Pattern Recognition 48 (2015) 317–330

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

Efficient subgraph matching using topological node feature constraints

Nicholas Dahm [a,*], Horst Bunke [b], Terry Caelli [c], Yongsheng Gao [a]

[a] School of Engineering, Griffith University, 170 Kessels Rd, Nathan, Brisbane, QLD, Australia
[b] Institute for Computer Science and Applied Mathematics, University of Bern, Switzerland
[c] Electrical and Electronic Engineering, University of Melbourne, Australia

ARTICLE INFO

Article history:
Received 31 August 2013
Received in revised form
14 May 2014
Accepted 28 May 2014
Available online 12 June 2014

Keywords:
Graph matching
Subgraph isomorphism
Topological node features

ABSTRACT

This paper presents techniques designed to minimise the number of states which are explored during subgraph isomorphism detection. A set of advanced topological node features, calculated from *n*-neighbourhood graphs, is presented and shown to outperform existing features. Further, the pruning effectiveness of both the new and existing topological node features is significantly improved through the introduction of strengthening techniques. In addition to topological node features, these strengthening techniques can also be used to enhance application-specific node labels using a proposed novel extension to existing pruning algorithms. Through the combination of these techniques, the number of explored search states can be reduced to near-optimal levels.

© 2014 Elsevier Ltd. All rights reserved.

1. Introduction

In structural pattern recognition, graphs are used to provide meaningful representations of objects and patterns, as well as more abstract descriptions. The representative power of graphs lies in their ability to characterise multiple pieces of information, as well as the relationships between them. These graphs can be used on a variety of applications including network analysis [1], face recognition [2], and image segmentation [3]. However at the heart of graph theory is the problem of graph matching, which attempts to find a way to map one graph onto another in such a way that both the topological structure and the node and edge labels are matched. For domains where data is noisy, an identical match may not be possible, so an inexact graph matching algorithm is used to search for the closest match, minimising some similarity function. In this paper, we deal only with exact graph matching, where a match must be perfect and error-free. Exact graph matching, due to its nature, is generally used on problems where the data is precise and does not contain noise. Examples of exact graph matching problems include finding chemical similarities [4] or substructures [5], protein–protein interaction networks [6], social network analysis [7], and in the semantic web [8].

Within exact graph matching, there are three graph isomorphism problems. These are, in increasing order of complexity, graph isomorphism, subgraph isomorphism, and maximum common subgraph isomorphism. While algorithms for all of these problems have an exponential time complexity in the general case, significant advances have been made towards making these problems tractable. In this paper we focus on techniques applicable to graph and subgraph isomorphism, with particular emphasis on non-induced subgraph isomorphism. Reviews of algorithms and approaches for maximum common subgraph isomorphism can be found in [9–11]. Those looking for a complete survey of graph matching techniques are directed to [9] for the years up to 2004, and [12] for those since.

One key concept in speeding up graph and subgraph isomorphism problems is that of a *topological node feature* (TNF). A TNF is a value assigned to a node which encodes information about the local graph topology into a simple value. For example, the simplest TNF, namely the node *degree*, simply counts the number of adjacent nodes. Topological node features are also known as *subgraph isomorphism consistents* or, in the case of graph isomorphism, *invariants*.

In graph isomorphism, the early Nauty algorithm [13] by McKay was able to make significant advances beyond existing algorithms by using TNFs and a strengthening procedure similar to the tree index method that is presented in this paper. Using these techniques, Nauty is able to effectively describe the graph topology surrounding each node, eliminating mappings to any nodes where the topology is not identical. This idea was extended by Sorlin and Solnon to create the IDL algorithm [14]. Some polynomial-time algorithms have been developed for special cases of graph isomorphism, such as planar graphs [15] and bounded valence graphs [16]. A recent paper by Fankhauser et al. [17] presents a polynomial-time algorithm for graph isomorphism in the general case. Their method is suboptimal, in that some graph pairs are rejected due to unresolved permutations. However the number of rejected pairs

* Corresponding author. Tel.: +61 7 3735 3753; fax.: +61 7 3735 5198.
E-mail addresses: n.dahm@griffith.edu.au (N. Dahm), bunke@iam.unibe.ch (H. Bunke), terry.caelli@gmail.com (T. Caelli), yongsheng.gao@griffith.edu.au (Y. Gao).

http://dx.doi.org/10.1016/j.patcog.2014.05.018
0031-3203/© 2014 Elsevier Ltd. All rights reserved.

was shown to be only 11 out of 1 620 000 graph pairs (0.00068%) from the MIVIA (previously SIVA) laboratory graph database [18].

Extending such concepts to subgraph isomorphism is challenging because, while the subgraph may exist within the full graph, the additional nodes and edges in the full graph create noise for TNFs. This essentially means that instead of searching for TNFs with identical values, algorithms must ensure that TNF values from subgraph nodes are less than or equal to the corresponding TNF values from the full graph. One of the earliest and most influential subgraph isomorphism algorithms was Ullmann's algorithm [19]. Based on a tree search with backtracking, and a refinement procedure to prune the search tree, Ullmann's algorithm maintained state of the art performance until being superseded by the VF2 algorithm. The VF2 algorithm by Cordella et al. [20] has established itself as the benchmark for subgraph isomorphism algorithms due to its impressive speed, outperforming Ullmann's in almost all cases. To achieve such impressive results, VF2 takes subsets of the degree TNF, creating either two values (for undirected graphs), or six values (for directed graphs). The effect of separating the degree value into multiple smaller values is that it reduces the chance that TNF noise will prevent an invalid mapping from being detected. Despite the impressive reduction of the computation complexity provided by VF2, the exponential nature of the subgraph isomorphism problem prevents such algorithms from being practical as the number of nodes increases [21]. An updated version of Ullmann's algorithm was recently published [22], which includes a technique called *focus search*. Focus search avoids some unnecessary backup and restore operations on the bit-vector domains (representing compatible node mappings) during the search.

A number of recent papers have shown that subgraph isomorphism can be efficiently solved by utilising *constraint programming* (CP). An early example of this is the nRF+ algorithm by Larrosa and Valiente [23], which is an extension of the non-binary *really full look ahead* (nRF) algorithm. The ILF algorithm by Zampelli et al. [24] explored the use of CP with multiple TNFs. The results show that ILF outperforms VF2 on most cases, even when restricted to only the TNF of degree. Another recent CP paper is the *local all different* (LAD) algorithm by Solnon [25]. The LAD constraint ensures that for every compatible node mapping, the nodes adjacent to the subgraph node can be uniquely mapped to nodes adjacent to the full graph node. Each mapping can be validated in this way by running the Hopcroft–Karp algorithm [26]. Given x nodes adjacent to the subgraph node and y nodes adjacent to the full graph node, the complexity of validating a single mapping is $O(xy\sqrt{x+y})$ in the worst case. Despite the high computational complexity of this step, the pruning power gained from it allows the LAD algorithm to outperform even the ILF algorithm, on most cases.

This paper presents a number of techniques which can be used to speed up subgraph isomorphism through the creation, strengthening, and effective use of TNFs. The problem of subgraph isomorphism is formalised in Section 2, as are other definitions and notations used in this paper. Section 3 introduces the concept of an n-neighbourhood, and proposes some novel topological features which can be calculated from it. In Section 4, the unified framework is presented, which consists of three TNF strengthening techniques. These strengthening techniques, introduced in Sections 4.1–4.3, can be applied to TNFs, as well as application-specific node labels. Section 4.4 then shows how these concepts can be combined to create strengthened features that are resistant to noise. Similar to [21], the techniques discussed in Sections 3 and 4 are designed so that they may be utilised to enhance any subgraph isomorphism algorithm.

