Available online at www.sciencedirect.com
SCIENCE DIRECT®

Neurocomputing 63 (2005) 45–67

NEUROCOMPUTING

www.elsevier.com/locate/neucom

Solving inexact graph isomorphism problems using neural networks

Brijnesh J. Jain*, Fritz Wysotzki

Department of Electrical Engineering and Computer Science, Technical University Berlin, Sekr. FR 5-8, Franklinstr. 28/29, D-10587 Berlin, Germany

Received 23 October 2003; accepted 23 January 2004
Available online 19 August 2004

Abstract

We present a neural network approach to solve exact and inexact graph isomorphism problems for weighted graphs. In contrast to other neural heuristics or related methods this approach is based on a neural refinement procedure to reduce the search space followed by an energy-minimizing matching process. Experiments on random weighted graphs in the range of 100–5000 vertices and on chemical molecular structures are presented and discussed.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Graph isomorphism; Association graph; Maximum clique; Hopfield network

1. Introduction

Given two graphs G and H the *graph isomorphism problem* (GIP) is the problem of deciding whether G and H are structurally equivalent. The problem is of practical as well as theoretical importance. Applications include the identification of isomorphic molecular structures in chemistry [23,34,41], the recognition of protein molecules [1], the detection of kinematic chains [32], or optimal routing of messages in multistage interconnecting networks [13], computer vision [5], and the construction and enumeration of combinatorial configurations [12]. The theoretical interest in the GIP

*Corresponding author. Tel.: +49-30-314-23938; fax: +49-30-314-24913.
E-mail address: bjj@cs.tu-berlin.de (B.J. Jain).

0925-2312/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2004.01.189

is based on the persistent difficulty in characterizing its computational complexity. The GIP is still unsolved in the sense that there is neither an NP-completeness proof, nor an efficient algorithm with polynomial complexity has yet been found.

Despite the practical and theoretical importance of the GIP no neural network approach and related heuristics can be used unmodified in a practical setting. Even the most powerful approaches by Pelillo [28] and Rangarajan et al. [30,31] require a prohibitive amount of time and are too erroneous on random graphs with only 100 vertices, although the GIP is considered to be trivial for almost all random graphs [6]. The main reason that neural networks or related methods are not competitive with efficient graph isomorphism algorithms like *Nauty* [25] is that they solely rely on powerful energy minimization procedures and neglect graph-theoretical properties that are preserved under isomorphism.

For inexact graph isomorphism problems, however, methods from the neural network community like the *Lagrangian Relaxation Network* (LRN) [31] and the *Optimizing Network Architecture* (ONA) [30] clearly outperform other approaches like the *Eigendecomposition Graph Matching* algorithm [39], *Polynomial Transform Graph Matching* algorithm [3], or the *Linear Programming Graph Matching* algorithm [4] on random weighted graphs as reported in [30]. More recent approaches like the *Principal Component Analysis Graph Matching* algorithm [42] and *RKHS Interpolator-Based Graph Matching* algorithm [40] were either tested on small graphs of order 5–10 or do not present execution times.

In this paper we focus on the inexact graph isomorphism problem comprising its exact counterpart as a special case. We extend the concept of vertex invariants to inexact isomorphism problems and devise a two stage neural graph isomorphism (NGI) algorithm. In a preprocessing step a neural refinement procedure decomposes the vertex sets of both graphs into subsets of structural similar vertices. In a second step the information about the decomposition is used to match the graphs with a special Hopfield network. The effectiveness of the proposed NGI approach is tested on random graphs with 100–5000 vertices and on chemical molecules.

The rest of this paper is organized as follows: The Section 2 introduces basic definitions and the statement of the problem. Section 3 defines and discusses vertex ε-invariants in the context of solving exact and inexact graph isomorphism problems. In Section 4 we propose the NGI algorithm. Section 5 presents the experimental results and Section 6 concludes this contribution.

2. Preliminaries

This section introduces basic notations and definitions used throughout this paper and gives a formal specification of the graph isomorphism problem.

Let U, V be sets. By $U \star V$ we denote the set of all 2-element subsets $\{i,j\}$ of the disjoint union $U \dot{\cup} V$ with $i \in U$ and $j \in V$. If $V = U$, we write $V^{[2]} = V \star V$, that is $V^{[2]}$ is the set of all 2-element subsets $\{i,j\} \subseteq V$.

A family $\mathscr{C} = \{V_1, \ldots, V_k\}$ of non-empty subsets $V_i \subseteq V$ with $V = \bigcup_i V_i$ is called a *cover* of V. The elements of a cover are its *clusters*. A *partition* $\mathscr{P}$ of a set V is a cover of V whose members are pairwise disjoint. The elements of a partition are usually called its *cells*. We say a cover $\mathscr{C}$ of V is *finer* than a cover $\mathscr{C}'$ of V, written as $\mathscr{C} \prec \mathscr{C}'$, if every cluster of $\mathscr{C}$ is a subset of some cluster of $\mathscr{C}'$. Under these conditions, $\mathscr{C}'$ is *coarser* than $\mathscr{C}$.

A *weighted graph* is a pair $G = (V, \mu)$ consisting of a finite set $V \neq \emptyset$ of *vertices* and a mapping $\mu : V^2 \to \mathbb{R}_+$ assigning each pair $(i,j) \in V^2$ a non-negative real valued weight $\mu(i,j) \geqslant 0$ with $\mu(i,j) = \mu(j,i)$. The weight of a vertex i is given by $\mu(i,i)$. The elements $\{i,j\} \in V^{[2]}$ with positive weight $\mu(i,j) > 0$ are the *edges* of G. The vertex set of a graph G is referred to as $V(G)$, its edge set as $E(G)$, and its weight mapping as μ_G. By $\mathscr{G}$ we denote the set of all weighted graphs. Note that a weighted graph is undirected, without multiple edges, and without loops.

A *binary graph* is a weighted graph $G = (V, \mu)$ with $\mu(V^2) \subseteq \{0, 1\}$. A binary graph assigns the weight 1 for its edges and the weight 0 for non-edges.

The *adjacency matrix* of a graph G is a matrix $A(G) = (g_{ij})$ with entries $g_{ij} = \mu_G(i,j)$.

A subset C_m of $V(G)$ consisting of m vertices is called *clique* of G, if $C_m^{[2]} \subseteq E(G)$. A *maximal clique* is a clique which is not properly contained in any larger clique. A *maximum clique* is a clique with maximum cardinality of vertices. The *clique number* $\omega(G)$ of a graph G is the number of vertices of a maximum clique in G.

A *matching* of a graph G is a subset $M \subseteq E(G)$ such that no two different edges are incident with a common vertex. A *perfect matching* is a matching which covers all vertices of G.

Let G and H be graphs with adjacency matrices $A(G) = (g_{ij})$ and $A(H) = (h_{ij})$, respectively. An *isomorphism* from G to H is a bijective mapping

$$\phi : V(G) \to V(H), \quad i \mapsto i^{\phi}$$

with $g_{ij} = h_{i^{\phi}j^{\phi}}$ for all $i,j \in V$. By $\mathscr{I}(G,H)$ we denote the set of all isomorphisms $\phi : V(G) \to V(H)$. An *automorphism* of G is an isomorphism from G to itself. Let Aut_G denote the set of all automorphisms of G. Two vertices $i, j \in V$ are *similar*, in symbols $i \sim j$, if there exists an automorphism $\phi \in \mathrm{Aut}_G$ with $i^{\phi} = j$. The *automorphism partition* $\mathscr{P}_G$ of G is the partition of $V(G)$ induced by the equivalence relation $\sim$. A cell of $\mathscr{P}_G$ is called *orbit*.

2.1. Statement of the problem

Throughout this contribution we assume that G and H are graphs with n vertices and adjacency matrices $A(G) = (g_{ij})$ and $A(H) = (h_{ij})$, respectively. By $\varepsilon \geqslant 0$ we denote a problem dependent threshold.

Two graphs G and H are *ε-isomorphic*, if there exists a bijection

$$\phi : V(G) \to V(H), \quad i \mapsto i^{\phi}$$

with $|g_{ij} - h_{i^{\phi}j^{\phi}}| \leqslant \varepsilon$ for all $i,j \in V(G)$. Such a mapping ϕ is called an *ε-isomorphism* between G and H. The *ε-graph isomorphism problem* (ε-`GIP`) is the problem of deciding whether two graphs are ε-isomorphic. In the following let $\mathscr{I}_{\varepsilon}(G,H)$ be the set of all ε-isomorphisms $\phi : V(G) \to V(H)$.

An *ε-automorphism* of G is an ε-isomorphism from G to itself. For $\varepsilon > 0$ the set $\text{Aut}_G^{\varepsilon}$ is in general not a group. Two vertices $i, j \in V$ are *ε-similar*, written as $i \sim_{\varepsilon} j$, if there exists an ε-automorphism ϕ with $i^{\phi} = j$. The *ε-automorphism cover* $\mathscr{C}_G$ of G is the cover of $V(G)$ induced by $\sim_{\varepsilon}$.

