Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 8, no. 2, pp. 215–238 (2004)

Computing and Drawing Isomorphic Subgraphs

S. Bachl

Batavis GmbH & Co. KG, 94036 Passau, Germany
Sabine.Bachl@batavis.de

*F.-J. Brandenburg**

University of Passau, 94030 Passau, Germany
brandenb@informatik.uni-passau.de

D. Gmach

Lehrstuhl für Informatik III, TU München, 85746 Garching, Germany
gmach@in.tum.de

Abstract

The isomorphic subgraph problem is finding two disjoint subgraphs of a graph which coincide on at least k edges. The graph is partitioned into a subgraph, its copy, and a remainder. The problem resembles the NP-hard largest common subgraph problem, which searches copies of a graph in a pair of graphs. In this paper we show that the isomorphic subgraph problem is NP-hard, even for restricted instances such as connected outerplanar graphs. Then we present two different heuristics for the computation of maximal connected isomorphic subgraphs. Both heuristics use weighting functions and have been tested on four independent test suites. Finally, we introduce a spring algorithm which preserves isomorphic subgraphs and displays them as copies of each other. The drawing algorithm yields nice drawings and emphasizes the isomorphic subgraphs.

Article Type	Communicated by	Submitted	Revised
regular paper	Xin He	April 2003	October 2004

The work by F.-J. Brandenburg was supported in part by the German Science Foundation (DFG), grant Br 835/9-1. Corresponding author: F.-J. Brandenburg.

1 Introduction

Graph drawing is concerned with the problem of displaying graphs nicely. There is a wide spectrum of approaches and algorithms [10]. Nice drawings help in understanding the structural relation modeled by the graph. Bad drawings are misleading. This has been evaluated by HCI experiments [31, 32], which have shown that symmetry is an important factor in the understanding of graph drawings and the underlying relational structure. There is an information theoretic foundation and explanation for this observation: symmetry increases the redundancy of the drawing and saves bits for the information theoretic representation of a graph. Drawings of graphs in textbooks [33] and also the winning entries of the annual Graph Drawing Competitions and the logos of the symposia on Graph Drawing are often symmetric. In [3] symmetry is used for nice drawings without defining nice.

Another application area for isomorphic subgraphs comes from theoretical biology. It is of great interest to compare proteins, which are complex structures consisting of several 10000 of atoms. Proteins are considered at different abstraction levels. This helps reducing the computational complexity and detecting hidden structural similarities, which are represented as repetitive structural subunits. Proteins can be modeled as undirected labeled graphs, and a pair of repetitive structural subunits corresponds to a subgraph and its isomorphic copy.

Symmetry has two directions: geometric and graph theoretic. A graph theoretic symmetry means a non-trivial graph automorphism. The graph automorphism problem has been investigated in depth by Lubiw [27]. She proved the NP-hardness of many versions. However, the general graph automorphism and graph isomorphism problems are famous open problems between polynomial time and NP-hardness [19, 27]. A graph has a geometric symmetry if it has a drawing with a rotational or a reflectional invariant. Geometric symmetries of graphs correspond to special automorphisms, which has been elaborated in [14]. Geometric symmetries have first been studied by Manning [28, 29], who has shown that the detection of such symmetries is NP-hard for general graphs. However, for planar graphs there are polynomial time solutions [23, 22, 24, 29]. Furthermore, there is a relaxed approach by Chen et al. [8] which reduces a given graph to a subgraph with a geometric symmetry by node and edge deletions and by edge contractions. Again this leads to NP-hard problems.

Most graphs from applications have only a trivial automorphism. And small changes at a graph may destroy an automorphism and the isomorphism of a pair of graphs. Thus graph isomorphism and graph automorphism are very restrictive. More flexibility is needed, relaxing and generalizing the notions of geometric symmetry and graph automorphism. This goal is achieved by the restriction to subgraphs. Our approach is the notion of isomorphic subgraphs, which has been introduced in [5] and has first 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. Thus a graph G partitions into $G = H_1 + H_2 + R$, where H_1 and H_2 are isomorphic subgraphs and R is the remainder. From another viewpoint one may create graphs using the cut&copy operation of a graph editor in the following way: select a portion of a graph, create a new copy thereof, and connect the copy to the original graph.

The generalization from graph automorphism to isomorphic subgraphs is related to the generalization from graph isomorphism to largest common subgraph. The difference lies in the number of input graphs. The coincidence or similarity of the subgraphs is measured in the number of edges of the common subgraphs, which must be 100% for the automorphism and isomorphism problems, and which may be less for the more flexible generalizations.

The isomorphic subgraph problem and the largest common subgraph problem are NP-hard, even when restricted to connected outerplanar graphs and to 2-connected planar graphs [1, 2, 19]. For 3-connected planar graphs the largest common subgraph problem remains NP-hard as a generalization of Hamiltonian circuit, whereas the complexity of the isomorphic subgraph problem is open in this case. Both problems are tractable, if the graphs have tree-width k and the graphs and the isomorphic subgraphs are k-connected [6], where k is some constant. 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. An instance of the largest common subgraph problem consists of a pair of graphs. The objective is a matching between pairs of nodes inducing an isomorphism, such that the matched subgraphs are maximal. For the isomorphic subgraph problem there is a single graph. Now the task is finding a partition into three components, H_1, H_2, and a remainder R, and an isomorphism preserving matching between the subgraphs H_1 and H_2. This is a hard problem, since both graph partition and graph isomorphism are NP-hard problems.

We approach the computation of maximal connected isomorphic subgraphs in two ways: First, the *matching approach* uses different weighting parameters for a local bipartite matching that determines the local extension of a pair of isomorphic subgraphs by pairs of nodes from the neighborhood of two isomorphic nodes. The computation is repeated with different seeds. This approach captures 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]. Second, the *square graph approach* uses Levi's transformation [26] of common adjacencies into a comparability graph. We transform a graph into its square graph, which is then searched for large edge weighted cliques by a heuristic. Both approaches are evaluated and compared on four test suites. These tests reveal versions with best weighting functions for each of the two approaches. To stabilize the results both algorithms must be applied repeatedly with different seeds. However, the computed common subgraphs are sparse, and are often trees with just a few additional edges. Between the two approaches there is no clear winner by our measurements. Concerning the size of the computed connected isomorphic subgraphs the matching outperforms the product graph method on sparse graphs and conversely on dense graphs. A major advantage of the matching approach over the product graph approach is that it operates directly on the given graph. The product graph approach operates on square graphs, whose size is quadratic in the size of the input graphs.

As an application in graph drawing 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 [18]. Each node and its copy are treated identically by averaging 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, which cannot be obtained by standard spring algorithms.

The paper is organized as follows: First we define the isomorphic subgraph problem. In Section 3 we show its NP-hardness, even for very restrictive cases, and address the problem of computing large isomorphic subgraphs. In Section 4 we introduce two different algorithms which have been evaluated and compared in several versions and on different test suites. Finally, in Section 5 we present a graph drawing algorithm for isomorphic subgraphs.

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 $e = (u, v)$. For convenience, self-loops and multiple edges are excluded. The set of *neighbors* of a node v is $N(v) = \{u \in V \mid (u, v) \in E\}$. Observe that a node is not its own neighbor. If v is a node from a distinguished subgraph H, then the *new neighbors* of v are its neighbors not in H.