An earlier version of this paper appeared in [27]. In this revised and extended version, we provide richer descriptions of the pruning techniques, and employ an improved algorithm for matching nodes using the tree index, which is considerably faster than its predecessor. We also introduce the novel SINEE algorithm in Section 5, which has been specifically designed to maximise the effectiveness of topological node features, as well as the strengthening framework. To evaluate the effectiveness of SINEE, Section 6 provides a more comprehensive experimental analysis than [27], both analytically and practically. Finally, in Section 7, a number of conclusions are drawn from the experimental results, and future extensions of this work are discussed.

2. Definitions and notations

The graphs used in this paper are simple (no self-loops, no duplicate edges) unlabelled graphs. However, the adaptation of the techniques discussed in this paper to non-simple graphs is trivial.

Definition 1 (*Graph*). A graph is defined as an ordered pair $G=(V,E)$, where $V=\{v_1,\ldots,v_n\}$ is a set of vertices and $E\subseteq V\times V$ is a set of edges.

The edges of a graph may be either directed $E=\{(v_x,v_y),\ldots\}$ or undirected $E=\{\{v_x,v_y\},\ldots\}$. For clarity, undirected graphs are used to introduce new concepts. The extension of these concepts to directed graphs is also given, for cases where such an extension is non-trivial.

Subgraph isomorphism detection is performed using a depth-first search. Each state in the search tree represents a permutation of mappings.

Definition 2 (*Subgraph Isomorphism State*). A subgraph isomorphism state S is a quadruple $S=(G_f,G_s,M,A)$, where G_f is the full graph, G_s is the subgraph, M is the set of valid mappings $M=\{(v_a\in V_f\rightarrow v_b\in V_s),\ldots\}$, from full graph nodes to subgraph nodes, and A is the set of assigned mappings, such that $A\subseteq M$.

At the root of the search tree is the initial state, where $A=\varnothing$. The leaf nodes of the search tree are (possibly invalid) subgraph isomorphisms, where $|A|=|V_s|$.

3. Topological *n*-neighbourhood features

A topological node feature is defined as any feature which is calculated solely from the graph topology, as viewed from a particular node. Existing TNFs utilised for subgraph isomorphism include:

- *degree*: The number of adjacent nodes.
- *clusterc (clustering coefficient)*: The number of edges between adjacent nodes (this does not include edges to the node being evaluated).
- $ncliques_k$: The number of cliques of size k that include a particular node.
- $nwalksp_k$: The number of walks of length k that pass through a particular node.

Both ncliques_k and nwalksp_k are vectors, holding values for each different k.

An n-neighbourhood (nN) of a node v is an induced subgraph formed from all the nodes that can be reached within n steps from v. This induced subgraph is centered around node v and not only contains all nodes up to n steps away, but also contains all edges between those nodes, as depicted in Fig. 1. It is denoted as $n\text{N}(v,n)$. It should be noted that while nNs are induced subgraphs, the structure they describe can be used for both induced, and non-induced, subgraph isomorphism. For each graph node v, a unique nN may be created for each value $n=1,2,\ldots,m$, where $n\text{N}(v,m)=G$ (the

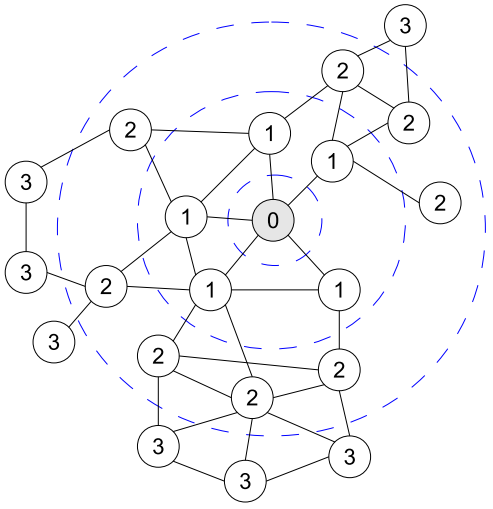

Fig. 1. Visualisation of $n\mathrm{N}(v,n)$ at $n=0,1,2,3$ for the central (grey) node. The value of n at which each node is added to the nN, is displayed for all nodes. All nodes and edges within (but not crossing) the chosen dotted line make up an nN.

entire graph can be reached in m steps). In the case of directed graphs, each nN is replaced by a pair of nNs, one for each direction of edges leaving the centre node.

Definition 3 (*n-Neighbourhood*). Given a graph G with node v, an n-neighbourhood $n\mathrm{N}(v,n)$ is a subgraph $G'=(V',E')$, where V' is the set of nodes in G that can be reached within n steps from v, and $E'=(V'\times V')\cap E$.

There are a number of TNFs that can be calculated from each nN of a node. Firstly we have the node count, or number of nodes in the nN, denoted by nN-ncount. Likewise we have the edge count, denoted by nN-ecount. Next we have the number of walks of length k in the nN, denoted nN-nwalks$_k$. Lastly we have the number of walks of length k in the nN, that pass through the main node, denoted nN-nwalksp$_k$. Each of these TNFs will give a different result for each nN of a node, giving n values, or $n\times p$ values for nN-nwalks$_k$ and nN-nwalksp$_k$ where $k=1,2,\ldots,p$. The primary benefit of calculating TNFs on nNs is the reduced likelihood of *noise* (topological structure not present in the subgraph) from distant nodes being encoded in the feature. For small values of n, the features contain less information but also less noise. On larger values of n, the amount of information encoded is higher, but so is the likelihood that noise will prevent the feature from detecting mismatches.

4. Node label strengthening framework

The node label *strengthening framework* is a collection of related techniques, designed to propagate, and encode, topological node features. At the core of the strengthening framework is the summation, listing, and tree indices, denoted SI, LI, and TI, respectively. In this order, each index utilises a higher-order data structure than the last, providing greater resolution, while taking longer to compare. Applied iteratively, this allows a single node to have distant graph topology encoded into its TNF. In addition to propagating TNFs, the listing and tree indices can also be used to propagate application-specific node labels. When applied to directed graphs, each TNF can be strengthened along either the in-edges, or the out-edges, resulting in two independent values.

A detailed description of these indices is given in the following sections. Table 1 in Section 4.3 compares the strengths and weaknesses of each index.

Table 1
A naive comparison of the strengths and weaknesses of each index.

Analysis criterion	SI	LI	TI
Calculation time	Very low	Moderate	Zero
Storage space	Very low	High	Zero
Comparison time	Very low	Low	Very high
Pruning effectiveness	Moderate	High	Very high

4.1. Summation index

The *Morgan index* [28] is a powerful TNF which calculates a hash value from a graph's topology. Originally used to characterise chemical structures, it has recently been effectively applied to graph isomorphism [17]. Despite its success in graph isomorphism, it has limited effectiveness on subgraph isomorphism.

Derived from the calculation procedure of the Morgan index, the *summation index* (SI) is proposed. The propagation technique utilised by SI is shown in Fig. 2, where the degree TNF is strengthened. Simply by taking the sum of the adjacent TNF values, each SI iteration encodes more distant topological information than the last. Given the initial node degrees in Fig. 2a, there are five unique TNF values. After only a single SI iteration, the number of unique TNF values has increased to 10. As the SI TNF values (one for each iteration) are numbers, each can be compared simply using $\leq$.

Definition 4 (*Summation index (SI)*).

$$\mathrm{SI}_i(v)=\begin{cases}\mathrm{feature}(v) & \text{if } i=0,\\ \sum_u \mathrm{SI}_{i-1}(u) & \text{otherwise.}\end{cases}$$

where u is a node adjacent to v.

As summation requires features to be added ($+$) and ranked ($\leq$), most application-specific labels cannot utilise it. To strengthen such labels, the listing or tree indices below can be used.