For $\varepsilon = 0$, we obtain the usual definitions of an isomorphism, automorphism, etc. In the following, we sometimes distinguish between $\varepsilon = 0$ and $\varepsilon > 0$ by referring to *exact* and *inexact* concepts, respectively.

Note, that the definition of an ε-isomorphism admits to map edges with weights less than ε and non-edges onto each other. Thus as opposed to exact isomorphisms the inexact counterpart can not only cope with noisy weights but also with missing edges to a certain degree.

3. Exact and inexact vertex invariants

To solve the exact graph isomorphism problem for G and H, any algorithm has to search for an isomorphism between G and H among $n!$ possible bijections $\phi : V(G) \to V(H)$. A practical and commonly applied technique to restrict the search space consisting of $n!$ possible candidates, is the use of *vertex invariants* [11]. A vertex invariant is a property of a vertex, which is preserved under isomorphism. Thus only vertices with the same invariants must be mapped onto each other under any isomorphism. This section briefly restates the well-known concept of vertex invariants for conventional isomorphism problems and then extends the basic ideas of that concept to the inexact counterpart.

3.1. Exact vertex invariants

Let $\mathscr{G}_V = \{(G, i) : G \in \mathscr{G}, i \in V(G)\}$. A function $f : \mathscr{G}_V \to \mathbb{R}^p$ is a (p-dimensional) *vertex invariant*, if

$$f(G, i) = f(H, i^{\phi})$$

for all $i \in V(G)$ and for any isomorphism $\phi \in \mathscr{I}(G, H)$. We call the vector $f(G, i)$ *invariant of vertex $i \in V(G)$*.

The best known and most frequently used (exact) vertex invariant is the degree of a vertex. Further examples assign each vertex the number of vertices reachable along a path of length k, the number of different cliques of size k, etc. All these invariants are one-dimensional. An example of a p-dimensional invariant is any combination of p one-dimensional invariants.

Any isomorphism of G and H must map vertices with the same invariant onto each other. Thus the objective of using vertex invariants is to decompose the vertex sets of both graphs such that the pruned search space becomes significantly smaller than $n!$.

A vertex invariant f induces a partition $\mathscr{P}_f(G) = \{V_1(G), \ldots, V_r(G)\}$ of $V(G)$ with two vertices in the same cell if and only if their invariants are identical. Since f is constant on the orbits of Aut_G, the partition $\mathscr{P}_f(G)$ is coarser than or equal to the automorphism partition $\mathscr{P}_G$.

If G and H are isomorphic graphs, then $|\mathscr{P}_f(G)| = |\mathscr{P}_f(H)| = r$ and there exists a numbering of the cells in $\mathscr{P}_f(H)$ such that $|V_k(G)| = |V_k(H)|$ and $f(G,i) = f(H,j)$ for all $i \in V_k(G), j \in V_k(H)$ and for all $1 \leqslant k \leqslant r$. Thus, to establish an isomorphism between G and H we may restrict the search space from $n!$ possible candidates to

$$\mathrm{c}(f, G, H) = \prod_{k=1}^{r} (n_k!) \tag{1}$$

candidates, where $n_k = |V_k(G)| = |V_k(H)|$. From Eq. (1) follows that the number of candidates in the pruned search space becomes smaller the better a partition approximates the automorphism partition.

3.2. *Inexact vertex invariants*

Let $\mathscr{G}_V = \{(G,i) : G \in \mathscr{G}, i \in V(G)\}$. A function

$$f : \mathscr{G}_V \to \mathbb{R}, \quad (G,i) \mapsto (f_1(G,i), \ldots, f_p(G,i))$$

is a (p-dimensional) *vertex ε-invariant*, if

$$\left| f_q(G,i) - f_q(H, i^{\phi}) \right| \leqslant \varepsilon$$

for all $1 \leqslant q \leqslant p$, for all $i \in V(G)$, and for any ε-isomorphism $\phi \in \mathscr{I}_\varepsilon(G,H)$. We call the vector $f(G,i)$ *ε-invariant of $i \in V(G)$* with respect to f. For $\varepsilon = 0$, we obtain the usual notion of an exact vertex invariant.

According to Proposition 1, examples of one-dimensional vertex ε-invariants are the mean weight, the maximum or the minimum weight of edges incident to a vertex. A set of p one-dimensional ε-invariants can be combined to a p-dimensional invariant.

Proposition 1. *Let G be a graph with adjacency matrix $A(G) = (g_{ij})$. For any $\varepsilon \geqslant 0$ the following functions are vertex ε-invariants*:

$$f_1(G,i) = \frac{1}{n} \sum_{j \neq i} g_{ij},$$

$$f_2(G,i) = \max\{g_{ij} : 1 \leqslant j \leqslant n\},$$

$$f_3(G,i) = \min\{g_{ij} : 1 \leqslant j \leqslant n\}.$$

Proof. Let G and H be ε-isomorphic graphs with adjacency matrices $A(G) = (g_{ij})$ and $A(H) = (h_{ij})$, respectively. Suppose that $\phi \in \mathscr{I}_\varepsilon(G,H)$ is an ε-isomorphism and $i \in V(G)$ with $j = i^{\phi} \in V(H)$.

Proof of assertion for f_1: By definition of an ε-isomorphism, we have $|g_{ik} - h_{jk^\phi}| \leqslant \varepsilon$ for all $k \neq i$. This yields

$$|f_1(G,i) - f_1(H,j)| = \left| \frac{1}{n} \sum_{k \neq i} g_{ik} - \frac{1}{n} \sum_{l \neq j} h_{jl} \right| \leqslant \frac{1}{n} \sum_{k \neq i} |g_{ik} - h_{jk^\phi}| \leqslant \varepsilon.$$

Proof of assertion for f_2: Let $g_i^* = f_2(G,i)$ and $h_j^* = f_2(H,j)$. Then there are vertices $k, l \in V(G)$ with $g_{ik} = g_i^*$ and $h_{jl^\phi} = h_j^*$, respectively. By definition of ϕ we have $|g_i^* - h_{jk^\phi}| \leqslant \varepsilon$ and $|g_{il} - h_j^*| \leqslant \varepsilon$. We distinguish three cases:

Case 1: Assume that $(g_i^* - h_j^*)(g_i^* - h_{jk^\phi}) = 0$. Then we have $g_i^* = h_j^*$ or $g_i^* = h_{jk^\phi}$. If $g_i^* = h_j^*$, then nothing is to show. Suppose that $g_i^* = h_{jk^\phi}$. Then we find that $g_{il} \leqslant g_i^* = h_{jk^\phi} \leqslant h_j^*$. Hence, $|g_i^* - h_j^*| \leqslant |g_{il} - h_j^*| \leqslant \varepsilon$.

Case 2: Let $(g_i^* - h_j^*)(g_i^* - h_{jk^\phi}) < 0$. From $h_j^* \geqslant h_{jk^\phi}$ follows $h_{jk^\phi} < g_i^* < h_j^*$. In addition, we have $g_{il} \leqslant g_i^* < h_j^*$ Hence, $|g_i^* - h_j^*| \leqslant |g_{il} - h_j^*| \leqslant \varepsilon$.

Case 3: Finally, assume that $(g_i^* - h_j^*)(g_i^* - h_{jk^\phi}) > 0$. We distinguish two further cases: (a) Either both terms are positive or (2) both terms are negative. If (a) holds, then $g_i^* > h_j^* \geqslant h_{jk^\phi}$ and therefore $|g_i^* - h_j^*| \leqslant |g_i^* - h_{jk^\phi}| \leqslant \varepsilon$. If (b) holds, we have $g_{il} \leqslant g_i^* < h_{jk^\phi} \leqslant h_j^*$. The assertion follows from $|g_i^* - h_j^*| \leqslant |g_{il} - h_j^*| \leqslant \varepsilon$.

Proof of assertion for f_3: Similar to proof of assertion for f_2. □

Similarly, as in the case of exact invariants, an ε-isomorphism of G and H maps vertices of G to vertices of H, if their respective ε-invariants do not differ by more than ε. Thus to prune the search space, we decompose the vertex sets of both graphs into clusters of an ε-invariant cover.

A vertex ε-invariant f induces a cover $\mathscr{C}_f(G) = \{C_1, \ldots, C_r\}$ of $V(G)$, called *f-cover*, such that

$$i,j \in C_k \Rightarrow |f_q(G,i) - f_q(G,j)| \leqslant \varepsilon$$

for all $1 \leqslant q \leqslant p$ and all $1 \leqslant k \leqslant r$.

Assume that G and H are ε-isomorphic. The decompositions of both vertex sets $V(G)$ and $V(H)$ into f-covers do not as well cooperate as in the case of exact invariants. The f-covers $\mathscr{C}_f(G)$ and $\mathscr{C}_f(H)$ may differ in the number of their clusters. Moreover, it is possible that there is no pair of clusters $(C, C') \in \mathscr{C}_f(G) \times \mathscr{C}_f(H)$ such that $|f(G,i) - f(H,j)| \leqslant \varepsilon$ for all $(i,j) \in C \times C'$.