In many applications there are node and edge labels and the labels have to be respected by an isomorphism. This is discarded here, although the labels are a great help in the selection and association of nodes and edges for the isomorphic subgraph problem. Often additional information is used for an initial seed before starting the comparison of two graphs. A good seed is an important factor, as our results shall confirm.

Definition 1 *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 $E' = E \cap V' \times V'$. The edge-induced subgraph $G[E''] = (V'', E'')$ consists of the edges of E'' and their incident nodes $V'' = \{u, v \in V \mid (u, v) \in E''\}$.*

An *isomorphism* between two graphs is a bijection $\phi : G \to G'$ on the sets of nodes that preserves adjacency. If $G = G'$, ϕ is a *graph automorphism*, which is a permutation of the set of nodes that preserves adjacency. If $v' = \phi(v)$, then v and v' are called a pair of *isomorphic copies*, and similarly for a pair of edges.

A *common subgraph* of two graphs G and G' is a graph H that is isomorphic to $G[E]$ and $G'[E']$ for some subsets of the sets of edges of G and G'. More precisely, H is an *edge-induced common subgraph* of size k, where k is the number of edges of H. Accordingly, H is a *node-induced common subgraph* if $G[E]$ and $G'[E']$ are node-induced subgraphs.

Our notions of isomorphic subgraphs use only single graphs.

Definition 2
The isomorphic node-induced subgraph problem, 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
The isomorphic edge-induced subgraph problem, 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 two isomorphic subgraphs H_1 and H_2 and a remainder R such that $G = H_1 + H_2 + R$. When computing isomorphic subgraphs the subgraphs H_1 and H_2 are supposed to be connected and maximal. IES is our most flexible version, whereas INS seems more appropriate for graph drawings. 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.

Common subgraphs can be represented as cliques in so-called comparability or product graphs. This has been proposed by Levi [26] for the computation of maximal common induced subgraphs of a pair of graphs. Then a common subgraph one-to-one corresponds to a clique in the comparability graph. We adapt this concept to isomorphic subgraphs of a single graph.

Definition 4 *The square of a graph $G = (V, E)$ is an undirected graph $G^2 = (V^2, E^2)$, whose nodes are the pairs of distinct nodes of G with $V^2 = \{(u, v) \mid u \neq v \text{ and } u, v \in V\}$. The edges of G^2 are generating or stabilizing edges between pairwise distinct nodes u_1, v_1, u_2, v_2. $((u_1, u_2), (v_1, v_2))$ is a generating edge, if both (u_1, v_1) and (u_2, v_2) are edges of E. $((u_1, u_2), (v_1, v_2))$ is a stabilizing edge, if there are no such edges in E. There is no edge in G^2, if there is exactly one edge between the respective nodes in G.*

Each pair of edges in G with pairwise distinct endnodes induces four generating edges in G^2, and accordingly there are four stabilizing edges resulting from two pairs of non-adjacent nodes. Hence, square graphs are dense and have an axial symmetry. In the later clique searching algorithm generating edges will receive a high weight of size $|V|^2$, and stabilizing edges are assigned the weight one.

Lemma 1 *The square graph of an undirected graph has an axial symmetry.*

Proof: Let $G = (V, E)$ and G^2 be the graph and its square. Use the Y-axis for the axial symmetry of G^2. For each pair of nodes $u, v \in V$ with $u \neq v$ there is a pair of symmetric nodes of G^2 such that (u, v) is placed at $(-x, y)$ and (v, u) is placed at $(+x, y)$ for some x, y, such that distinct nodes are placed at distinct positions. If v_1, v_2, v_3, v_4 are distinct nodes of V, then the edges induced by these nodes in G^2 are symmetric, mapping $((v_1, v_2), (v_3, v_4))$ to $((v_2, v_1), (v_4, v_3))$. □

The usability of square graphs for the computation of isomorphic subgraphs is based on the following fact.

Theorem 1 *Let V_1 and V_2 be subsets of the set of nodes V of a graph G. $H_1 = G[V_1]$ and $H_2 = G[V_2]$ are isomorphic disjoint node-induced subgraphs with an isomorphism $\phi : V_1 \to V_2$ if and only if the set of nodes $\{(v, \phi(v)) \mid v \in V_1\}$ induces a clique in the square graph G^2. The isomorphism ϕ can be retrieved from the clique.*

Proof: If ϕ is an isomorphism from H_1 into H_2, then for nodes $u, v \in V_1$ with $u \neq v$ the nodes $(u, \phi(u))$ and $(v, \phi(v))$ are connected in G^2 by a generating edge or by a stabilizing edge. Hence, the set of nodes $\{(v, \phi(v)) \mid v \in V_1\}$ induces a clique in G^2. Conversely, suppose that a set of nodes W of G^2 defines a clique. Let $V_1 = \{v_1 \mid (v_1, v_2) \in W\}$ and $V_2 = \{v_2 \mid (v_1, v_2) \in W\}$ be the projections onto the first and second components. If (u_1, u_2) and (v_1, v_2) are nodes of W, then the nodes u_1, v_1, u_2, v_2 are pairwise distinct. Hence $V_1 \cap V_2 = \emptyset$. Define $\phi : V_1 \to V_2$ by $\phi(v_1) = v_2$ if $(v_1, v_2) \in W$. From the distinctness ϕ is a bijection. Moreover, for nodes $u_1, v_1 \in V_1$ and their images $u_2, v_2 \in V_2$ either there are two edges (u_1, v_1) and (u_2, v_2) in G or these pairs of nodes are not adjacent. Hence, $G[V_1]$ and $G[V_2]$ are isomorphic disjoint node-induced subgraphs. □

Consequently, a maximal clique in the node square graph G^2 corresponds to a pair of maximal node-induced subgraphs in G. However, the isomorphic subgraphs of G are not necessarily connected. Connectivity can be established by the requirement that the clique nodes in G^2 are connected by generating edges.

Accordingly, we use so-called edge-square graphs for the edge-induced isomorphic subgraph problem.

Definition 5 *The edge-square graph of a graph $G = (V, E)$ consists of the pairs of edges of G with distinct endnodes $\{(e, e') \mid e, e' \in E$ and u, v, u', v' are pairwise distinct for $e = (u, v)$ and $e' = (u', v')\}$ as its nodes. Two such nodes (e, e') and (f, f') are connected by an undirected edge if and only if the endnodes of e, e' and of f, f' are pairwise distinct and either e, e' and f, f' are adjacent in G or e, e' and f, f' are not adjacent in G. In the first case, the edge between (e, e') and (f, f') is a generating edge, in the latter case a stabilizing edge.*

Notice that the edge-square graph of $G = (V, E)$ is the node square graph of its edge graph $G_e = (E, F)$ with $(e, e') \in F$ if and only if the edges e and e' are adjacent in G. Its size is quadratic in the number of edges of G.