4.2. Listing index

The second feature strengthening technique in the framework is the *listing index* (LI). The listing index is a natural progression from summation, containing more information but also requiring more complex comparisons. Fankhauser et al. [17] presented this technique for graph isomorphism under the name *neighbourhood information*. A node's neighbourhood information is essentially a list (formally a multiset) of all TNF values from adjacent nodes.

The key difference between Fankhauser's neighbourhood information and the listing index is that the listing index evaluates each TNF separately. For graph isomorphism, this distinction is irrelevant. However for subgraph isomorphism, the difference is quite significant. As with summation, the listing index can be iteratively repeated, including more distant graph topology. Formally, the listing index of a node, at iteration i, is equal to the union of the listing indices of all adjacent nodes at $i-1$.

For Fig. 2, the listing index at iteration 1, for the centre node, would be {3, 4, 4, 4, 4}. The listing index follows the same convention as summation in that, on each iteration, only the previous values from the adjacent nodes are included, while disregarding the node's own previous value. At iteration 2, the listing index would become {1, 1, 2, 2, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5}. Taking the sum of the values in this list gives the summation index value of 69, proving that any information encoded in the summation index is also encoded in the listing index.

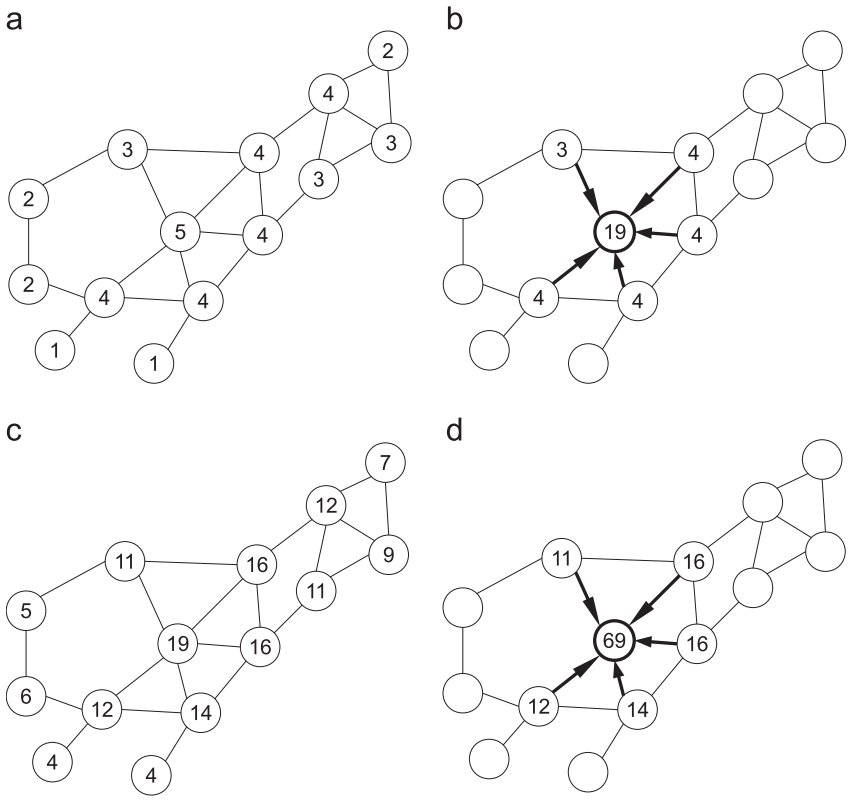

Fig. 2. The summation index propagation procedure, shown for the degree TNF. (a) Node degrees, (b) propagating degrees, (c) SI=1 node degrees, (d) propagating SI=1 degrees.

Definition 5 (*Listing index (LI)*).

$$\mathrm{LI}_i(v) = \begin{cases} \{\text{feature}(v)\} & \text{if } i = 0, \\ \cup_u \mathrm{LI}_{i-1}(u) & \text{otherwise.} \end{cases}$$

Definition 6 (*Tree index (TI)*).

$$\mathrm{TI}_i(v) = \begin{cases} \text{feature}(v) & \text{if } i = 0, \\ \cup_u \{\mathrm{TI}_{i-1}(u)\} & \text{otherwise.} \end{cases}$$

Given a full graph node v_a and a subgraph node v_b, the LI comparison technique depends on the type of values propagated. For TNF values, where the values can be ranked ($\leq$), the comparison is straightforward. First, the lists $\mathrm{LI}_i(v_a)$ and $\mathrm{LI}_i(v_b)$ must be sorted from highest to lowest. At this point, for each position k in $\mathrm{LI}_i(v_b)$, the value at position k in $\mathrm{LI}_i(v_a)$ must be greater or equal. However for application-specific node labels, where the values cannot be ranked ($\not\leq$), the comparison becomes an assignment problem. Such an assignment problem can be solved by the Hungarian algorithm [29].

4.3. Tree index

The final feature strengthening technique in the framework is the *tree index* (TI). This technique is shown in its pure form, so as to highlight the natural progression from the other indices. As its name suggests, the tree index forms a tree from the adjacent TNF values. On each iteration, the adjacent tree indices from the previous iteration are gathered into a list. This creates an iteratively deeper tree, beginning from the node and branching i layers, where i is the iteration number. For the centre node in Fig. 2, the tree index at iteration 1 is the same as LI, {3, 4, 4, 4, 4}. On iteration 2, the tree index becomes {{1, 2, 4}, {1, 4, 4}, {3, 4, 5}, …}.

This provides us with a rich description of the node's local structure, resulting in more complex comparison challenges. However since the tree created by the tree index is inherently present in the graph, there is no need to store the tree index. Instead, a tree matching algorithm is employed, which traverses the graph as if it were a tree.

Alternative versions of the tree index have existed for over 45 years [30] and have been presented for graph isomorphism [13,14], subgraph isomorphism [24], and graph kernels [31,32]. These alternative versions precompute the values and use a renaming step in an attempt to limit the size that must be stored. The effectiveness of this renaming step depends on the type of graph and can vary greatly.

4.4. Strengthening in n-neighbourhood

As mentioned in Section 3, TNFs calculated on an nN can be thought of as less noisy than their counterparts obtained on the main graph. This same concept applies equally (if not more) to the indices introduced in Sections 4.1–4.3. Instead of propagating the indices through the original graph, we can propagate them through the nN of each node. Although this means that each node's strengthened TNF values are calculated on a unique nN graph, these values are still valid to compare in subgraph isomorphism. The benefit of propagating through nNs is that it allows us to construct a very distinct picture of the local structure without

being distorted by structural information many steps from the node. The downside to this is that the structure many steps away is ignored completely, regardless of how useful such information could have been. Since nN propagation requires propagating information through each nN separately, the computation required is far more than propagation on the main graph. For each nN, the storage cost is not higher than when propagating on the main graph. However in general, the nNs and their intermediary node values are kept (to speed up recalculation), so the overall storage cost will be significantly higher.

5. SINEE algorithm

Although these neighbourhood features have been examined in detail, the questions around how to utilise them optimally are still open. To this end, we have developed efficient online pruning extensions to the existing *iterative node elimination* (INE) methods [21]. By eliminating nodes (which caused the elimination of edges), INE was able to refine TNF values, allowing their pruning power to be significantly increased. However these eliminations were all performed prior to any matching, so nodes could only be eliminated when they could not possibly be part of any subgraph isomorphism. With our proposed *subgraph isomorphism via node and edge elimination* (SINEE) algorithm, the elimination of nodes and edges is performed whenever possible throughout the entire matching process to maximise the refinement, and hence the pruning power, of TNFs. As the nN TNFs and strengthening procedures described in this paper work on the same principles as regular TNFs, they can also be refined by the elimination of nodes and edges.