To assess the effects of an ε-invariant f on pruning the search space we consider the bipartite graph $B_f = G_f H$ of G and H induced by f. The graph B_f is a binary graph with vertex and edge set

$$V = V(G) \dot{\cup} V(H),$$
$$E = \{\{i,j\} \in V(G) \star V(H) : |f_q(G,i) - f_q(H,j)| \leqslant \varepsilon, 1 \leqslant q \leqslant p\}.$$

Since G and H are ε-isomorphic, there exists at least one perfect matching of B_f. To find an ε-isomorphism of G and H we may restrict the search space from $n!$ possible candidates to

$$\mathfrak{c}(f, G, H) = |\mathscr{M}(B_f)|, \tag{2}$$

where $\mathscr{M}(B_f)$ is the set of all perfect matchings of B_f. Unfortunately, $|\mathscr{M}(B_f)|$ is in general unknown and time consuming to compute [15]. Under some mild assumptions the next results show how $\mathfrak{c}$ can be minimized without knowing its value.

Let B_f be the bipartite graph of G and H induced by f. By

$$N_f(i) = \{j \in V(H) : (i,j) \in E(B_f)\}$$

we denote the *neighborhood* of vertex $i \in V(G) \subseteq V(B_f)$ in B_f.

Lemma 1. *Let G, H be graphs, let $\{f_1, \ldots, f_p\}$ be a set of one-dimensional vertex ε-invariants, and let $F_q = (f_1, \ldots, f_q)$ for all $1 \leqslant q \leqslant p$. Then*

$$q < r \quad \Rightarrow \quad \mathfrak{c}(F_q, G, H) \geqslant \mathfrak{c}(F_r, G, H)$$

for all $1 \leqslant q, r \leqslant p$.

Proof. Assume that $p = 2$. To get rid of indices, we set $f = f_1 = F_1$, $g = f_2$, and $h = F_2 = (f, g)$. Thus the assertion to show is $\mathfrak{c}(f, G, H) \geqslant \mathfrak{c}(h, G, H)$.

Let B_f and B_h be the bipartite graphs of G and H induced by f and h, respectively. First we show, that $N_h(i) \subseteq N_f(i)$ for all $i \in V(G)$. Let $j \in N_h(i)$. Then we have $|f(G,i) - f(H,j)| \leqslant \varepsilon$ and $|g(G,i) - g(H,j)| \leqslant \varepsilon$. From the former inequality follows that $j \in N_f(i)$. Thus $N_h(i) \subseteq N_f(i)$.

By definition, $N_h(i)$ and $N_f(i)$ are complete bipartite subgraphs of B_h and B_f, respectively. Deleting a vertex from a neighborhood of $i \in V(G)$ in a bipartite graph of G and H induced by some ε-invariant does not increase the number of possible perfect matchings. Hence, $\mathfrak{c}(f, G, H) \geqslant \mathfrak{c}(h, G, H)$.

The proof for $p > 2$ follows the same argumentation as for $p = 2$. □

According to Lemma 1 refining a cover by adding ε-invariants does not increase $\mathfrak{c}$. A decrease of $\mathfrak{c}$ may occur, when additional invariants narrow the neighborhood $N_f(i)$ in B_f of a vertex $i \in V(G)$.

We conclude this section with a result telling us that the ε-automorphism cover is not coarser than a vertex ε-invariant cover. Hence, the best reduction of the search space we can expect is obtained by approximating the ε-automorphism cover.

Lemma 2. *Let G be a graph, and let f be a vertex ε-invariant. Then*

$$\mathscr{C}_G \prec \mathscr{C}_f(G).$$

Proof. Let $C \in \mathscr{C}_G$ be a cluster of the ε-automorphism cover. We show that there is a cluster $C' \in \mathscr{C}_f(G)$ with $C \subseteq C'$. Let $i,j \in C$. Then there is an ε-automorphism ϕ with $i^\phi = j$. This implies $|f_q(G,i) - f_q(G,j)| \leqslant \varepsilon$ for all q. Since f is an ε-invariant, there is a cluster $C' \in \mathscr{C}_f$ with $i,j \in C'$. Hence, we have $\mathscr{C}_G \prec \mathscr{C}_f(G)$. □

4. A neural ε-GIP solver

In practice, most algorithms adopt the same basic approach to the exact graph isomorphism problem, though the details may vary. To reduce the search space the common approach first approximates the automorphism partition of each graph using a vertex classification procedure which is based on a set of vertex invariants. In a second step an isomorphism is constructed or non-isomorphism is established by applying a breadth-first search, depth-first search, or a mixture of both methods. The NGI algorithm extends this approach for solving exact and inexact graph isomorphism problems.

Outline of the NGI *algorithm*: Let G and H be graphs with n vertices.

(1) Find covers $\mathscr{C}(G)$ of $V(G)$ and $\mathscr{C}(H)$ of $V(H)$ by using a neural refinement procedure.
(2) Use the covers $\mathscr{C}(G)$ and $\mathscr{C}(H)$ to construct an ε-association graph $G\diamond_\varepsilon H$ of G and H. The ε-association graph is an auxiliary structure to cast the isomorphism problem to a maximum clique problem.
(3) Find a maximum clique C_m in $G\diamond_\varepsilon H$ by using a special Hopfield network.
(4) G and H are ε-isomorphic if and only if $m = n$.

Note that the NGI algorithm can be modified in Step 3 by using other clique algorithms or even a different method to establish (non-) isomorphism.

Before describing Step 1–3 of the NGI algorithm in detail, we precede the presentation with a general notion to simplify technicalities. All neural networks involved in the NGI algorithm are associated with a specific graph. Networks for refining covers are associated with the given graphs to test for ε-isomorphism and the network for solving the maximum clique problem is associated with an association graph. A neural network N_G associated with G consists of n fully connected units. The dynamics of the network is given by

$$x_i(t+1) = (1-d)x_i(t) + \sum_{j\neq i} w_{ij}o_j(t), \tag{3}$$

where $x_i(t)$ denotes the activity of unit i and d is the self-inhibition. The synaptic weights w_{ij} between unit i and unit j are of the form

$$w_{ij} = \begin{cases} >0 : & \text{if } \{i,j\} \in E(G), \\ \leqslant 0 : & \text{if } \{i,j\} \notin E(G). \end{cases}$$

The output function $o_i(t) = o_i(x(t))$ of unit i is a non-decreasing function applied on $x_i(t)$. The *state vector* of the network N_G at time t is defined by the vector $\boldsymbol{x}(t) = (x_1(t), \ldots, x_n(t))$.

4.1. Step 1—Approximating the ε-automorphism partition

A state vector $\boldsymbol{x}(t)$ of a network N_G associated with a graph G can be regarded as a function

$$f_t : \mathscr{G}_V \to \mathbb{R}, \quad (G,i) \mapsto x_i(t)$$

assigning each vertex of a graph the state of its corresponding unit. This formulation serves to illustrate that a state vector of N_G has the same functional form as a vertex ε-invariant. Thus the states $x_i(t)$ determine a cover such that two units belong to the same cluster of that cover, if and only if their initial activations are ε-similar. With evolving time the states and thus the cover changes. In the following we show under which conditions activations of units are vertex ε-invariants of the underlying graph. Finally, we mention the limitations of this approach.

4.1.1. Exact case

Let N_G be a neural network associated with a graph G. Suppose the initial activation of each unit i of N_G is given by $h(g_{ii})$ where h is an arbitrary real valued function. Then N_G together with update rule (3) is a refinement procedure, which approximates the automorphism partition of G: At each time instance $t \geqslant 0$ the state $\boldsymbol{x}(t)$ of N_G induces a partition $\mathscr{P}_t(G)$ of the vertex set $V(G)$. Two vertices i and j are members of the same cell $V_k(t) \in \mathscr{P}_t(G)$ if and only if $x_i(t) = x_j(t)$. The initial state induces the initial partition $\mathscr{P}_0(G)$ which is iteratively refined according to dynamical rule (3). The refinement procedure terminates, when the current partition $\mathscr{P}_t(G)$ is not finer than $\mathscr{P}_{t-1}(G)$. Theorem 1 shows that the state $\boldsymbol{x}(t)$ of N_G is a vertex invariant and therefore justifies this approach.

Theorem 1. *Let G be a graph with adjacency matrix* $A(G) = (g_{ij})$ *and let* N_G *be the neural network associated with G. Suppose that h is a real-valued function. If* $w_{ij} = h(g_{ij})$ *and* $x_i(0) = h(g_{ii})$ *for all* $i,j \in V(G)$, *then*

$$i \sim j \quad \Rightarrow \quad x_i(t) = x_j(t)$$

for all $i,j \in V(G)$ *and all* $t \geqslant 0$.