3 Isomorphic Subgraph Problems

The largest common subgraph problem [19] generalizes many other NP-hard graph problems such as the clique, Hamiltonian circuit, and subgraph isomorphism problems. The isomorphic subgraph problem is similar; however, it uses only a single graph. The analogy between the largest common subgraph problem and the isomorphic subgraph problem is not universal, since graph isomorphism is a special case of the largest common subgraph problem, whereas graph automorphism is not a special case of the isomorphic subgraph problem. Nevertheless, the isomorphic subgraph problems are NP-hard, too, even for related specializations. However, at present we have no direct reductions between the isomorphic subgraph and the largest common subgraph problems.

When restricted to trees the isomorphic subtree problem is solvable in linear time. This has been established in [1, 2] for free and rooted, ordered and unordered trees by a transformation of trees into strings of pairs of parentheses and a reduction of the largest common subtree problem to a longest non-overlapping well-nested substring and subsequence problems. 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, where the isomorphic subgraphs may be disconnected. In [8] there are quite similar results for a related problem.

Theorem 2 *The isomorphic subgraph problems INS and IES are NP-hard. These problems remain NP-hard if the graph G is*

- *1-connected outerplanar and the subgraphs are or are not connected.*
- *2-connected planar and the subgraphs are 2-connected.*

Proof: We reduce from the strongly NP-hard 3-partition problem [19]. First consider IES.

Let $A = \{a_1, \ldots, a_{3m}\}$ and $B \in \mathbb{N}$ be an instance of 3-partition with a size $s(a)$ for each $a \in A$ and $B/4 < s(a) < B/2$. The elements of A must be grouped into m triples whose sizes sum up to B.

We construct a graph G as shown in Figure 1. The left subgraph L consists of $3m$ fans, where the i-th fan is a chain of $s(a_i)$ nodes which are all connected to v_1. It is called the $s(a_i)$-fan. The right subgraph R has m fans each consisting of a chain of B nodes which are all connected to v_2. The fans of R are called B-fans. Let $k = 2Bm - 3m$. Then G has $2k + 2m + 1$ edges.

Now G has an IES solution of size k if and only if there is a solution of 3-partition.

First, let $A_1, \ldots, A_m$ be a solution of 3-partition with each A_i containing three elements of A. Define $H_1 = L$ and $H_2 = R$ and a bijection $\phi : H_1 \to H_2$, such that v_1 is mapped to v_2 and for each $a \in A_j$ $(1 \leq j \leq m)$ the $s(a)$-fans are mapped to the j-th B-fan. This cuts two edges of each B-fan. Then H_1 and H_2 have $\sum_{a \in A} (2s(a) - 1) = 2Bm - 3m = k$ common edges and IES has a solution.

Conversely, let H_1 and H_2 be a solution of IES with $k = 2Bm - 3m$. Then one of the graphs must be L and the other is R, and they must be mapped

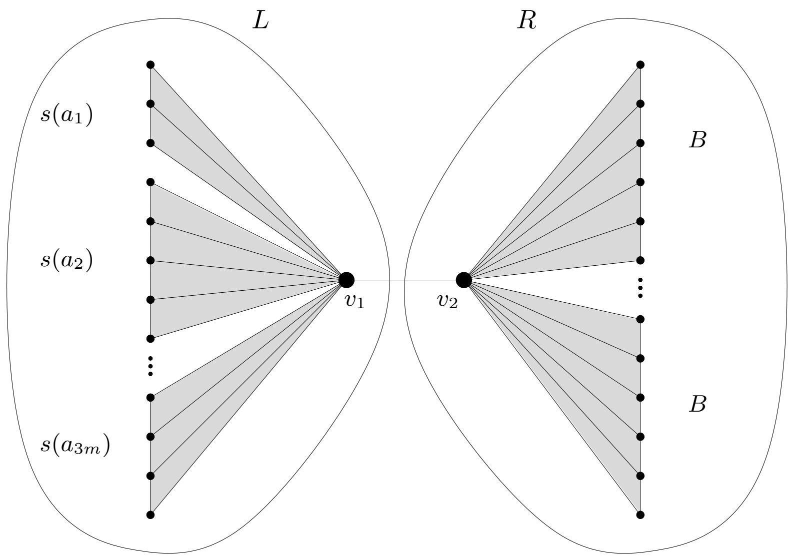

Figure 1: Reduction to IES for connected outerplanar graphs

to each other by ϕ, such that only $2m + 1$ edges go lost by the mapping. In particular, $\phi(v_1) = v_2$, since the degree of v_1 and v_2 is $Bm + 1$, whereas all other nodes have a degree of at most three. Hence, if $\phi(v_1) \neq v_2$, then at least $2 \cdot (Bm+1-3)$ edges go lost and at most $|E| - 2(Bm-2)$ edges are left for H_1 and H_2. Since $|E| = 4Bm - 4m + 1$ and $B > 1$, less than $k - 3$ edges are left for each of H_1 and H_2, contradicting the assumption $\phi(v_1) = v_2$. Let v_1 be in H_1 and v_2 in H_2. If the subgraphs must be connected, then H_1 is a subgraph of L and $H_1 = L$ to achieve the bound k, and similarly H_2 is a subgraph of R. This is also true, if H_1 and H_2 may be disconnected. To see this suppose the contrary. If $j > 0$ edges of R are in H_1, then at least $j + 1$ nodes of R are in H_1. Each such node decreases the number of edges for the isomorphic copies at least by one, since the edges to v_2 go lost. And for every three $s(a_i)$-fans from L at least two edges from R go lost. Since R has $k + 2m$ edges, $j = 0$, $H_1 = L$ and H_2 is a subgraph of R with k edges. Now every $s(a_i)$-fan must be mapped to a B-fan, such that all edges of the $s(a_i)$-fan are kept. Since $\frac{B}{4} < s(a) < \frac{B}{2}$, exactly three $s(a_i)$-fans must be mapped to one B-fan, and each B-fan looses exactly two edges. Hence, there is a solution of 3-partition.

The reduction from 3-partition to INS is similar. Consider the graph G from Figure 2 and let $k = 3Bm - 6m$.

The left subgraph L consists of $3m$ fans, where the i-th fan is a chain of $s(a_i) + (s(a_i) - 1)$ nodes and the nodes at odd positions are connected to v_1. The right subgraph R has m fans where each fan has exactly $B + (B - 1)$ nodes and the nodes at odd positions are connected to v_2. The nodes v_1 and v_2 are adjacent.

By the same reasoning as above it can be shown that G has two isomorphic

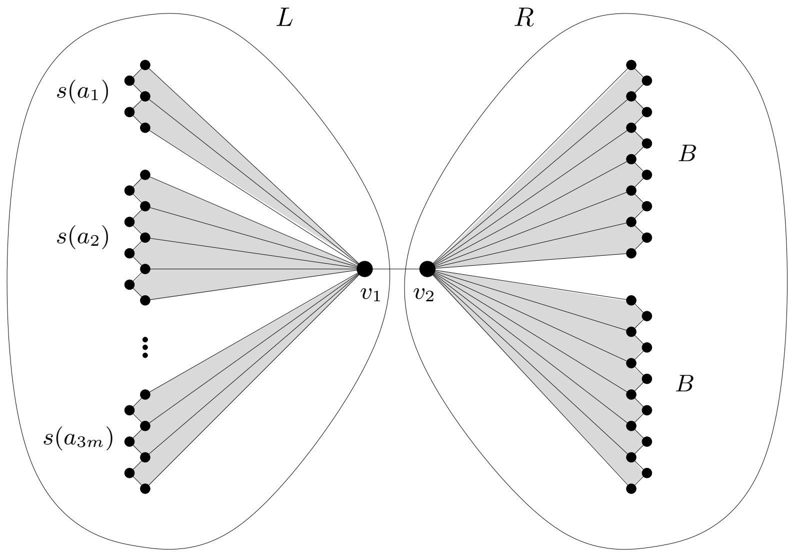

Figure 2: Reduction to INS for connected outerplanar graphs

node-induced subgraphs H_1 and H_2, with k edges each, and which may be connected or not if and only if $H_1 = L$ if and only if there is a solution of 3-partition.