The advantage to performing node and edge elimination during the matching is that the current partial matching restricts the allowable node mappings. For example, suppose we have two full graph nodes, v_a and v_b, both of which are only compatible with subgraph node v_c. If the current partial matching has assigned v_a as the match for v_c, then node v_b no longer has any valid mappings, so it can be eliminated. Once the current state backtracks so that v_a is no longer assigned to v_c, then node v_b will be restored. Such eliminations are not possible with INE, as they are conditional on the partial matching state.

5.1. Algorithm structure

As with almost all subgraph isomorphism algorithms, SINEE is based on a depth-first search with pruning. At each search tree state, a new node mapping is added to the partial matching.

An overview of the algorithm structure given in Algorithm 1. The procedure is started by passing an initial state to the SINEE() function. This function first calls the UPDATE_AND_ELIMINATE() function to refine the TNF values.

The UPDATE_AND_ELIMINATE() function updates the TNF values, then proceeds to eliminate any invalid mappings, nodes, and edges. Mappings are eliminated if the TNF values between the two nodes are no longer compatible. The methods used to eliminate nodes and edges are discussed in Section 5.2. If any mappings, nodes, or edges are deleted, the function recurses, again refining its values.

Once the revised state is returned, a *node mapping selection function* called GET_NEXT_MAPPING() is used to find a node mapping m which will be added to the current state. In many algorithms, the order in which node mappings are explored is not important. However with SINEE, when one node mapping is eliminated (after being explored), it may cause nodes, edges, or other mappings, to be deleted. More details on the node mapping selection function are given in Section 5.3.

Algorithm 1. SINEE

```
1:  procedure UPDATE_AND_ELIMINATE(state)
2:      state ← UPDATE_TNFS(state)
3:      saved_state ← state
4:      state ← ELIMINATE_INVALID_MAPPINGS(state)
5:      state ← ELIMINATE_INVALID_NODES(state)
6:      state ← ELIMINATE_INVALID_EDGES(state)
7:      if saved_state ≠ state then
8:          state ← UPDATE_AND_ELIMINATE(state)
9:      end if
10:     return state
11: end procedure
12: procedure SINEE(state)
13:     state ← UPDATE_AND_ELIMINATE(state)
14:     while m ← GET_NEXT_MAPPING() do
15:         saved_state ← state
16:         ADD_MAPPING (m, state)
17:         if state has mapped all subgraph nodes then
18:             PROCESS_ISOMORPHISM(state)
19:         else
20:             SINEE(state)
21:         end if
22:         state ← saved_state
23:         ELIMINATE_MAPPING (m, state)
24:         state ← UPDATE_AND_ELIMINATE(state)
25:     end while
26: end procedure
```

Assuming a valid node mapping m is found, the algorithm then proceeds to save the state (so it can backtrack later), then branch to a successor state by ADD_MAPPING(). If, after adding mapping m, all subgraph nodes are mapped, then the current state is an isomorphism. However if there are still some unmapped subgraph nodes, then SINEE() recurses (continues down the search tree). After isomorphism processing or the recursion is complete, the algorithm then backtracks, restoring the state from before the mapping was added. This restoration is complete, including TNF values, mappings, nodes, and edges. At this point, all possible substates which utilise mapping m have been explored, so we can remove it through ELIMINATE_MAPPING(). As this mapping elimination could lead to the elimination of more mappings, nodes, or edges, we call UPDATE_AND_ELIMINATE(). If any more valid mappings can be added, the while loop calls GET_NEXT_MAPPING() and iterates on the new mapping.

It should be noted that the above explanation, as well as Algorithm 1, do not include superfluous details. An example of such a detail is that, whenever UPDATE_AND_ELIMINATE() is called, there is a possibility that an assigned node will be deleted. In such a case, the current state becomes invalid, and the algorithm must backtrack to the previous state.

5.2. Eliminations

The conditions for eliminating mappings are trivial and consistent across most subgraph isomorphism algorithms. A mapping is deleted when it is pruned (in the case of SINEE, when the TNF values become incompatible), or when one of the nodes has already been assigned. Depending on the algorithm, conditions may also be present for eliminating nodes and edges. Definitions 7 and 8 outline these conditions for SINEE.

Definition 7 (*Node elimination conditions*). There are two conditions under which a node is eliminated. Any full graph node $v \in V_f$ is eliminated if it has no valid mappings $(v \to v_x) \notin M$, or if it has no

Table 2
Definition of MIVIA database parameters.

Subgraph ratio	Type					Graph size	
	Random	Mesh		Bounded		Small	Medium
% of full graph	Edge density	Dimensions	Irregularity	Valence	Irregularity	# nodes	# nodes
20	0.01	2	–	3	–	20	200
40	0.05	4	0.2	6	α	40	400
60	0.1	6	0.4	9		60	600
			0.6			80	800
						100	1000

edges $\{v, v_y\} \notin E_f$. This second condition is removed in applications involving non-connected subgraphs.

Definition 8 (*Edge elimination conditions*). Likewise, there are two conditions under which an edge is eliminated. Any full graph edge $\{v_a, v_b\} \in E_f$ is eliminated if one of its nodes is being eliminated, or if there is no pair of supporting mappings. A pair of mappings $(v_a \to v_c), (v_b \to v_d)$, support full graph edge $\{v_a, v_b\}$ if there is a subgraph edge, $\{v_c, v_d\}$ between their subgraph nodes.

5.3. Node mapping selection function

The node mapping selection function controls the order in which branches of the search tree are explored. Once a search tree branch has been exhausted, the corresponding node mapping is eliminated. In the case of SINEE, the elimination of any mapping can lead to the elimination of nodes and edges, as well as other mappings. These eliminations can cascade, pruning large sections of the search tree. For this reason, the choice of node mapping selection function is critical to the effectiveness of SINEE.

The SINEE node mapping selection function is inspired by that of VF2 [20]. The VF2 algorithm looks for a mapping in which both the full graph node and the subgraph node are adjacent to mapped nodes, by both an in-edge, and an out-edge. If such a mapping cannot be found, the function then searches for a mapping adjacent by either an in edge or an out edge.

The SINEE node mapping selection function instead searches for the mapping where the total number of adjacent mapped nodes is maximised. The motivation behind this choice is that this mapping has survived the most number of constraints, so is the most likely to lead to an isomorphism.

6. Experimental results

In this section, the techniques described in this paper are evaluated in detail, both analytically and practically. The type of matching performed is exact non-induced subgraph isomorphism, searching for *all* solutions. All experiments were run on an Intel® Xeon® X5650 Processor, running at 2.67 GHz, with 100 GB RAM. A time limit of 1 h per graph–subgraph pair has been enforced. If a pair cannot be solved within the time limit, the test will label it *unsolved*, record its final statistics, then move on to the next pair.

6.1. Graph dataset

The MIVIA (formerly SIVA) laboratory ARG graph database [18] is one of the most extensive publicly-available databases for graph matching. It contains synthetically-generated matched pairs for graph and subgraph isomorphism. The 54 600 graph–subgraph pairs for subgraph isomorphism are split into random, mesh, and bounded valence graph classes, containing 9000, 27 600 and 18 000 graph–subgraph pairs, respectively. The generation of these graph–subgraph pairs was directed by 546 different configurations (63 not including graph size) of the parameters in Table 2, and 100 graph–subgraph pairs were generated for each configuration.[1] The term irregularity in Table 2 refers to the relocation of edges in the graphs, such that they no longer perfectly adhere to their type category. A comprehensive explanation of the full dataset and parameters is available in [33,18].