Proof. Let $i,j \in V(G)$ be vertices with $i \sim j$ and $\phi \in \mathrm{Aut}_G$ with $j = i^{\phi}$. For $t = 0$ the assertion follows from $x_i(0) = h(g_{ii}) = h(g_{jj}) = x_j(0)$. Now assume that $x_i(t) = x_j(t)$ holds for some $t > 0$. Since $x_i(t) = x_j(t)$, we have $o_i(t) = o_j(t)$. Furthermore, any automorphism preserves adjacency relations. Hence,

$$w_{ik} = h(g_{ij}) = h(g_{jk^{\phi}}) = w_{jk^{\phi}}.$$

Then by induction we have

$$\begin{aligned} x_i(t+1) &= (1-d)x_i(t) + \sum_{k \neq i} w_{ik} o_k(t) = (1-d)x_j(t) + \sum_{k^{\phi} \neq j} w_{jk^{\phi}} o_{k^{\phi}}(t) \\ &= x_j(t+1). \qquad \square \end{aligned}$$

Note that, if $x_i(0) = 1$, $d = 1$, and

$$w_{ij} = \begin{cases} 1 : & \text{if } \{i,j\} \in E(G) \\ 0 : & \text{if } \{i,j\} \notin E(G) \end{cases}$$

for all $i,j \in V(G)$, then we obtain Morgan's procedure [26].

A refinement using $\boldsymbol{x}(t)$ as ε-invariant may have the undesirable effect of merging members $i \in C$ and $j \in C'$ of different cells $C, C' \in \mathscr{P}_{t-1}(G)$ at time step t-1 to a common cell $C'' \in \mathscr{P}_t(G)$ at time step t. It is a characteristic feature of invariants that members of different cells are not similar and therefore should remain in different cells. One way around this problem is to use the current partition as starting point for the next partition. In mathematical terms: Since each state $x_i(t)$ is an one-dimensional invariant of vertex i for all $t \geqslant 0$, the vector $f_i(t) = (x_i(0), \ldots, x_i(t))$ is a $t+1$-dimensional invariant of i. At each time step $t+1$, dynamics (3) separately refines each cell of the current partition $\mathscr{P}_{f(t)}(G)$ induced by $f(t) = (f_1(t), \ldots, f_n(t))$ such that members from different cells are not joined.

On the other hand, a refinement procedure using $f(t)$ as a set of ε-invariant usually results in a finer partition than the same procedure using $\boldsymbol{x}(t)$, but may require more computational time to converge to a locally finest partition. Thus, one has to compromise between computational complexity of partitioning the vertex sets and establishing (non-) isomorphism. In this sense using invariants is more an art than a science.

It is left to show that the neural refinement procedure of N_G terminates.

Theorem 2. *Let N_G be a neural network associated with a graph G. Then the partition refinement procedure* (3) *using $\boldsymbol{x}(t)$ and $f(t)$ as ε-invariant terminates within finite time.*

Proof. Update rule (3) induces a sequence of partitions $(\mathscr{P}_t(G))_{t \geqslant 0}$. Since $|V(G)|$ is finite there are only a finite number of possible partitions. Hence, there is a $t_0 \geqslant 0$ such that $\mathscr{P}_{t_0+1}(G)$ is not finer than $\mathscr{P}_{t_0}(G)$. The convergence proof for $(\mathscr{P}_{f(t)}(G))_{t \geqslant 0}$ follows a similar argumentation. $\square$

4.1.2. Inexact case

Let G be a graph with adjacency matrix $A(G) = (g_{ij})$. For convenience of presentation we assume that the weights of G are normalized, that is $g_{ij} \in [0,1]$ for all $i,j \in V(G)$. At each time step t, the state vector $\boldsymbol{x}(t)$ of the network N_G associated with G induces a cover $\mathscr{C}_t(G)$ of $V(G)$. Two vertices $i,j \in V(G)$ are member of the same cluster $V_k(t) \in \mathscr{C}_t(G)$, if and only if $|x_i(t) - x_j(t)| \leqslant \varepsilon$. The initial state $\boldsymbol{x}(0)$ induces an initial cover $\mathscr{C}_0(G)$ which is then iteratively refined by applying update rule (3). In the following we show under which conditions the activations of N_G are ε-invariants of G.

We assume that the network N_G associated with G satisfies the following conditions:

IE1 The self-inhibition of each unit is $d = 1$.

IE2 The weights of N_G are of the form

$$w_{ij} = \begin{cases} g_{ij}/3(n-1) : \text{ if } \{i,j\} \in E(G), \\ 0 : \text{ if } \{i,j\} \notin E(G). \end{cases}$$

IE3 $x_i(0) = g_{ii}$ for all $i \in V(G)$.

IE4 $|o_i(t)| \leqslant 1$ for all $i \in V(G)$ and all $t \geqslant 0$.

IE5 $|o_i(t) - o_i(t)| \leqslant |x_i(t) - x_j(t)|$ for all $i,j \in V(G)$ and all $t \geqslant 0$.

The first condition requires that self-weights $w_{ii} = 1 - d$ are set to zero. The second condition serves to average out the error $|g_{ij} - g_{kl}|$ when mapping all edges incident to a vertex i to all edges incident to a vertex k. The factor $\frac{1}{3}$ scales the average error to keep $|x_i(t) - x_j(t)|$ bounded through evolution, if i and j are ε-similar. The third condition determines the initial state of N_G. The fourth condition bounds o_i. Finally, the fifth condition states that the output function satisfies the Lipschitz condition with Lipschitz constant $K \leqslant 1$. The Lipschitz condition on o_i ensures that the difference of the inputs is not larger than the difference of the outputs. Note, that conditions `IE1`, `IE3` and `IE4` turn the network N_G in a special variant of a Hopfield model.

Theorem 3 states that the activation of units corresponding to ε-similar vertices remain sufficiently close together. Stated in other terms, the activations are an ε-invariant of their corresponding vertices.

Theorem 3. *Let G be a graph with adjacency matrix $A(G) = (g_{ij})$. Assume that the network N_G associated with G satisfies the properties* `IE1–IE5`. *Then*

$$i \sim_\varepsilon j \quad \Rightarrow \quad |x_i(t) - x_j(t)| \leqslant \varepsilon$$

for all $i,j \in V(G)$ and all $t \geqslant 0$.

Proof. Let $i,j \in V(G)$ be two vertices with $i \sim_\varepsilon j$ and $\phi \in \mathrm{Aut}_G$ with $i^\phi = j$. Clearly, the assertion holds for $t = 0$, since

$$|x_i(0) - x_j(0)| = |g_{ii} - g_{jj}| \leqslant \varepsilon$$

by `IE3`. Now assume that $|x_i(t) - x_j(t)| \leqslant \varepsilon$ for some $t \geqslant 0$. By induction we have

$$\begin{aligned} \left|x_i(t+1) - x_j(t+1)\right| &= \left|\sum_{k \neq i} w_{ik} o_k(t) - \sum_{k^\phi \neq j} w_{jk^\phi} o_{k^\phi}(t)\right| \\ &= \frac{1}{3(n-1)} \left|\sum_{k \neq i} g_{ik} o_k(t) - \sum_{k^\phi \neq j} g_{jk^\phi} o_{k^\phi}(t)\right| \\ &= \frac{1}{3(n-1)} \left|\sum_{k \neq i} g_{ik} o_k(t) - \sum_{k \neq i} (g_{ik} + \varepsilon_{ik})(o_k(t) + \varepsilon_{kk})\right| \end{aligned}$$

with $\varepsilon_{ik} = g_{i^\phi k^\phi} - g_{ik}$ and $\varepsilon_{kk} = o_{k^\phi}(t) - o_k(t)$. We have $|\varepsilon_{ik}| = |g_{i^\phi k^\phi} - g_{ik}| \leqslant \varepsilon$. Furthermore, from $|x_i(t) - x_j(t)| \leqslant \varepsilon$ by assumption together with $|o_i(t) - o_j(t)| \leqslant |x_i(t) - x_j(t)|$ by `IE5` directly follows $|\varepsilon_{kk}| \leqslant \varepsilon$. Hence,

$$\begin{aligned} |x_i(t+1) - x_j(t+1)| &= \frac{1}{3(n-1)} \left|\sum_{k \neq i} \varepsilon_{kk} g_{ik} + \sum_{k \neq i} \varepsilon_{ik} o_k(t) + \sum_{k \neq i} \varepsilon_{ik} \varepsilon_{kk}\right| \\ &\leqslant \frac{\varepsilon}{3(n-1)} \left(\underbrace{\sum_{k \neq i} g_{ik} + \sum_{k \neq i} |o_k(t)|}_{\leqslant 2(n-1)} + \varepsilon(n-1) \right) \end{aligned}$$

$$\leqslant \frac{\varepsilon}{3(n-1)}\left(\underbrace{2(n-1)+\varepsilon(n-1)}_{\leqslant 3(n-1)}\right)$$
$$= \varepsilon.$$

The inequality from the second to the third line uses `IE4`. This proves the assertion. □

As opposed to the exact case and despite Lemma 2, simply refining the vertex set of $V(G)$ by means of $\boldsymbol{x}(t)$ has no theoretical justification. Due to the inexactness the effects of finer covers on pruning the search space are in general not clear. Lemma 1 rectifies the situation, if we consider $f(t) = (\boldsymbol{x}(0), \ldots, \boldsymbol{x}(t))$ as $t+1$-dimensional ε-invariant composed of the $t+1$ one-dimensional ε-invariants $\boldsymbol{x}(0), \ldots, \boldsymbol{x}(t)$.[1]

The convergence proof of the inexact refinement procedure using $f(t)$ as ε-invariant is almost identical to the proof of Theorem 2. We include Theorem 4 for sake of completeness.