Accordingly, we can proceed if we increase the connectivity and consider 2-connected planar graphs. For IES consider the graph G from Figure 3 and let $k = 3Bm - 3m$. Again it can be shown that L and R are the edge-induced isomorphic subgraphs. The construction of the graph for INS is left to the reader. However, the isomorphic subgraph problem is polynomially solvable for 2-connected outerplanar graphs [6]. □

4 Computing Isomorphic Subgraphs

In this section we introduce two different heuristics for the computation of maximal connected isomorphic subgraphs. 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 properties such as the degree and adjacencies of isomorphic nodes are not necessarily preserved, since an edge from a node in one of the isomorphic subgraphs may end in the remainder or in the other subgraph. 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.

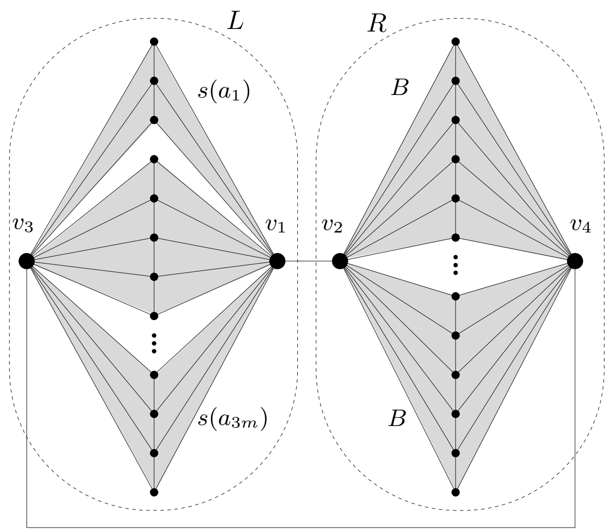

Figure 3: Reduction to IES for 2-connected planar graphs

4.1 The Matching Approach

Our first approach is a greedy algorithm, whose core is a bipartite matching for the local search. The generic algorithm has two main features: the initialization and a weighting function for the nodes. Several versions have been tested in [2]. The algorithm proceeds step by step and attempts to enlarge the intermediate pair of isomorphic subgraphs by a pair of new nodes, which are taken from the neighbors of a pair of isomorphic nodes. The correspondence between the neighbors 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. For each pair of new nodes (v_1, v_2) not in H_1 and H_2 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 neighbors)
- w_4 = the number of new neighbors 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 $(H_1 \cup v_1, H_2 \cup v_2)$.

These parameters can be combined arbitrarily to a weight $W = \sum \lambda_i w_i$ with $0 \leq \lambda_i \leq 1$. Observe that w_2 and w_3 are taken negative. Let $w_0 = \sum w_i$.

The greedy algorithm proceeds as follows: Suppose that H_1 and H_2 are disjoint connected isomorphic subgraphs which have been computed so far and are identified by the mapping $\phi : V_1 \to V_2$ on their sets of nodes. The algorithm selects the next pair of nodes $(u_1, u_2) \in V_1 \times V_2$ with $u_2 = \phi(u_1)$ from the queue of such pairs. It constructs a bipartite graph whose nodes are the new neighbors of u_1 and u_2. For each such pair (v_1, v_2) of neighbors of (u_1, u_2) with $v_1, v_2 \notin V_1 \cup V_2$ and $v_1 \neq v_2$ there is an edge with weight $W(v_1, v_2) = w_i$, where w_i is any of the above weighting functions. For example, $W(v_1, v_2)$ is the sum of the degrees of v_1 and v_2 for w_1. There is no edge if $v_1 = v_2$. Then compute an optimal weighted bipartite matching. For the matching we have used the Kuhn-Munkres algorithm (Hungarian method) [9] with a runtime of $O(n^2m)$. In decreasing order by the weight of the matching edges the pairs of matched nodes (v_1, v_2) are taken for an extension of H_1 and H_2, provided that (v_1, v_2) passes the isomorphism test and neither of them has been matched in preceding steps. The isomorphism test is only local, since each time only one new node is added to each subgraph. Since the added nodes are from the neighborhood, the isomorphic subgraphs are connected, and they are disjoint since each node is taken at most once.

In the node-induced case, a matched pair of neighbors (v_1, v_2) is added to the pair of isomorphic subgraphs if the isomorphism can be extended by v_1 and v_2. Otherwise v_1 and v_2 are skipped and the algorithm proceeds with the endnodes of the next matched edge. In the case of edge-induced isomorphic subgraphs the algorithm successively adds pairs of matched nodes and all edges between pairs of isomorphic nodes. Each node can be taken at most once, which is checked if u_1 and u_2 have common neighbors.

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 the neighbors of two nodes. The overall runtime of our algorithm is $O(n^3m)$. However, it performs much better in practice, since the expensive Kuhn-Munkres algorithm often operates on small sets. The greedy algorithm terminates if there are no further pairs of nodes which can extend the computed isomorphic subgraphs, which therefore are maximal.

For an illustration consider the graph in Figure 4 with $(1, 1')$ as the initial pair of isomorphic nodes. The complete bipartite graph from the neighbors has the nodes $2, 3, 4$ and $6, 7$. Consider the sum of degrees as weighting function. Then $w_1(3, 6) + w_1(2, 7) = 15$ is a weight maximal matching; the edges $(2, 6)$ and $(3, 7)$ have the same weight. First, node 3 is added to 1 and 6 is added to $1'$. In the node-induced case, the pair $(2, 7)$ fails the isomorphism test and is discarded. In the next step, the new neighbors of the pair $(3, 6)$ are considered, which are $2, 4, 5$ and $2, 5$. In the matching graph there is no edge between the two occurrences of 2 and 5. A weight maximal matching is $(2, 5)$ and $(4, 2)$ with weight 14. Again the isomorphism test fails, first for $(2, 5)$ and then for $(4, 2)$. If it had succeeded with $(2, 5)$ the nodes 2 and 5 were blocked and the edge $(4, 2)$ were discarded. Hence, the approach finds only a pair of isomorphic subgraphs whereas the optimal solution for INS are the subgraphs induced by $\{1, 3, 4, 7\}$ and by $\{2, 5, 6, 1'\}$ which are computed from the seed $(7, 1')$. In the edge-induced case there are the copies $(3, 6)$ and $(2, 7)$ from the neighbors of $(1, 1')$ discarding the edge $(2, 3)$ of G. Finally, $(4, 5)$ is taken as neighbors of $(3, 6)$. The edges of the edge-induced isomorphic subgraphs are drawn bold.

