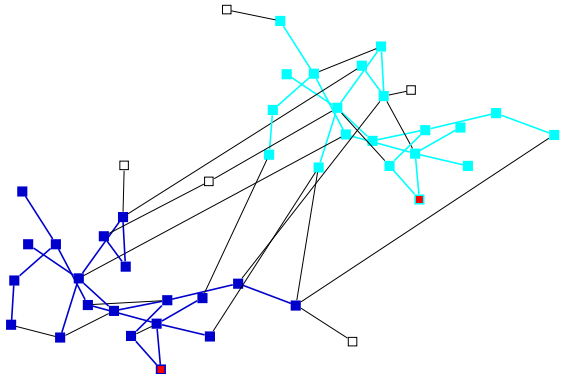

Fig. 8. Graph (number 06549 in [28]) of test suite 3 with 45 nodes, and the computed isomorphic edge-induced subgraphs.

and third test suites, see Figures 7 and 8. Many more drawings can be found in [2] or by using the algorithm in Graphlet.

The algorithm produces many nice drawings. However there is space for improvements. In Figure 5, for example, if the left subgraph were reflected the grid were unfolded. The isomorphic subgraphs must therefore be drawn identically up to translation and reflection. This is an extension of isomorphic subgraphs.

References

1. S. Bachl. Isomorphic subgraphs. Proc. Graph Drawing'99, LNCS 1731 (1999), 286-296.
2. S. Bachl. Erkennung isomorpher Subgraphen und deren Anwendung beim Zeichnen von Graphen. Dissertation, University of Passau, (2001), http://elib.ub.uni-passau.de/index.html.
3. L. Babai and L.Kucera. Canonical labelling of graphs in average linear time. Proc. 20th IEEE FOCS (1979), 39–46.
4. G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis. Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall, (1999).
5. T. Biedl, J. Marks, K. Ryall, and S. Whitesides. Graph multidrawing: Finding nice drawings without defining nice. Proc. Graph Drawing'98, LNCS 1547 (1998), 347-355.
6. F.J. Brandenburg. Pattern matching problems in graphs. Unpublished manuscript, (2000).
7. H.-L. Chen, H.-I. Lu and H.-C. Yen. On maximum symmetric subgraphs. Proc. Graph Drawing'00, LNCS 1984 (2001), 372–383.
8. J. Clark and D. Holton. Graphentheorie - Grundlagen und Anwendungen. Spektrum Akademischer Verlag, (1991).
9. D. Corneil and D. Kirkpatrick. A theoretical analysis of various heuristics for the graph isomorphism problem. SIAM J. Comput. 9, (1980), 281–297.
10. P. Eades. Drawing free trees. Bulletin of the Institute for Combinatorics and its Applications 5, (1992), 10–36.

11. P. Eades. A heuristic for graph drawing. Cong. Numer. 42, (1984), 149–160.
12. P. Eades and X. Lin. Spring algorithms and symmetry. Theoret. Comput. Sci. 240 (2000), 379–405.
13. M. Forster. Zeichnen ungerichteter Graphen mit gegebenen Knotengrößen durch ein Springembedder-Verfahren. Diplomarbeit, Universität Passau, (1999).
14. A. Frick, A. Ludwig and H. Mehldau. A fast adaptive layout algorithm for undirected graphs. Proc. Graph Drawing'94, LNCS 894 (1995), 388–403.
15. T. Fruchterman and E.M. Reingold. Graph drawing by force-directed placement. Software – Practice and Experience 21, (1991), 1129–1164.
16. M.R. Garey and D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, San Francisco, (1979).
17. A. Gupta and N. Nishimura. The complexity of subgraph isomorphism for classes of partial k-trees. Theoret. Comput. Sci. 164, (1996), 287–298.
18. S.-H. Hong, B. McKay and P. Eades. Symmetric drawings of triconnected planar graphs. Proc. 13 ACM-SIAM Symposium on Discrete Algorithms (2002), 356–365.
19. I. Koch. Enumerating all connected maximal common subgraphs in two graphs. Theoret. Comput. Sci. 250, (2001), 1–30.
20. G. Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. Calcolo 9, (1972), 341–352.
21. A. Lubiw. Some NP-complete problems similar to graph isomorphism. SIAM J. Comput. 10, (1981), 11–21.
22. J. Manning. Geometric symmetry in graphs. Ph.D. thesis, Purdue Univ., (1990).
23. J. Manning. Computational complexity of geometric symmetry detection in graphs. LNCS 507, (1990), 1–7.
24. Passau Test Suite. `http://www.infosun.uni-passau.de/br/isosubgraph`.
25. H. Purchase, R. Cohen and M. James. Validating graph drawing aesthetics. Proc. Graph Drawing'95, LNCS 1027 (1996), 435–446.
26. H. Purchase. Which aesthetic has the greatest effect on human understanding. Proc. Graph Drawing'97, LNCS 1353 (1997), 248–261.
27. R.C. Read and R.J. Wilson An Atlas of Graphs. Clarendon Press Oxford (1998)
28. Roma Graph Library. `http://www.inf.uniroma3.it/people/gdb/wp12/LOG.html`.
29. E.M. Reingold and J.S. Tilford. Tidier drawings of trees. IEEE Trans. SE 7, (1981), 223-228.
30. K.J. Supowit and E.M. Reingold. The complexity of drawing trees nicely. Acta Informatica 18, (1983), 377-392.
31. J.R. Ullmann. An algorithm for subgraph isomorphism. J. Assoc. Comput. Mach. 16, (1970), 31–42.