Theorem 4. *Let N_G be a neural network associated with a graph G. Then the refinement procedure* (3) *using $f(t)$ as ε-invariant terminates within finite time.*

4.1.3. Limitations

For exact graph isomorphism problems, the neural refinement procedure will not partition the vertex set into proper subsets for *regular graphs*[2], not even in the case when the automorphism group $\mathrm{Aut}_G = \{id\}$ is the trivial group.

In the case of inexact graph isomorphism problems, the proposed refinement procedure fails to partition the vertex set, if the pairwise average weighted degrees[3] of all vertices do not differ by more than $\varepsilon/3$. Stated in equivalent terms, partitioning of the vertex set fails, if ε is too large.

Moreover, the assumptions `IE2` is rather strict. Due to the scaling factor $1/(3(n-1))$ the states of N_G converge to zero while the sequence of partitions $\mathscr{P}_t(G)$ induced by the ε-invariant $\boldsymbol{x}(t)$ converges to the trivial partition $\{V(G)\}$ after a few iterations. Since partitioning terminates as soon as the next partition is not finer than the current one, the proposed inexact refinement procedure may stop after the first or second iteration with a cover which is much coarser than the ε-automorphism cover. To obtain a finer partition of $V(G)$ we may relax `IE2` provided some assumptions are satisfied. For example, assume that G is a model graph and H is a noisy copy such that G and H are ε-isomorphic. Then there is a graph H' isomorphic to H such that

$$A(H') = A(G) + (\varepsilon_{ij}),$$

[1]Note that $\boldsymbol{x}(t)$ is n-dimensional as state vector of N_G, but one-dimensional as vertex ε-invariant of G.

[2]A graph is regular, if all vertices have the same number of incident edges.

[3]The weighted degree of a vertex is the sum of the weights of its incident edges.

where ε_{ij} denotes the noise of edge $\{i,j\} \in E(G)$. Suppose also that the noise ε_{ij} is a realization of a random *expectational error* $\boldsymbol{\varepsilon}$ with zero mean. Provided that $\boldsymbol{\varepsilon}$ is uncorrelated with g_{ij} we may relax `IE2` to

`IE2'` The weights of N_G are of the form

$$w_{ij} = \begin{cases} g_{ij}/\alpha : & \text{if } \{i,j\} \in E(G), \\ 0 : & \text{if } \{i,j\} \notin E(G), \end{cases}$$

where $\alpha \leqslant 1/(3(n-1))$ is a problem dependent scaling factor.

A suitable choice of α, however, is an open problem.

4.2. *Step 2—Construction of an ε-association graph*

The problem of deciding whether two graphs G and H are ε-isomorphic can be transformed to the problem of finding a maximum clique in an ε-association graph of G and H. This concept apparently has been first suggested by Ambler [5], Barrow and Burstall [8], and Levi [24]. Since then it has been applied not only to the graph isomorphism, but also to the more general graph matching problem [9,21,22,28,29,33,35].

Let f be a p-dimensional vertex ε-invariant. The ε-association graph $G\diamond_\varepsilon H$ of graphs G and H is a graph with

$$V(G\diamond_\varepsilon H) = \{(i,j) \in V(G) \times V(H) : |f_q(G,i) - f_q(H,j)| \leqslant \varepsilon, 1 \leqslant q \leqslant p\},$$
$$E(G\diamond_\varepsilon H) = \{\{(i,k),(j,l)\} \in V(G\diamond_\varepsilon H)^{[2]} : |g_{ij} - h_{kl}| \leqslant \varepsilon, i \neq j, k \neq l\}.$$

The following Theorem justifies to cast the ε-`GIP` to the maximum clique problem in an ε-association graph. It states that G and H are ε-isomorphic if and only if $\omega(G\diamond_\varepsilon H) = n$.

Theorem 5. *Let $G\diamond_\varepsilon H$ be the ε-association graph of graphs G and H with n vertices. Then there exists a bijection*

$$\chi : \mathscr{I}_\varepsilon(G,H) \to \mathscr{C}(G\diamond_\varepsilon H)$$

from the set $\mathscr{I}_\varepsilon(G,H)$ of all ε-isomorphism from G to H and the set of all maximum cliques $\mathscr{C}(G\diamond_\varepsilon H)$.

Proof. Define $\chi(\phi) = C_\phi = \{(i,i^\phi) : i \in G(V)\}$ for all $\phi \in \mathscr{I}_\varepsilon(G,H)$. Then C_ϕ is a clique with n vertices by construction of $G\diamond_\varepsilon H$. But then C_ϕ is a maximum clique by definition of $G\diamond_\varepsilon H$. In addition, χ is well-defined. The mapping χ is bijective by construction of $\chi(\phi)$ and $G\diamond_\varepsilon H$. $\square$

It is noteworthy to mention, that in its original formulation, the maximum (maximal) cliques of an association graph are in one-to-one correspondence with the maximum (maximal) isomorphisms[4] between induced subgraphs of both graphs of

[4]The term maximum clique (subgraph, etc.) refers to the largest clique (subgraph, etc.) with respect to cardinality while the term maximal clique (subgraph, etc.) refers to the largest clique (subgraph, etc.) with respect to the subset relation '$\subseteq$'.

consideration [7]. To simplify the problem of finding a maximum clique in an association graph $G\diamond_\varepsilon H$, using ε-invariants aims at reducing the size of $G\diamond_\varepsilon H$ such that several undesirable maximal cliques disappear while all maximum cliques are maintained at the same time.

4.3. Step 3—Solving the maximum clique problem using Hopfield networks

In the previous step, we casted the problem of deciding whether G and H are ε-isomorphic to the problem of finding a maximum clique C_n in $G\diamond_\varepsilon H$ with $n = |G| = |H|$. Finding a maximum clique of a given graph is a well-known *NP*-complete combinatorial optimization problem [16]. Following the seminal paper of Hopfield and Tank [18], the general approach to solve combinatorial optimization problems maps the objective function of the optimization problem onto an energy function of a neural network. The constraints of the problem are included in the energy function as penalty terms, such that the global minima of the energy function correspond to the solutions of the combinatorial optimization problem.

Let G be a binary (undirected) graph with adjacency matrix $A(G) = g_{ij}$. To solve the maximum clique problem of G using a Hopfield clique network (HCN) $\mathscr{H}_G$ we associate the network with the graph G (see page 9).For sake of readability we restate the architecture and dynamical rule of $\mathscr{H}_G$.

The HCN $\mathscr{H}_G$ consists of n fully interconnected units. The synaptic weights w_{ij} between distinct units i and j are given by

$$w_{ij} = \begin{cases} w_E \geqslant 0 \ : \ \text{if}\{i,j\} \in E(G)\,(\text{excitation}), \\ -w_I < 0 \ : \ \text{if}\{i,j\} \notin E(G)\,(\text{inhibition}). \end{cases}$$

Note that $w_{ij} = w_{ji}$, since G is an undirected graph by assumption. The dynamical rule of $\mathscr{H}_G$ is of the form

$$x_i(t+1) = x_i(t) + \sum_{j \neq i} w_{ij} o_j(t), \tag{4}$$

where $x_i(t)$ denotes the activation of unit i at time step t. The output function $o_i(t)$ of unit i is a piecewise linear limiter function of the form

$$o_i(t) = \begin{cases} 1 \ : \ x_i(t) \geqslant \tau_t, \\ 0 \ : \ x_i(t) \leqslant 0, \\ x_i(t)/\tau_t \ : \ \text{otherwise}, \end{cases} \tag{5}$$

where τ_t is a time dependent control parameter called *pseudo-temperature*. The output function $o_i(t)$ has a lower and an upper saturation point at 0 and 1, respectively. Starting with a sufficient large initial value τ_0 the pseudo-temperature is decreased according to an *annealing-schedule* to a final value τ_f.