Figure 5 shows the algorithm for the computation of node-induced subgraphs. P consists of the pairs of nodes which have already been identified by

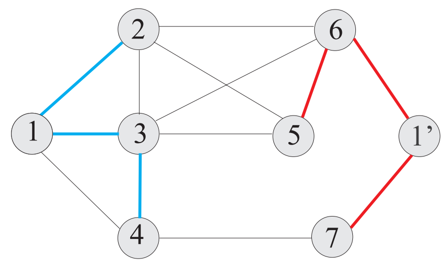

Figure 4: The matching approach

```
initialize(P);
while (P ≠ ∅) do
    (u1, u2) = next(P); delete (u1, u2) from P;
    N1 = new neighbors of u1;
    N2 = new neighbors of u2;
    M = optimal weighted bipartite matching over N1 and N2;
    forall_edges(v1, v2) of M decreasingly by their weight do
        if G[V1 ∪ {v1}] is isomorphic to G[V2 ∪ {v2}] then
            P = P ∪ {(v1, v2)};
            V1 = V1 ∪ {v1}; V2 = V2 ∪ {v2};
```

Figure 5: The matching approach for isomorphic node-induced subgraphs.

the isomorphism.

In our experiments a single parameter w_i or the sum w_0 is taken as a weight of a node. 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 distinct nodes as a seed? We ran our experiments

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

If the algorithm runs repeatedly it keeps the largest subgraphs. 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 elapsed time and the size of the computed subgraphs.

4.2 The Square Graph Approach

In 1972 Levi [26] has proposed a transformation of the node-induced common subgraph problem of two graphs into the clique problem of the so-called node-product or comparability graph. For two graphs the product graph consists of all pairs of nodes from the two graphs and the edges represent common adjacencies. Accordingly, the edge-induced common subgraph problem is transformed into the clique problem of the edge-product graph with the pairs of edges of the two graphs as nodes. Then a maximal clique problem must be solved. This can be done by the Bron-Kerbosch algorithm [7] or approximately by a greedy heuristic [4]. The Bron-Kerbosch algorithm works recursively and is widely used as an enumeration algorithm for maximal cliques. However, the runtime is exponential, and becomes impractical for large graphs. A recent study by Koch [25] introduces several variants and adaptations for the computation of maximal cliques in comparability graphs representing connected edge-induced subgraphs.

We adapt Levi's transformation to the isomorphic subgraph problem. In the node-induced case, the graph G is transformed into the (node) square graph, whose nodes are pairs of nodes of G. In the edge-induced case, the edge-square graph consists of all pairs of edges with distinct endnodes. The edges of the square graphs express a common adjacency. They are classified and labeled as generating and as stabilizing edges, representing the existence or the absence of common adjacencies, and are assigned high and low weights, respectively.

Due to the high runtime of the Bron-Kerbosch algorithm we have developed a local search heuristic for a faster computation of large cliques. Again we use weighting functions and several runs with different seeds. Several versions of local search strategies have been tested in [20]. The best results concerning quality and speed were achieved by the algorithm from Figure 6, using the following weights. The algorithm computes a clique C and selects a new node u with maximal weight from the set of neighbors U of the nodes in C. Each edge e of the square graph G^2 is a generating edge with weight $w(e) = |V|^2$ or a stabilizing edge of weight $w(e) = 1$. For a set of nodes X define the weight of a node v by $w_X(v) = \sum_{x \in X} w(v, x)$. The weight of a set of nodes is the sum of the weights of its elements. In the post-processing phase the algorithm exchanges a node in the clique with neighboring nodes, and tests for an improvement.

```
Construct the square graph G^2 = (X, E) of G;
Select two adjacent nodes v_1 and v_2 such that w_X({v_1, v_2}) is maximal;
    C = {v_1, v_2}; U = N(v_1) ∩ N(v_2);
while U ≠ ∅ do
    select u ∈ U such that w_C(u) + w_{C∪U}(u) is maximal;
    U = U ∩ N(u);
forall_nodes v ∈ C do
    forall_nodes u ∉ C do
        if C − {v} ∪ {u} is a clique
        then C = C − {v} ∪ {u};
```

Figure 6: The square graph approach.

In our experiments the algorithm is run five times with different pairs of nodes in the initialization, and it keeps the clique with maximal weight found so far. The repetitions stabilize the outcome of the heuristic. The algorithm runs in linear time in the size of G^2 provided that the weight of each node can be determined in $O(1)$ time.

4.3 Experimental Evaluation

The experiments were performed on four independent test suites [30].

- 2215 random graphs with a maximum of 10 nodes and 14 edges
- 243 graphs generated by hand
- 1464 graphs selected from the Roma graph library [35], and
- 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.

The graphs from the second test suite were constructed by taking a single graph from the Roma library, making a copy and adding a few 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 all connected graphs from the Roma graph library with $10, 15, \ldots, 60$ nodes are selected and inspected for isomorphic subgraphs. These graphs have been used at several places for experiments with graph drawing algorithms. However, these 1464 graphs are sparse with $|E| \sim 1.3|V|$. This specialty has not been stated at other places and results in very sparse isomorphic subgraphs. It is reflected by the numbers of nodes and edges in the isomorphic subgraphs as Figure 8 displays.

Finally, we randomly generated 740 dense graphs with $10, 15, \ldots, 50$ nodes and $25, 50, \ldots, 0.4 \cdot n^2$ edges, with five graphs for each number of nodes and edges, see Table 1.

nodes	**edges (average)**	**number of graphs**
10	25	5
15	50.0	15
20	87.7	30
25	125.4	45
30	188.1	70
35	250.7	95
40	325.7	125
45	413.3	160
50	500.9	195

Table 1: The dense graphs

First, consider the performance of the matching approach. It performs quite well, particularly on sparse graphs. The experiments on the first three test suites give a uniform picture. It 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 7.

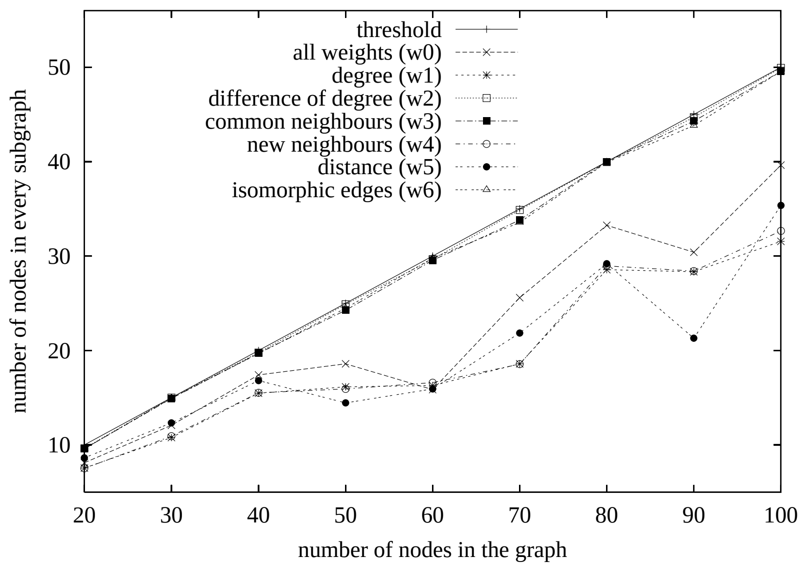

Figure 7: The matching approach computing isomorphic edge-induced subgraphs of the second test suite for the 90% best weighted pairs of starting nodes.