As subgraph isomorphism has an exponential complexity, it is imperative to demonstrate how performance scales with graph size. Thus, in order to effectively analyse different TNFs and strengthening procedures, we select one configuration of each graph type, and test on all graph sizes. All tests are performed with a subgraph ratio of 60% (si6). These have the lowest number of isomorphisms, and therefore the effectiveness of pruning techniques can be more clearly identified. From the random graphs, we choose the si6_r001 subset, which have edge density 0.01. The chosen mesh graphs are the si6_m2Dr4 subset, which contain 2D mesh graphs with an irregularity of 0.4. Finally, from the bounded valence graphs, we choose the si6_b03m subset, which have valence of 3 and irregularity α (the m in b03m signifies irregularity).

6.2. Implementation details

A prototype system for the SINEE algorithm has been developed in C++. The pruning techniques described in this paper are implemented as configurable modules which can be enabled, disabled, and combined extensively. TNF values are implemented as lazy variables, which are only calculated when required.

The feasibility rules from the VF2 algorithm [20] have been interwoven into the SINEE implementation. These feasibility rules consist of the 0-look-ahead rules $R_{\text{pred}}, R_{\text{succ}}$, the 1-look-ahead rules $R_{\text{in}}, R_{\text{out}}$, and the 2-look-ahead rule R_{new}. The 1-look-ahead rules $R_{\text{in}}, R_{\text{out}}$ have been implemented as the *VF TNFs*. The benefit of such an implementation is that it allows them to be strengthened by the indices (see Section 6.4.3). The SINEE algorithm also inherently includes an injective version of the 0-look-ahead rules $R_{\text{pred}}, R_{\text{succ}}$. Bijective versions of the 0-look-ahead rules, along with the 2-look-ahead rule R_{new} are not used, as they are applicable only to *induced* subgraph isomorphism, whereas we will be performing non-induced subgraph isomorphism here.

The entire source code for the TNFs, strengthening indices, SINEE, and VF2 modules has been made publicly available.[2]

[1] Mesh graphs have dimensions of equal length, requiring graph sizes of x^2, x^3, and x^4 (where x is a positive integer) for 2D, 3D, and 4D mesh graphs, respectively. As such, their sizes may differ from those specified in Table 2.

[2] https://maxwell.ict.griffith.edu.au/cvipl/projects.html

6.3. Evaluation criteria

Since an optimal pruning algorithm is one for which the number of explored states is equal to the number of required states, we have used the following evaluation criteria. The primary evaluation criterion is the percentage of extra states for each graph–subgraph pair in a particular database subset. This measures the number of non-required (failed) states as a percentage of the number of required states. The second criterion is the computational time required to detect all subgraph isomorphisms between such graph–subgraph pairs. Next we have the maximum memory usage of a single graph–subgraph pair in the subset. Lastly, for the scalability experiments, we also include the number of graph pairs solved, as some graph pairs exceed the time limit. As the configuration experiments do not include any unsolved pairs, we instead include the matching time per state, which helps to highlight the strengths and weaknesses of different pruning techniques.

Table 3
Matching results for the SINEE algorithm on the si6_r005_s20 dataset, with various TNF configurations.

TNFs	Extra states (%)	Matching time (s)	Matching time per state (s)
No TNFs	74 340	0.90	$3.53e^{-5}$
VF TNFs	4140	0.32	$2.23e^{-4}$
Degree	7980	0.12	$4.40e^{-5}$
clusterc	52 260	7.05	$3.96e^{-4}$
nwalksp	4825	0.94	$5.38e^{-4}$

6.3.1. Graph pairs solved

For many configurations, not all graph–subgraph pairs will be solved, especially on larger graph sizes. As such, the number of solved pairs is recorded.

6.3.2. Percentage of extra states

Subgraph isomorphism detection is most commonly performed using a depth-first tree search, along with some pruning techniques. Given a full graph G_f and a subgraph G_s, there are approximately $|V_f|!/(|V_f|-|V_s|)!$ possible subgraph isomorphisms, each of which is a leaf state on the unpruned search tree. The total number of states in the entire search tree is at least $|V_s|$ higher. The number of search states explored by the matching algorithm is an effective measure of the implementation-independent computational complexity of each technique. However the number of explored states is not a stable metric unless all graph–subgraph pairs are solved.

The states explored by a subgraph isomorphism detection algorithm can be classified as either required or failed. Required states are those states which have at least one descendant that is a valid isomorphism, while all leaf states descending from failed states are invalid isomorphisms. In order to evaluate the pruning effectiveness of different algorithm configurations, the percentage of extra (failed) states is given. This metric is formally calculated as (Failed States/Required States) $\times$ 100%. So if one test explores 60 states, including 45 failed and 15 required, it can be said to have explored $45/15 \times 100\% = 300\%$ extra states. As the number of required states found approaches zero, the percentage of extra states approaches infinity ($\lim_{RS\to 0} FS/RS \times 100\% = \infty\%$). Therefore, in cases where no required states are found, we simply use an arbitrarily large value (outside of

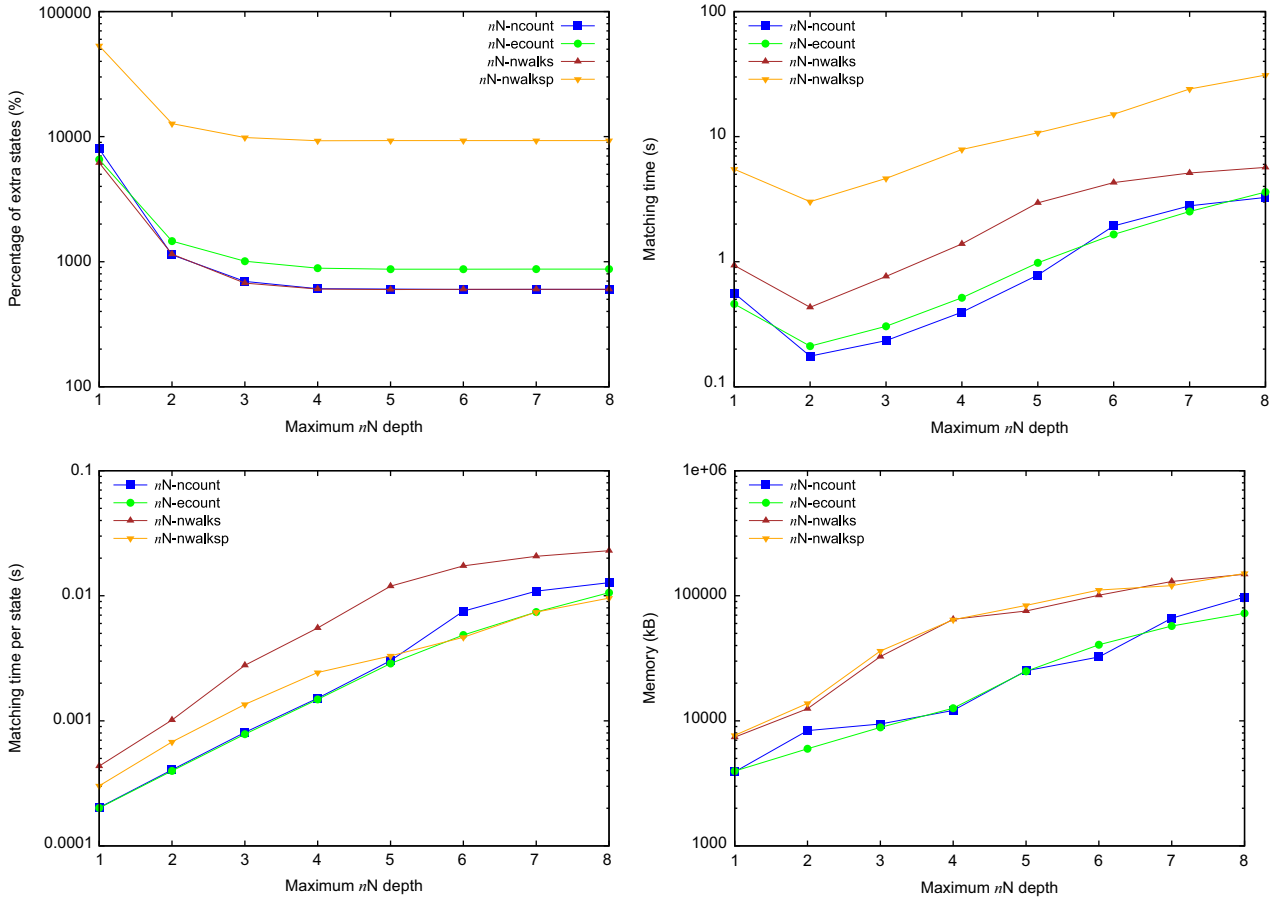

Fig. 3. Matching results as a factor of maximum nN depth, for all nN TNFs.

the plot ranges) to visually show that such cases are significantly higher than others.