The energy function of the network to be minimized is then of the form

$$E(t) = -\frac{1}{2} \sum_i \sum_{j \neq i} w_{ij} o_i(t) o_j(t). \tag{6}$$

Provided an appropriate parameter setting is given, Theorem 6 states that dynamical rule (3) performs a gradient descent with respect to the energy function E where the global (local) minima of E are in an one-to-one correspondence to the maximum (maximal) cliques of G. Before stating Theorem 6, we introduce two technical terms to simplify its formulation. Let $\deg_E(i)$ be the number of excitatory connections incident to unit i. We call

$$\deg_E = \max\{\deg_E(i) \mid 1 \leqslant i \leqslant n\}$$

the *excitatory degree* of $\mathscr{H}_G$ and

$$\deg_I = \max\{n - \deg_E(i) - 1 \mid 1 \leqslant i \leqslant n\}$$

the *inhibitory degree* of $\mathscr{H}_G$. Since any vertex can have at most $n-1$ adjacent neighbors, the excitatory degree deg_E is less than n and therefore $deg_I \geqslant 0$. Now we are able to formulate Theorem 6.

Theorem 6. *Let G be a binary graph of order $|G| = n$ and $\mathscr{H}_G$ be a* HCN *associated with G. Assume that $\tau_t \geqslant 1$ for all $t \geqslant 0$. If*

$$w_E \leqslant \frac{2}{n + \deg_I(\deg_E - 1)}, \tag{7}$$

$$w_I \geqslant \deg_E w_E \tag{8}$$

then

(1) *$E(t+1) \leqslant E(t)$ for all $t \geqslant 0$.*
(2) *There is a bijection between the global (local) minima of E and the maximum (maximal) cliques of G.*

Proof. [20]. □

From the first statement together with the fact that E is bounded follows that $\mathscr{H}_G$ converges. Since the local minima of $E(t)$ correspond to maximal cliques, we cannot guarantee that the network $\mathscr{H}_G$ converges to an optimal solution corresponding to a maximum clique. In addition the network can converge to unstable points $\boldsymbol{u} \in \mathbb{R}^n$ of $E(t)$. Due to their instability, imposing random noise onto $\mathscr{H}_G$ may shift the output vector $\boldsymbol{o}(t)$ away from $\boldsymbol{u}$. An example of an unstable point is the trivial solution $\mathbf{0} \in \mathbb{R}^n$.

The upper bound of w_E ensures that $\mathscr{H}_G$ performs a gradient descent with respect to E. The lower bound of w_I guarantees that $\mathscr{H}_G$ converges to a feasible solution provided that saddle points are avoided by imposing random noise onto the network.

Given a parameter setting satisfying the bounds of Theorem 6 the HCN operates as follows: An initial activation is imposed on the network. Finding a maximum clique then proceeds in accordance with dynamical rule (3) until the system converges to a stable state. During evolution of the network any unit is excited by all active units with which it can form a clique and inhibits all other units. After convergence the stable state corresponds to a maximal weighted clique C of G. The units corresponding to the vertices of C can be identified by their respective non-negative activation.

5. Experimental results

We tested the NGI algorithm on random binary, chemical molecules, and random weighted graphs. For approximately solving the maximum clique problem in Step 3 of NGI we used the HCN algorithm presented in Section 4.3 and compared it with other clique algorithms of the neural network community: Steepest Descent (SD) [19], ρ-Annealing (RHO) [19], Mean-Field Annealing (MFA) [19], and the Exponential Replicator Equations (ERE) [28]. SD and RHO are fast algorithms which work on a discrete search space. MFA is the slowest but apparently the most accurate of all 5 clique algorithms. The replicator dynamics is derived from evolutionary game theory [17] and is despite its simplicity a powerful method to approximately solve the maximum clique problem. It uses the Motzkin–Strauss formulation of the maximum clique problem [27] and its spurious-free extensions [10]. The algorithms were implemented in Java using JDK 1.2. All experiments were conducted on a multi-server Sparc SUNW Ultra-4.

5.1. Random binary graphs

For each isomorphism test we considered pairs of graphs (G, H) where G is a randomly generated graph with n vertices and edge probability p and H is a randomly permuted version of G. The chosen parameters were $n = 100$, 500, 1000, 2500, 5000 and $p = 0.01$, 0.05, 0.1, 0.3, 0.5. We generated 100 examples for each pair (n, p) giving a total of 2500 isomorphism tests. Note, that the GIP of graphs with $p > 0.5$ is equivalent to the GIP of the complementary graphs.

The chosen invariant was the vertex invariant $\boldsymbol{x}(t)$ as proposed in Theorem 1. For sparse graphs we additionally employed the number n_i of connected components P_i consisting of i vertices for $i \in \{1, 2, 3\}$ as additional graph invariant. The components P_i are uniquely determined and form paths of length $i - 1$. Since isomorphic graphs have the same number of components, it is sufficient to compare the numbers n_i for each $i \in \{1, 2, 3\}$. If the graphs of consideration pass that test the components P_i can be discarded and isomorphism testing continues on the reduced graphs consisting of the remaining components.

We used random graphs for the purpose of comparing the proposed NGI algorithm with the best methods applied to the GIP within the neural network community. In particular, we compared our results with the *Lagrangian Relaxation Network* (LRN) [31], the *optimizing Network Architecture* (ONA) [30], and the *Exponential Replicator Dynamics* (REP) [28]. Note that REP uses the same dynamics as ERE but is applied on a different association graph using the degree of a vertex as invariant. Due to their high computational effort LRN, ONA, and REP were tested on 100-vertex random graphs only. Other neural networks approaches as [2,36] have been empirically proven to be not competitive with LRN, ONA, and REP. In addition we compared NGI with two well-known non-neural methods, *Ullman's* algorithm [38], and *Nauty* [25]. Ullman's algorithm and Nauty are both exact algorithms which guarantee to return a correct solution. Ullman's algorithm is still one of the most commonly used algorithms for graph and subgraph isomorphism problems. Nauty is considered to be the fastest graph isomorphism algorithm today available. Results on the performance of Ullman's algorithm and Nauty on random graphs are taken from [14]. Both algorithms are implemented in C/C++.

Table 1 summarizes the results. The results of ERE and HCN are roughly equivalent. Both algorithms terminated with correct results on all trials. The errors of MFA, shown in Table 1 as bracketed numbers, are all caused by exceeding the given time-limit of 10,000 iterations. The erroneous results of SD and RHO indicate that solving the maximum clique problem in Step 3 of NGI is not a trivial task.

As expected, NGI using any of the five clique algorithms SD, RHO, MFA, ERE, and HCN outperformed LRN, ONA, and REP with respect to both accuracy and speed. Accuracy of NGI using SD and RHO is roughly equivalent to LRN and superior than ONA, and REP. Accuracy of LRN, ONA, REP degraded for sparse graphs of order 100. The LRN algorithm terminated with a correct solution for all test runs except for 5% failures at $p = 0.01$. ONA and REP performed poorly on 100-vertex random graphs with $p < 0.05$. As an example for $p = 0.01$ the percentage of correct solutions is about 0.11 for REP and 0.0 for ONA. But even if we are willing to live with a small degree of uncertainty, LRN, ONA, and REP are prohibitively slow. The average times to match two 100-vertex random graphs were about 600–1800 s for LRN on a SGI workstation, about 80 s for ONA on the same SGI workstation, and about 3–2000 s for REP on a SPARC-20 workstation. In contrast, the average time required by NGI using HCN is about 0.002–0.006 s for 100-vertex graphs and 4–6 s for 5000-vertex graphs. The results indicate, that structural preprocessing might be of much more impact on combinatorial optimization problems than solely using sophisticated and powerful energy minimization methods. On the other hand, using invariants is involved with loss of generality. As opposed to LRN and ONA, the NGI algorithm as well as REP are not applicable for more general graph matching problems unless they abandon from invariants.

Ullman's algorithm and Nauty clearly outperformed LRN, ONA, and ERE. According to the results reported in [14] the average computational time of Ullman's algorithm and Nauty for sparse 100-vertex random graphs with edge probability $p \in [0.01, 0.1]$ were about 10^{-2} and 10^{-3} s on an Intel Celeron 500 MHz PC. Thus Ullman's algorithm and Nauty do not only guarantee to return a correct solution, but also are about 10^5–10^6 times faster on sparse random graphs than LRN, ONA, and ERE. These results confirm that previous neural network solutions applied to the graph isomorphism problem are not competitive with other approaches.