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 the weights w_0, w_1, w_4 and w_5 on a 750 MHz PC with 256 MB RAM, 30 seconds for w_2 and 100 seconds for w_3 and w_6, since in the latter cases the algorithm iterates over almost all pairs of starting nodes. In summary, the difference of degree weight w_2 gives the best performance.

For the third test suite the matching heuristic detects large isomorphic subgraphs. Almost all nodes are contained in one of the node- or edge-induced isomorphic subgraphs, which are almost trees. This can be seen from the small gap between the curves for the nodes and edges in Figure 8. There are only one or two more edges than nodes in each subgraph. This is due to the sparsity of the test graphs and shows that the Roma graphs are special. Again weight w_2 performs best, as has been discovered by tests in [2].

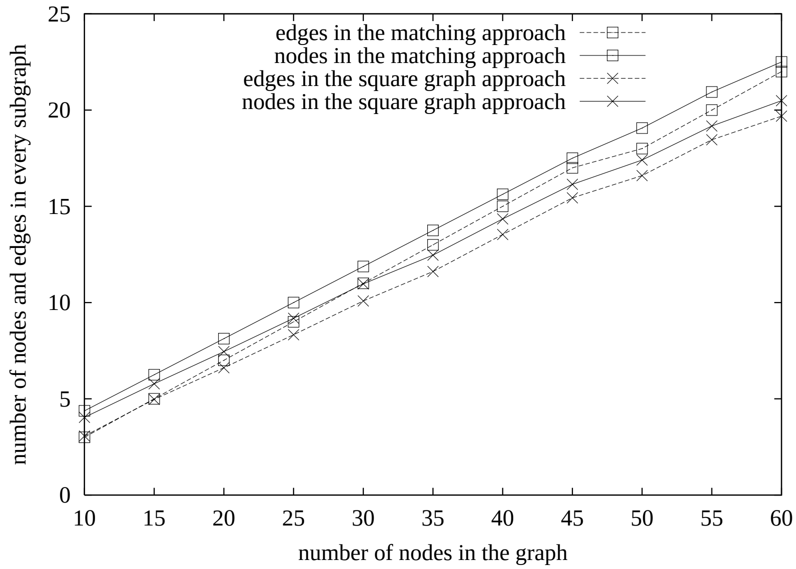

Figure 8: Comparison of the approaches on the Roma graphs

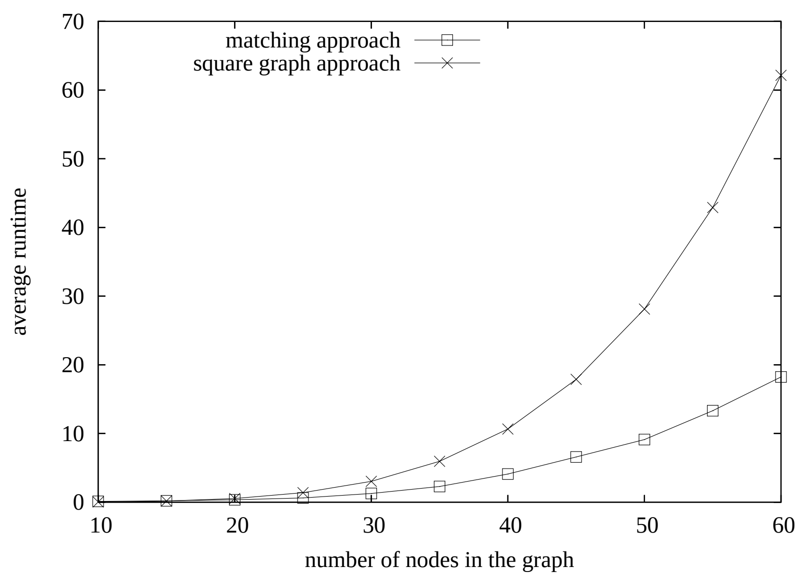

Figure 9: Run time of the approaches on the Roma graphs

Finally, on the dense graphs the matching heuristic with the 90% best weighted pairs of starting nodes and weight w_2 finds isomorphic edge-induced subgraphs which together contain almost all nodes and more than 1/3 of the edges. In the node-induced case each subgraph contains about 1/4 of the nodes and decreases from 1/5-th to 1/17-th for the edges, see Figure 10. For INS the isomorphic subgraphs turn out to be sparse, although the average density of the graphs increases with their size. The runtime of the matching approach increases due to the higher runtime of the Kuhn-Munkres matching algorithm on larger subgraphs, as Figure 11 shows.

The product graph approach has been tested on the same test suites for connected node-induced isomorphic subgraphs. The runtime is above that of the matching approach on the graphs from the Roma library, and takes about 25 seconds for graphs with 50 nodes. It is about the same for sparse and for dense graphs and grows quadratically with the size of the input graph due to the quadratic growth of the square graph. In contrast, the runtime of the matching approach heavily depends on the density and seems to grow cubic with the number of neighbors due to the Bron-Kerbosch matching algorithm. It takes 350 seconds on dense graphs with 50 nodes and 500 edges on the average. It should be noted that the absolute runtimes on a 650 MHz PC are due to the programming styles and are slowed down by the general-purpose data structure for graphs from Graphlet.

On the first three test suites the product graph algorithm is inferior to the matching approach, both in the size of the detected isomorphic subgraphs and in the run time, as Figures 8 and 9 show. The picture changes for our test suite of dense graphs. Here the product graph algorithm finds node-induced subgraphs with about 10% more nodes and edges for each subgraph than the matching heuristic, and the product graph algorithm is significantly faster, see Figures 10 and 11. Notice that the largest graphs from this suite have 50 nodes and 500 edges on the average from which 86 appear in the two isomorphic copies. The comparisons in Figures 8 and 10 show the numbers of nodes and edges of the computed isomorphic node-induced subgraphs, from which we can estimate the density resp. sparsity of the isomorphic subgraphs.

5 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 [10, 34] 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 [36] have introduced the notion of eumorphous tree drawings. However, eumorphous tree drawings of minimal width and integral coordinates are NP-hard. The radial tree algorithm of Eades [13] squeezes a subtree into a sector with a wedge in proportion to the number of the leaves of the subtree. Hence, isomorphic