6.3.3. Matching time

Naturally, the most common benchmark for subgraph isomorphism algorithms is the matching time. However the matching time is heavily dependent on the implementation, so should not be used as the sole metric in prototype systems.

6.3.4. Computation time per state

The matching time can be decomposed into the number of explored states, and the computation time required at each state. By comparing the gradients of both the total matching time, as well as the computation time per state, over a set of graphs, one can better ascertain the influences of the computation time per state and the number of explored states. This metric is formally calculated as Matching Time/Explored States.

6.3.5. Memory used

The practicality of all algorithms depend on their spatial complexity as well as computational complexity. To measure the practical memory requirements of different configurations, the peak memory usage is recorded. Note that test pairs are matched in series, so the memory value is the maximum required for a single test pair.

6.4. Configuration experiments

In this section, the pruning techniques described in this paper are evaluated, in order to determine the most effective techniques and parameters. The graphs used for these experiments are the si6_r005_s20 (random graphs with 0.05 edge density and size 20) subset. These graphs are sufficiently small enough to be solved within the time limit without using any TNFs, yet still contain enough structural complexity to showcase the effectiveness of the more advanced pruning techniques. For comparison with the techniques described in this paper, the results for SINEE with three regular TNFs, the VF TNFs, and no TNFs, are provided in Table 3.

6.4.1. n-neighbourhood TNFs

The nN TNFs are calculated on nNs of depth $n = 1, 2, \ldots, m$, where m is the maximum nN depth. Fig. 3 shows the results from each of the nN TNFs, for maximum nN depth $m = 1, 2, \ldots, 8$. The pruning power of these nN TNFs is immediately apparent, as even at just $m=2$, three of the four nN TNFs have achieved more pruning than all of the regular TNFs. As the matching time per state increases exponentially as the maximum nN depth is increased, the optimum maximum nN depth is confirmed to be two.

The poor performance of nN-nwalksp$_k$ is explained by the process of forming nNs on directed graphs. An nN is formed by following either in-edges or out-edges, but not both. This often (but not always) results in the centre node being the only root node (out-edges) or leaf node (in-edges). In such cases, no walks actually pass through the centre node, however any walks beginning or ending at it are counted. On undirected graphs, such issues are not prevalent, and nN-nwalksp$_k$ has been shown to perform better than its regular counterpart [27]. The regular TNF of clustering coefficient is afflicted by a similar issue on directed graphs.

6.4.2. Strengthening indices

The three strengthening indices, and their nN-strengthened counterparts are shown in Fig.2pc 4. Comparing the three indices,

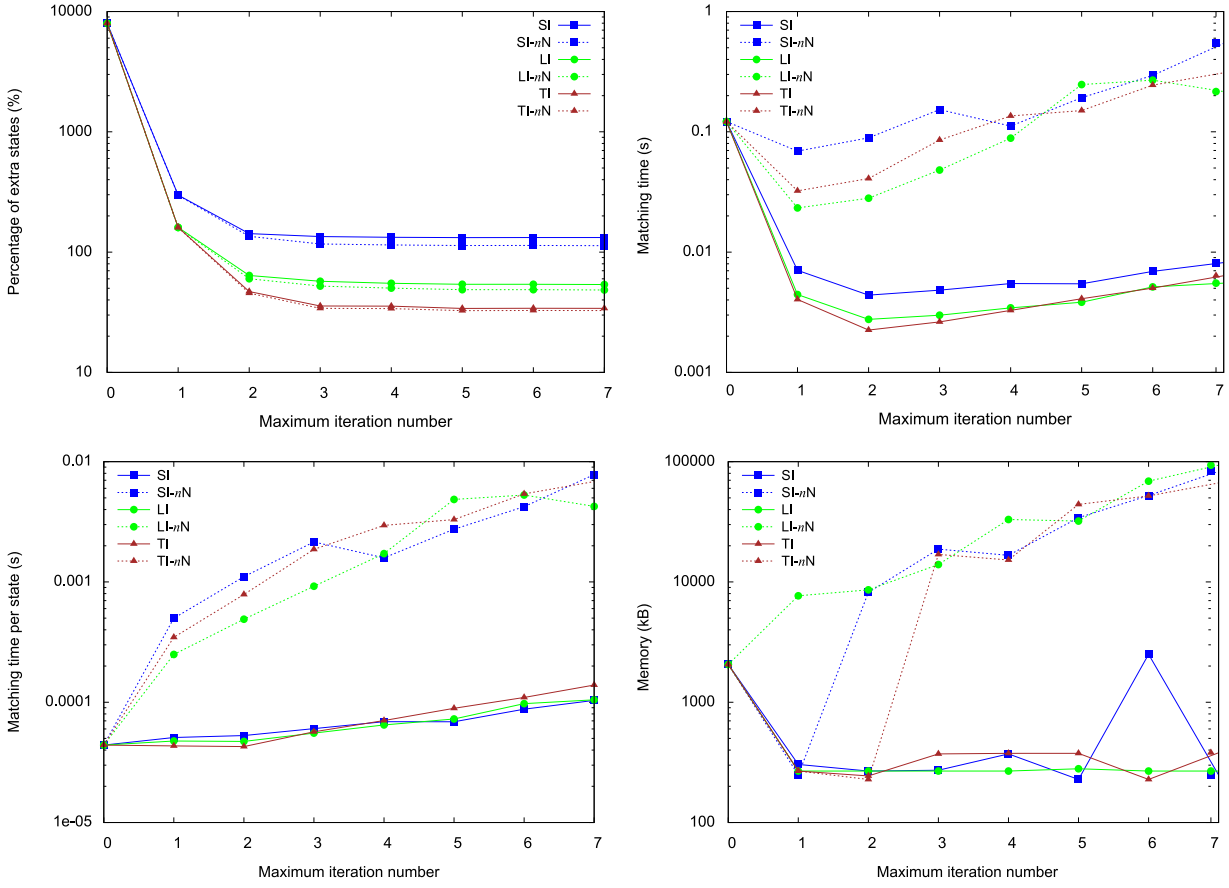

Fig. 4. Matching results as a factor of maximum iteration number, for each index, strengthening degree.

the pruning increase from SI→LI→TI is as was expected. The matching time per state however, appears to be counter-intuitive. Given that a SI comparison involves a number comparison ($>$), LI requires multiple number comparisons, and TI involves a tree search, it is surprising that the matching time per state is so similar between SI/LI/TI. Two hypotheses could explain such behaviour. The first is that search tree states near the root state are less constrained, so take longer to process. Slower but more effective pruning techniques may prune away many of these expensive states, resulting in an average mainly calculated from cheap states. The second hypothesis is that, at each state, TI reaches convergence faster than LI, which in turn converges faster than SI. This faster convergence results in lower algorithm overhead. By utilising, or drawing inspiration from, focus search [22], an updated implementation of the indices may reduce this overhead.