NGI using HCN outperformed Ullman's algorithm on random graphs. Taking into account the different programming languages and machines, the execution times of NGI and Ullman's algorithm are approximately of the same order for 100-vertex graphs. But NGI is about 100 times faster than Ullman's algorithm for 500-vertex random graphs. According to [14] Ullman's algorithm was not able to find an

Table 1
Results of isomorphism tests on random binary graphs

n	*p*	Time (s)				
		SD	RHO	MFA	ERE	HCN
	0.01	0.012	0.016	0.160	0.023	**0.006**
	0.05	$^{(6)}$0.004	$^{(8)}$ 0.009	0.091	0.011	**0.003**
100	0.1	$^{(6)}$0.006	$^{(9)}$ 0.009	0.039	0.006	**0.002**
	0.3	$^{(1)}$0.006	$^{(9)}$0.014	0.400	0.003	**0.003**
	0.5	$^{(2)}$**0.003**	$^{(3)}$0.007	0.099	0.008	**0.003**
	0.01	0.073	0.129	0.400	0.101	**0.070**
	0.05	$^{(25)}$0.052	$^{(22)}$0.087	0.407	0.090	**0.046**
500	0.1	$^{(12)}$0.368	$^{(3)}$0.649	0.058	0.074	**0.056**
	0.3	0.082	0.178	$^{[8]}$0.398	0.070	**0.050**
	0.5	0.064	0.115	1.188	0.065	**0.056**
	0.01	$^{(8)}$0.237	$^{(7)}$0.383	$^{[4]}$1.408	0.274	**0.222**
	0.05	$^{(26)}$0.188	$^{(28)}$0.303	0.518	0.379	**0.175**
1000	0.1	$^{(5)}$2.595	$^{(3)}$3.616	$^{[28]}$79.517	0.274	**0.201**
	0.3	0.278	0.672	$^{[60]}$14.272	0.274	**0.191**
	0.5	0.240	0.458	3.523	0.370	**0.222**
	0.01	$^{(24)}$1.314	$^{(8)}$1.310	—	1.344	**1.247**
	0.05	$^{(24)}$25.091	$^{(20)}$25.610	—	2.640	**1.073**
2500	0.1	$^{(4)}$13.972	13.846	—	1.288	**1.138**
	0.3	1.710	1.561	—	1.288	**1.169**
	0.5	1.490	1.492	—	1.517	**1.372**
	0.01	$^{(8)}$5.072	$^{(8)}$6.547	—	5.677	**4.999**
	0.05	$^{(36)}$**4.522**	4.701	—	5.953	4.684
5000	0.1	54.792	80.722	—	5.165	**4.514**
	0.3	6.172	7.498	—	5.576	**4.782**
	0.5	6.051	8.909	—	6.516	**5.802**

Shown are the average computational times in seconds for varying size *n* and density *p*. The numbers in parenthesis indicate the percentage error, if at least on isomorphism test failed. Bracketed numbers show the percentage of time-out errors. Best execution times are highlighted.

isomorphism for graphs with more than 700 vertices. This shows that NGI is competitive with standard algorithms for the isomorphism problem.

On the other hand Nauty is superior than NGI for large graphs. For sparse random graphs with 1000 vertices the average time required by Nauty is about 0.05 s while NGI using HCN required about 0.2 s. Even if we take into account that an algorithm implemented in Java is slower than the same algorithm implemented in C/C++, the results in [14] clearly indicate that Nauty outperformed NGI for large random graphs. The main reason for the better performance of Nauty is its more extensive and sophisticated use of vertex and edge invariants together with some efficient pruning techniques.

5.2. Chemical molecules

In this experiment, we applied the NGI algorithm on 230 chemical compounds of the mutagenicity dataset described in [37]. The number of atoms are in the range of 13–40. The average size of the data set is 25.6 and its deviation 6.7. Each molecule was transformed to an attributed graph where vertices represent atoms and edges represent bonds between the corresponding atoms. To each vertex and edge we assigned the name of the atom and the type of the bond as their attribute. For each graph representing a molecule we generated 25 randomly permuted versions of that graph giving a total of 5750 test instances.

Two experiments were conducted. In the first experiment we used the vertex invariant $f(t) = (\mathbf{x}(0), \ldots, \mathbf{x}(t))$ given in Section 4.1.1. In the second experiment we additionally used the distance (d_{ij}) between two vertices as edge invariant. Note, that the distance invariant can also be obtained by evolving a neural network.

Table 2 summarizes the results. As opposed to random graphs the structure of chemical compounds is more regular and symmetric. As expected the performance of NGI using $f(t)$ as vertex invariant is significantly worse with respect to both, accuracy and speed. Adding the edge invariant (d_{ij}) removes edges in the association graph and improves the percentage of correct results. The high accuracy of MFA is at the expense of speed. In particular, when using $f(t) + (d_{ij})$ as invariant, MFA is about 100 times slower than HCN to improve the accuracy about 0.3%. An interesting observation is that the simplest of all five algorithms, SD and RHO, performed better than HCN and ERE when using the weaker invariant $f(t)$. Finally, it is noteworthy to mention, that the NGI algorithm using HCN is about 50 times faster than the algorithm proposed by [1] applied on synthetical protein molecules of about the same size.

5.3. Random weighted graphs

For each ε-isomorphism test we generate pairs of graphs (G, H) as follows: G is a randomly generated graph with n vertices and edge probability p. To each edge $\{i, j\}$ of G we assigned a randomly chosen weight $w_{ij} \in [0, 1]$. To construct H we randomly permuted G and added uniform noise from the interval $[-\varepsilon, +\varepsilon]$ to all edges of H.

Table 2
Results of isomorphism tests on chemical molecules

	$f(t)$			$f(t) + (d_{ij})$		
	err	*acc*	*msec*	*err*	*acc*	*msec*
SD	649	88.7	0.008	24	99.6	0.008
RHO	650	88.7	0.009	18	99.7	0.009
MFA	25	99.6	0.555	0	100.0	0.573
ERE	934	83.7	0.018	104	98.1	0.016
HCN	825	83.9	0.005	7	99.9	0.004

Shown are the total number of errors (*err*), the percentage accuracy (*acc*), and the average computational time (*msec*).

The chosen parameters were $n = 100, 250, 500, 750, 1000$, $p = 0.25, 0.5, 0.75, 1.0$, and $\varepsilon = \alpha/n$ with $\alpha = 1.0$, 0.5, 0.25, 0.1. We generated 100 examples for each pair (n, α) giving a total of 8000 isomorphism tests.

We have chosen random weighted graphs to compare the proposed `NGI` algorithm with `LRN` and `ONA`, apparently the most powerful methods applied on inexact graph isomorphism problems reported in the neural network literature.

In this experiment we only have applied `ERE` and `HCN` for solving the maximum clique problem in Step 3 of the `NGI` algorithm. As invariant we used $f(t)$ as defined in Section 4.1.2.

The `NGI` algorithm using `ERE` and `HCN` returned a correct solution on all 8000 trials. Table 3 shows the average computational times in ms required by both methods for testing on ε-isomorphism. In average the `HCN` implementation is about 1.5 times faster than the `ERE` implementation. Since this is not a striking improvement we regard the results of both methods as to be roughly equivalent.

Table 3
Results of ε-isomorphism tests on random weighted graphs

n	p	ERE					HCN				
		Noise factor α					Noise factor α				
		1.0	0.75	0.5	0.25	0.1	1.0	0.75	0.5	0.25	0.1
100		11	9	8	6	4	6	5	4	4	3
250		51	50	47	42	30	30	28	27	25	21
500	0.25	204	199	186	179	133	119	115	112	108	92
750		484	470	468	458	372	281	273	268	259	225
1000		997	962	958	1006	799	549	537	531	519	448
100		9	8	7	6	5	6	5	5	4	4
250		47	44	43	42	35	34	31	30	28	24
500	0.50	190	184	181	179	157	135	129	126	121	107
750		529	520	490	459	422	325	314	303	301	265
1000		1125	1035	992	967	900	663	642	629	620	562
100		9	8	7	6	5	8	6	5	5	4
250		56	47	43	41	35	39	35	32	30	26
500	0.75	197	187	181	179	159	151	142	134	131	119
750		570	531	537	530	482	371	352	336	329	310
1000		1127	1110	1080	1078	1017	675	716	721	716	675
100		10	9	8	6	5	10	7	6	5	4
250		58	47	47	43	38	48	38	34	33	29
500	1.00	213	200	192	190	175	173	156	146	140	129
750		618	586	585	549	523	416	383	367	355	342
1000		1263	1240	1216	1217	1192	879	841	818	800	774

Shown are the average computational times in ms required by the `NGI` algorithm using `ERE` and `HCN` applied on the ε-isomorphism test on random weighted graphs.

From the results we see that the NGI algorithm performs best on sparse graphs with small noise level and worse for dense graphs with high noise level.

If the noise is sufficiently small such that it does not disrupt the structure of a graph, NGI is clearly superior than LRN and ONA. The faster of both other approaches (ONA) takes about 80 seconds to match two fully connected 100-vertex graphs, while NGI using HCN requires about 0.004–0.01 s for matching two fully connected 100-vertex graphs and about 0.4–0.9 s to match two arbitrary 1000-vertex graphs. If the noise level becomes too large ($\varepsilon > 1/n$ for uniformly distributed weights) and disrupts the structure of the graph, then the ε-GIP turns into a general graph matching problem. In this case LRN and ONA outperform NGI with respect to both accuracy and speed. An interesting observation is that the execution times of NGI approximately increase quadratic with the number of vertices. This indicates that NGI is faster on random weighted graphs than the RKHS Interpolator-Based Graph Matching algorithm [40] which has cubic complexity.