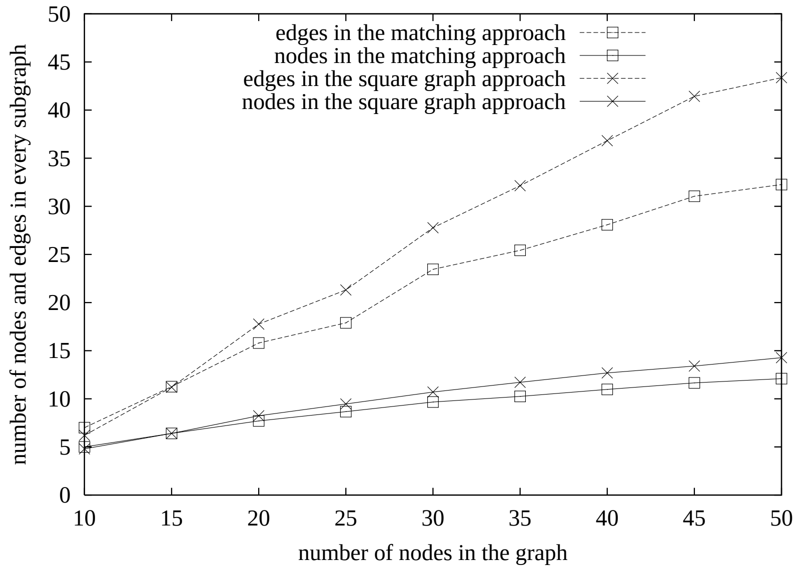

Figure 10: Comparison of the approaches on the dense graphs

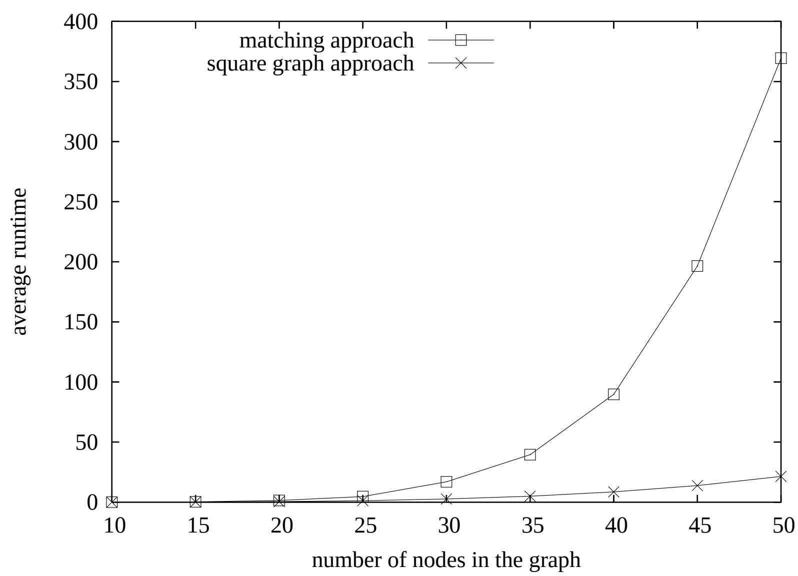

Figure 11: Run time of the approaches on the dense graphs

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 [14]. For planar graphs there are efficient algorithms for the detection and display of symmetries of the whole graphs [23, 22, 24]. Hierarchical graphs (DAGs) can be drawn with a symmetry by a modified barycenter algorithm [15] and by using a dominance drawing [11] if the graphs are planar and planar and reduced. For arbitrary hierarchical graphs symmetry preserving drawing algorithms are yet unknown.

The problem of drawing symmetric structures can be solved by a *three-phase method*, which is generally applicable in this framework. If a graph G is partitioned into two 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 remaining 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. The difficulties lie in the routing of the edges to the nodes of H_1 and H_2. This may lead to bad drawings with many node-edge crossings and parallel edges.

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 is achieved automatically by some tree drawing algorithms, and is solved here for a spring embedder. Spring algorithms have been introduced to graph drawing by Eades [12]. Fruchterman and Reingold [18] have simplified the forces to improve the running time, which is a critical factor for spring algorithms. Our algorithm is based on the advanced USE algorithm [16], which is included in the Graphlet system and is an extension of GEM [17]. In particular, USE supports individual distance parameters between any pair of nodes. Other spring algorithms allow only a uniform distance.

Our goal is 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, if they are isomorphic subgraphs. The placement of pairs of nodes $(v, \phi(v))$ remains the same, even if H_1 and H_2 are not isomorphic and are disconnected.