The pruning effectiveness of the tree index is especially significant, as it has reduced the percentage of extra states for the degree TNF from 7980% to 34%.

As expected, the nN-strengthening improves the pruning effectiveness of the indices. However, the additional pruning comes at a significant cost in terms of matching time per state, resulting in a far higher matching time.

6.4.3. Tree index on all TNFs

As the tree index has been established as the most effective index, it will be used as the primary index for all subsequent experiments. In Fig. 5, the tree index is used to propagate all of the TNFs discussed thus far. The maximum nN depth used for the nN TNFs is set to two.

The effect of the tree index is clearly evident on all TNFs. Even techniques which performed poorly on their own, such as clusterc and nN-nwalksp$_k$ were significantly improved. The improvement of nN-nwalksp$_k$ is especially noteworthy, as it shows how TNFs which appear to be ineffective, can be strengthened to the point where they become competitive.

The most effective TI-strengthened TNF, in terms of pruning power, is the VF TNFs. While initially having 4140% extra search states, the tree index was able to strengthen the VF TNFs to prune all but 12% of the extra states. This shows the true potential of the VF TNFs, and helps to explain why the VF2 algorithm has been so successful.

In terms of total matching time, degree is clearly the most efficient at 0.0022 s per graph–subgraph pair. This is due to it having the lowest matching time per state, combined with its competitive number of explored states. The pruning effectiveness of degree, combined with the fact that it is inherently calculated in many algorithms, make it an ideal TNF to add to the implementation of any subgraph isomorphism algorithm. For example, the igraph implementation of VF2 [34] uses degree in addition to the 1-look-ahead R_{in}, R_{out} rules.

Next after degree, in terms of matching time, are nN-ncount and nN-ecount, with 0.014 and 0.016 s per graph–subgraph pair, respectively. Despite being substantially slower than degree, the results show the potential of these nN TNFs. As the graph size or complexity increases, the pruning difference between degree and these nN TNFs increases. For example, on the more dense si6_r01_s20 subset, degree explores 5949% extra states, while nN-ncount explores 4415%. Given larger and more complex graphs, this pruning advantage enables these nN TNFs to effectively compete with, or combine with, cheaper TNFs like degree.

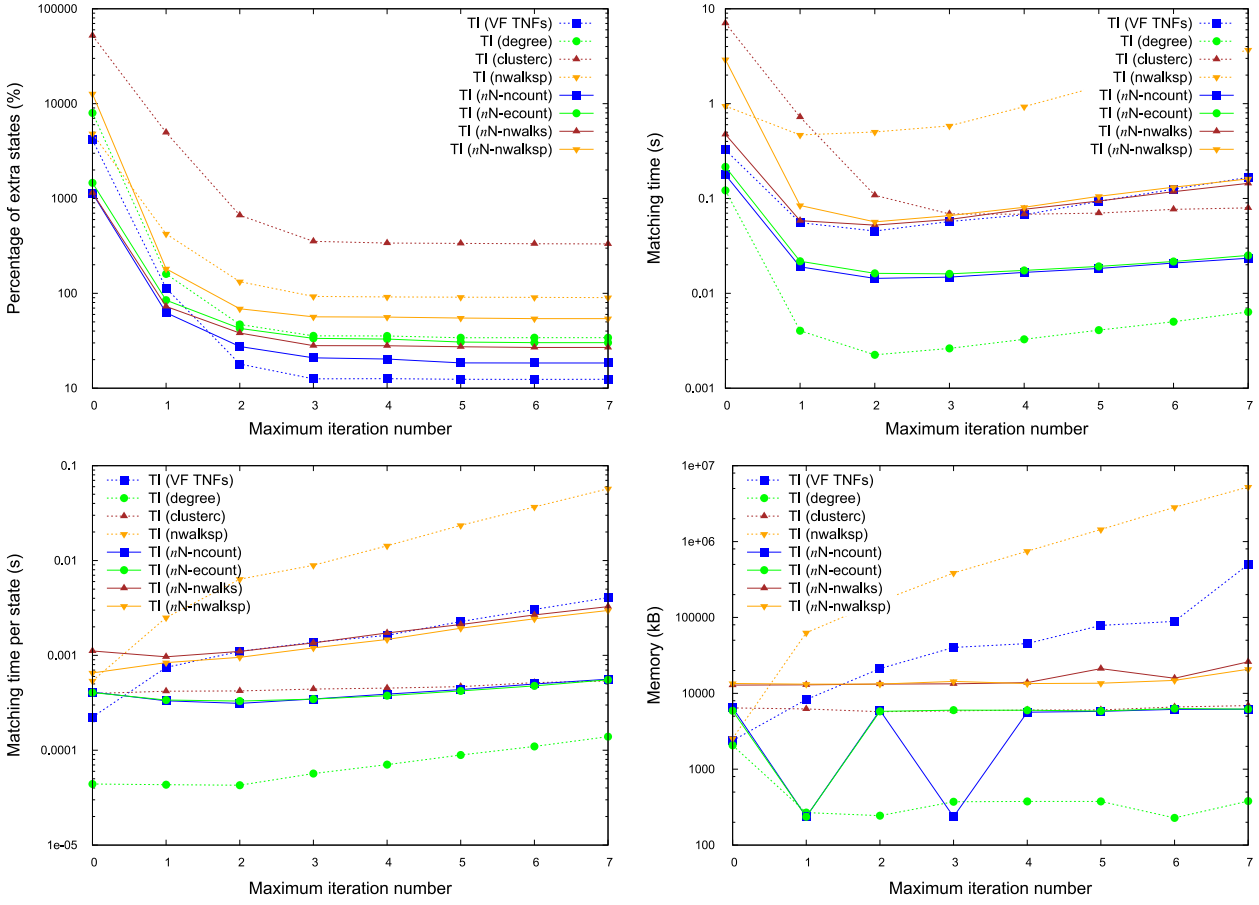

Fig. 5. Matching results as a factor of maximum iteration number, for tree index, strengthening each TNF.

Increasing the maximum iteration number has diminishing returns, with the number of explored states effectively converging at a maximum iteration number of three. Interestingly, for most TNFs, the matching time per state appears unaffected by the maximum iteration number, despite this requiring ever-deeper tree searches to confirm node compatibilities.

Table 4
Pruning configurations for practical experiments.

TNFs	1	2	3	4	5	6	7
VF TNFs	×				×		×
Degree		×			×		×
*n*N-ncount			×			×	×
*n*N-ecount				×		×	×
TI	×	×	×	×	×	×	×

Table 5
Matching results for non-strengthened TNFs on the si6_r001_s20 dataset.

TNFs	Extra states (%)	Matching time (s)	Matching time per state (s)
No TNFs	9700	0.25	$3.43e^{-5}$
VF TNFs	1240	0.16	$1.61e^{-4}$
Degree	1193	0.03	$3.50e^{-5}$
clusterc	9656	2.36	$3.24e^{-4}$
nwalksp	357	0.05	$1.38e^{-4}$
*n*N-ncount	253	0.07	$2.94e^{-4}$
*n*N-ecount	257	0.07	$2.92e^{-4}$
*n*N-nwalks	216	0.15	$6.61e^{-4}$
*n*N-nwalksp	2446	0.97	$5.32e^{-4}$

6.5. Scalability experiments

The experiments in this section evaluate the scalability, and hence the practicality, of the techniques discussed in this paper. The graphs for these experiments are the si6_r001, si6_m2Dr4, and si6_b03m subsets at *all* graph sizes contained in the MIVIA database. For each graph type and size, 100 graph–subgraph pairs are to be matched.