6. Conclusion

We proposed and tested a neural network approach to solve exact and inexact graph isomorphism problems based on using vertex ε-invariants. Experimental results on random weighted graphs yield exact results on all test instances within impressive time limits. The results indicate that (1) neural networks are capable of discovering structural properties in a preprocessing step, (2) neural preprocessing may be of greater impact than solely using sophisticated energy minimization methods. For large scale application problems, like identifying slightly perturbed query graphs in a data set of model graphs, it is of increasing importance to develop more powerful inexact vertex and edge invariants.

References

[1] M. Abdulrahim, M. Misra, A graph isomorphism algorithm for object recognition, Pattern Anal. Appl. 1 (3) (1998) 189–201.

[2] T. Ae, K. Agusa, S. Fujita, M. Yamashita, On neural networks for graph isomorphism problem, in: The RNNS/IEEE Symposium on Neuroinformatics and Neurocomputers, 1992, pp. 1142–1148.

[3] H.A. Almohamad, A polynomial transform for matching pairs of weighted graphs, J. Appl. Math. Model. 15 (4) (1991).

[4] H.A. Almohamad, S.O. Duffuaa, A linear programming approach for the weighted graph matching problem, IEEE Trans. Pattern Anal. Mach. Intell. 15 (5) (1993) 522–525.

[5] A.P. Ambler, H.G. Barrow, C.M. Brown, R.M. Burstall, R. J. Popplestone, A versatile computer-controlled assembly system, in: International Joint Conference on Artificial Intelligence, Stanford University, California, 1973, pp. 298–307.

[6] L. Babai, P. Erdös, S. Selkow, Random graph isomorphism, SIAM J. Comput. 9 (1980) 628–635.

[7] D.H. Ballard, C.M. Brown, Computer Vision, Prentice-Hall, Englewood Cliffs, N.J., 1982.

[8] H. Barrow, R. Burstall, Subgraph isomorphism, matching relational structures and maximal cliques, Inform. Process. Lett. 4 (1976) 83–84.

[9] S. Bischoff, D. Reuss, F. Wysotzki, Applied connectionist methods in computer vision to compare segmented images, in: A. Günter, R. Kruse, B. Neumann (Eds.), KI 2003: Advances in Artificial

Intelligence, 26th Annual Conference on AI, Lecture Notes On Artificial Intelligence 2821, Springer, Berlin, 2003, pp. 312–326.
[10] I.M. Bomze, Evolution towards the maximum clique, J. Global Optimiz. 10 (1997) 143–164.
[11] D.G. Corneil, D.G. Kirkpatrick, A theoretical analysis of various heuristics for the graph isomorphism problem, SIAM J. Comput. 9 (2) (1980) 281–297.
[12] D.G. Corneil, R.A. Mathon, Algorithmic techniques for the generation and analysis of strongly regular graphs and other combinatorial configurations, Ann. Discrete Math. 1 (1978) 1–32.
[13] N. Das, B.B. Bhattacharya, J. Dattagupta, Isomorphism of conflict graphs in multistage interconnection networks and its application to optimal routing, IEEE Trans. Comput. 42 (6) (1993) 665–677.
[14] P. Foggia, C. Sasone, M. Vento, A performance comparison of five algorithms for graph isomorphism, in: CUEN (Ed.), Third IAPR-TC-15 Workshop on Graph-based Representations in Pattern Recognition, 2001.
[15] K. Fukuda, T. Matsui, Finding all the perfect matchings in bipartite graphs, Appl. Math. Lett. 7 (1994) 15–18.
[16] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, New York, 1979.
[17] J. Hofbauer, K. Sigmund, Evolutionary Games and Population Dynamics, Cambridge University Press, UK, 1998.
[18] J.J. Hopfield, D.W. Tank, Neural computation of decisions in optimization problems, Biol. Cybernet. 52 (1985) 141–152.
[19] A. Jagota, Approximating maximum clique with a Hopfield network, IEEE Trans. Neural Networks 6 (1995) 724–735.
[20] B.J. Jain, F. Wysotzki, The maximum ω-clique problem and the Hopfield ω-clique model, submitted for publication.
[21] B.J. Jain, F. Wysotzki, A competitive winner-takes-all architecture for classification and pattern recognition of structures, in: E. Hancock, M. Vento (Eds.), Graph Based Representations in Pattern Recognition, Fourth IAPR International Workshop, GbRPR 2003, Lecture Notes in Computer Science 2726, Springer, Berlin, 2003, pp. 259–270.
[22] B.J. Jain, F. Wysotzki, A k-winner-takes-all classifier for structured data, in: A. Günter, R. Kruse, B. Neumann (Eds.), KI 2003: Advances in Artificial Intelligence, 26th Annual Conference on AI, Lecture Notes in Artificial Intelligence 2821, Springer, Berlin, 2003, pp. 342–354.
[23] M. Klin, Ch. Rücker, G. Rücker, G. Tinhofer. Algebraic combinatorics in mathematical chemistry methods and algorithms. I. Permutation groups and coherent (Cellular) algebras, Commun. Math. Comput. Chem. (1999) 1–138.
[24] G. Levi, A note on the derivation of maximal common subgraphs of two directed or undirected graphs, Calcolo 9 (1972) 341–352.
[25] B.D. McKay, Practical graph isomorphism, Congressus Numerantium 30 (1981) 45–87.
[26] H.L. Morgan, The generation of a unique machine description for chemical structures, J. Chem. Doc. 5 (1965) 107–112.
[27] T.S. Motzkin, E.G. Strauss, Maxima for graphs and a new proof of a theorem of Turan, Can. J. Math. 17 (1965) 533–540.
[28] M. Pelillo, Replicator equations, maximal cliques, and graph isomorphism, Neural Comput. 11 (8) (1999) 1933–1955.
[29] M. Pelillo, K. Siddiqi, S.W. Zucker, Matching hierarchical structures using association graphs, IEEE Trans. Pattern Anal. Mach. Intell. 21 (11) (1999) 1105–1120.
[30] A. Rangarajan, S. Gold, E. Mjolsness, A novel optimizing network architecture with applications, Neural Comput. 8 (5) (1996) 1041–1060.
[31] A. Rangarajan, E. Mjolsness, A Lagrangian relaxation network for graph matching, IEEE Trans. Neural Networks 7 (6) (1996) 1365–1381.
[32] A.C. Rao, D.V. Raju, Application of the hamming number technique to detect isomorphism among kinematic chains and inversions, Mech. Mach. Theory 26 (1) (1991) 55–75.

[33] J.W. Raymond, E.J. Gardiner, P. Willett, Heuristics for similarity searching of chemical graphs using a maximum common edge subgraph algorithm, J. Chem. Inform. Comput. Sci. 42 (2) (2002) 305–316.
[34] R.C. Read, D.G. Corneil, The graph isomorphism disease, J. Graph Theory (1977) 339–363.
[35] K. Schädler, F. Wysotzki, Comparing structures using a Hopfield-style neural network, Appl. Intell. 11 (1999) 15–30.
[36] P.D. Simic, Constrained nets for graph matching and other quadratic assignement problems, Neural Comput. 3 (1991) 268–281.
[37] A. Srinivasan, S. Muggleton, R. King, M. Sternberg, Mutagenesis: ILP experiments in a non-determinate biological domain, in: S. Wrobel (Ed.), Proceedings of the Fourth International Workshop on Inductive Logic Programming, Number 237 in GMD Studien, 1994, pp. 217–232.
[38] J.R. Ullman, An algorithm for subgraph isomorphism, J. ACM 23 (1) (1976) 31–42.
[39] S. Umeyama, An eigen decomposition approach to weighted graph matching problems, IEEE Trans. Pattern Anal. Mach. Intell. 10 (5) (1998) 695–703.
[40] B.J. van Wyk, M.A. van Wyk, T.S. Durrani, A rkhs interpolator-based graph matching algorithm, IEEE Trans. Pattern Anal. Mach. Intell. 24 (7) (2003) 988–995.
[41] J. Xu, GMA: a generic match algorithm for structural homomorphism, isomorphism, and maximal common substructure match and its applications, J. Chem. Inform. Comput. Sci. 36 (1) (1996) 25–34.
[42] L. Xu, I. King, A PCA approach for fast retrieval of structural patterns in attributed graphs, IEEE Trans. Systems, Man, Cybernet. Part B 31 (5) (2001) 812–817.

Brijnesh J. Jain received a diploma degree in mathematics and computer science. Currently he is working as a teaching and research assistant at the TU Berlin.

Fritz Wysotzki received a diploma degree in physics from Humboldt University Berlin in 1961, and the Ph.D. degree in physics from the same university in 1967. From 1982 – 1989 he was professor of the "Academy of Sciences" and head of the Department "Artificial Intelligence". Since 1994 he is Professor of Computer Science and head of the Artificial Intelligence Group at Technical University Berlin. His fields of research are Artificial Intelligence, Machine Learning, Cognitive Science and Formal Methods.