Large isomorphic subgraphs should be emphasized. They should be separated from each other and distinguished from the remainder. The distinction can be achieved by a node coloring. 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, with $k_1 < k_3 < k_2$. Additionally, H_1 and H_2 are moved in opposite directions by *move_subgraph*(H_1, f'') and *move_subgraph*$(H_2, -f'')$. The force f'' is the sum of the forces acting on the subgraph H_1 and an extra repelling force between the barycenters of the placements 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 [16, 17]. 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 H1
and H2, and a remainder R, and the isomorphism φ : H1 → H2
Initial_placement(H1, H2)
    while (termination is not yet accomplished) do
        forall_nodes_at_random (v, V(G)) do
            f = force(v);
            if (v ∈ V(H1) ∪ V(H2)) then
                f' = force(φ(v));
                f = (f+f')/2;
                move_node(φ(v), f);
            move_node(v, f);
        move_subgraph(H1, f''); move_subgraph(H2, −f'');
```

Figure 12: Isomorphism preserving spring embedder.

Figure 12 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(v,f)* applies the computed forces f to a single node v.

The examples shown below are computed by the heuristic and drawn by the spring algorithm. Nice drawings of graphs come from the second and third test suites, see Figures 13 and 14. More drawings can be found in [2] or by using the algorithm in Graphlet.

6 Acknowledgment

We wish to thank the anonymous referees for their useful comments and constructive criticism.

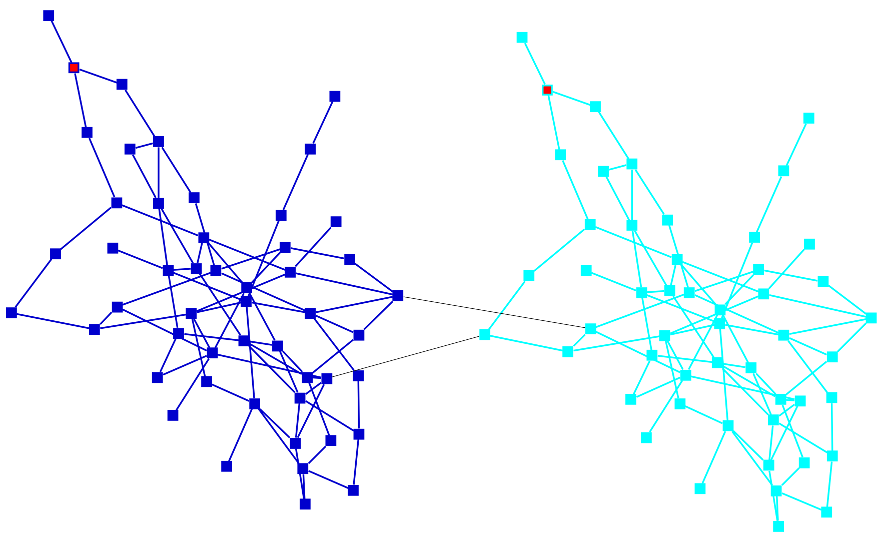

Figure 13: A graph of test suite 2. The original graph with 50 vertices and 76 edges was copied and then 2 edges were added. Our algorithm has re-computed the isomorphic edge-induced subgraphs.

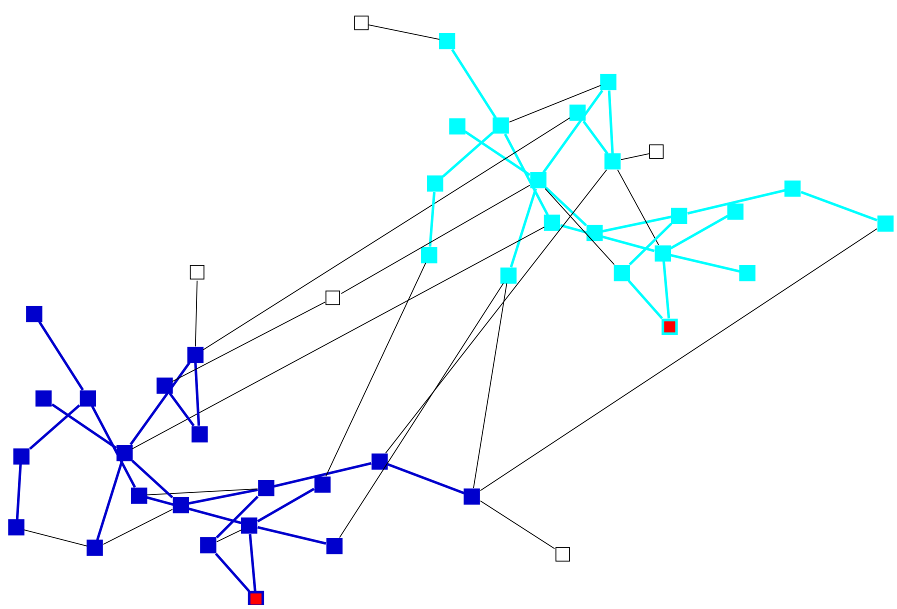

Figure 14: Graph (number 06549 in [35]) from test suite 3 with 45 nodes, and the computed isomorphic edge-induced subgraphs.

References

[1] S. Bachl. Isomorphic subgraphs. In *Proc. Graph Drawing'99*, volume 1731 of *LNCS*, pages 286–296, 1999.

[2] S. Bachl. *Erkennung isomorpher Subgraphen und deren Anwendung beim Zeichnen von Graphen*. Dissertation, Universität Passau, 2001. `www.opus-bayern.de/uni-passau/volltexte/2003/14/`.

[3] T. Biedl, J. Marks, K. Ryall, and S. Whitesides. Graph multidrawing: Finding nice drawings without defining nice. In *Proc. Graph Drawing'98*, volume 1547 of *LNCS*, pages 347–355. Springer Verlag, 1998.

[4] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo. *The maximum clique problem*, volume 4 of *Handbook of Combinatorial Optimization*. Kluwer Academic Publishers, Boston, MA, 1999.

[5] F. J. Brandenburg. Symmetries in graphs. *Dagstuhl Seminar Report*, 98301:22, 1998.

[6] F. J. Brandenburg. Pattern matching problems in graphs. Unpublished manuscript, 2000.

[7] C. Bron and J. Kerbosch. Algorithm 457 - finding all cliques in an undirected graph. *Comm. ACM*, 16:575–577, 1973.

[8] H.-L. Chen, H.-I. Lu, and H.-C. Yen. On maximum symmetric subgraphs. In *Proc. Graph Drawing'00*, volume 1984 of *LNCS*, pages 372–383, 2001.

[9] J. Clark and D. Holton. *Graphentheorie – Grundlagen und Anwendungen*. Spektrum Akademischer Verlag, Heidelberg, 1991.

[10] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, Englewood Cliffs, NJ, 1999.

[11] G. Di Battista, R. Tamassia, and I. G. Tollis. Area requirements and symmetry display of planar upwards drawings. *Discrete Comput. Geom.*, 7:381–401, 1992.

[12] P. Eades. A heuristic for graph drawing. In *Cong. Numer.*, volume 42, pages 149–160, 1984.

[13] P. Eades. Drawing free trees. *Bulletin of the Institute for Combinatorics and its Applications*, 5:10–36, 1992.

[14] P. Eades and X. Lin. Spring algorithms and symmetry. *Theoret. Comput. Sci.*, 240:379–405, 2000.

[15] P. Eades, X. Lin, and R. Tamassia. An algorithm for drawing a hierarchical graph. *Int. J. Comput. Geom. & Appl.*, 6:145–155, 1996.

[16] M. Forster. Zeichnen ungerichteter graphen mit gegebenen knotengrößen durch ein springembedder- verfahren. Diplomarbeit, Universität Passau, 1999.

[17] A. Frick, A. Ludwig, and H. Mehldau. A fast adaptive layout algorithm for undirected graphs. In *Proc. Graph Drawing'94*, volume 894 of *LNCS*, pages 388–403, 1995.

[18] T. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software – Practice and Experience*, 21:1129–1164, 1991.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, San Francisco, 1979.

[20] D. Gmach. Erkennen von isomorphen subgraphen mittels cliquensuche im produktgraph. Diplomarbeit, Universität Passau, 2003.

[21] A. Gupta and N. Nishimura. The complexity of subgraph isomorphism for classes of partial k-trees. *Theoret. Comput. Sci.*, 164:287–298, 1996.

[22] S.-H. Hong and P. Eades. A linear time algorithm for constructing maximally symmetric straight-line drawings of planar graphs. In *Proc. Graph Drawing'04*, volume 3383 of *LNCS*, pages 307–317, 2004.

[23] S.-H. Hong and P. Eades. Symmetric layout of disconnected graphs. In *Proc. ISAAC 2005*, volume 2906 of *LNCS*, pages 405–414, 2005.

[24] S.-H. Hong, B. McKay, and P. Eades. Symmetric drawings of triconnected planar graphs. In *Proc. 13 ACM-SIAM Symposium on Discrete Algorithms*, pages 356–365, 2002.

[25] I. Koch. Enumerating all connected maximal common subgraphs in two graphs. *Theoret. Comput. Sci.*, 250:1–30, 2001.

[26] G. Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, 9:341–352, 1972.

[27] A. Lubiw. Some np-complete problems similar to graph isomorphism. *SIAM J. Comput.*, 10:11–21, 1981.

[28] J. Manning. Computational complexity of geometric symmetry detection in graphs. In *LNCS*, volume 507 of *LNCS*, pages 1–7, 1990.

[29] J. Manning. *Geometric symmetry in graphs*. PhD thesis, Purdue Univ., 1990.

[30] Passau test suite. `http://www.infosun.uni-passau.de/br/isosubgraph`. University of Passsau.

[31] H. Purchase. Which aesthetic has the greatest effect on human understanding. In *Proc. Graph Drawing'97*, volume 1353 of *LNCS*, pages 248–261, 1997.

[32] H. Purchase, R. Cohen, and M. James. Validating graph drawing aesthetics. In *Proc. Graph Drawing'95*, volume 1027 of *LNCS*, pages 435–446, 1996.

[33] R. C. Read and R. J. Wilson. *An Atlas of Graphs*. Clarendon Press Oxford, 1998.

[34] E. M. Reingold and J. S. Tilford. Tidier drawings of trees. *IEEE Trans. SE*, 7:223–228, 1981.

[35] Roma graph library. `http://www.inf.uniroma3.it/people/gdb/wp12/LOG.html`. University of Rome 3.

[36] K. J. Supowit and E. M. Reingold. The complexity of drawing trees nicely. *Acta Informatica*, 18:377–392, 1983.

[37] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. Assoc. Comput. Mach.*, 16:31–42, 1970.