6.5.1. Difficulties specific to each graph type

The three different types of graphs each contain their own difficulties. For the bounded valence graphs, aside from the irregularities, all nodes have exactly the same degree. However for directed graphs, the degree TNF is generally split into *in*-degree and *out*-degree, which are not individually bounded in this database.

The mesh graphs, aside from their irregularities, are highly symmetric. Even the directed edges of these meshes flow uniformly from one corner to the opposite. As such, aside from irregularities, all nodes which are not on the borders have identical *in*-degree and *out*-degree. The strengthening indices and *n*N TNFs are invaluable in these graphs, as they help to describe nodes by their proximity to irregularities and border nodes.

The random graphs are of course the most asymmetric, however as the edge density is fixed, the average degree of nodes increases as the

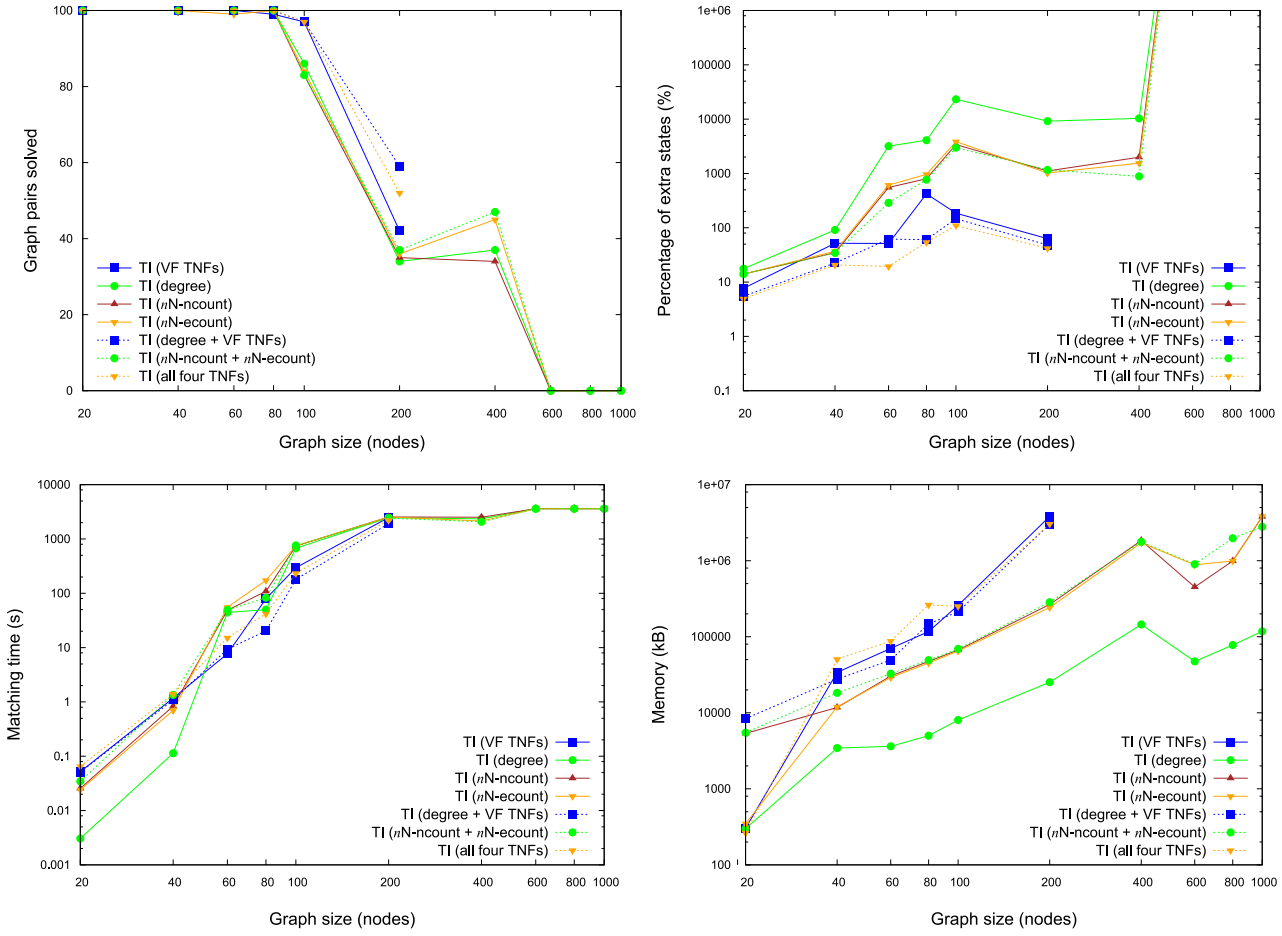

Fig. 6. *Random* graph (si6_r001) scalability results as a factor of graph size, for tree index, strengthening each TNF. (Some tests were stopped after they exceeded the 100 GB RAM limit.)

graph size increases. For a graph of size N, there are $0.05N \times (N-1)$ edges, leading to an average node degree of 1.9 for a size 20 graph, and 9.9 for a size 100 graph. This increase in node degree hinders both the discriminative power, and computation time, of the strengthening indices.

6.5.2. Definition of TNF configurations

A number of pruning configurations have been defined, combining TNFs and strengthening indices. These configurations are shown in Table 4. Only strengthened TNFs are used for these experiments, as preliminary testing has shown that non-strengthened TNFs cannot complete even the si6_r001_s40 batch within the 1 h time limit.

For comparison with the results of these experiments, the results for non-strengthened TNFs on si6_r001_s20 are given in Table 5. The maximum nN depth used in all configurations is two and the maximum tree index iteration number is four.

6.5.3. Scalability results

Figs. 6–8 show the results of the different pruning configurations on graph–subgraph pairs of increasing size from the si6_r001, si6_m2Dr4, and si6_b03m subsets, respectively. The figures clearly show the exponential nature of the subgraph isomorphism problem as the graph size increases.

When comparing the size 20 si6_r001 results in Fig. 6 with those of the non-strengthened TNFs (Table 5), the benefit of strengthening becomes clear, as each strengthened TNF has far fewer extra states to explore than every non-strengthened TNF.

Naturally those tests which utilised all four TNFs had the most effective pruning, however all configurations which included the VF TNFs were particularly effective. This is due to the fact that the VF TNFs are technically not *pure* TNFs, since they are dependent on the matching state in addition to the graph topology. All other TNFs discussed in this paper rely solely on the graph topology.

Unfortunately this dependence on the current matching state means that some VF TNF values will change (and hence be backed up) every state, instead of only when nodes or edges are eliminated. This resulted in a high memory requirement for configurations utilising the VF TNFs, to the point where experiments on the si6_r001 subset with more than 200 nodes exceeded the 100 GB memory limit. Configurations utilising nN TNFs also used considerable memory on the larger tests, however the gradient of the increase was significantly lower, similar to that of degree.

The degree TNF was, by far, the most efficient in terms of both memory usage and matching time *per state*, with all competitors using $10\times$ more memory and taking $10\times$ longer per state. However the lack of pruning power (random graphs had 91% extra states at size 40, rising to 23 141% at size 100) prevented degree from maintaining an ideal matching time. Aside from degree, the other configurations have a similar matching time per state, generally with a factor of less than $5\times$ between them all.

In terms of the total matching time, the combination of degree and VF TNFs comes out in front for larger graph sizes. These two TNFs compliment each other well, and since neither is too slow per state, the overall result is impressive.

The mesh graphs results (Fig. 7) show a significant difference between the configurations which use VF TNFs and those that do

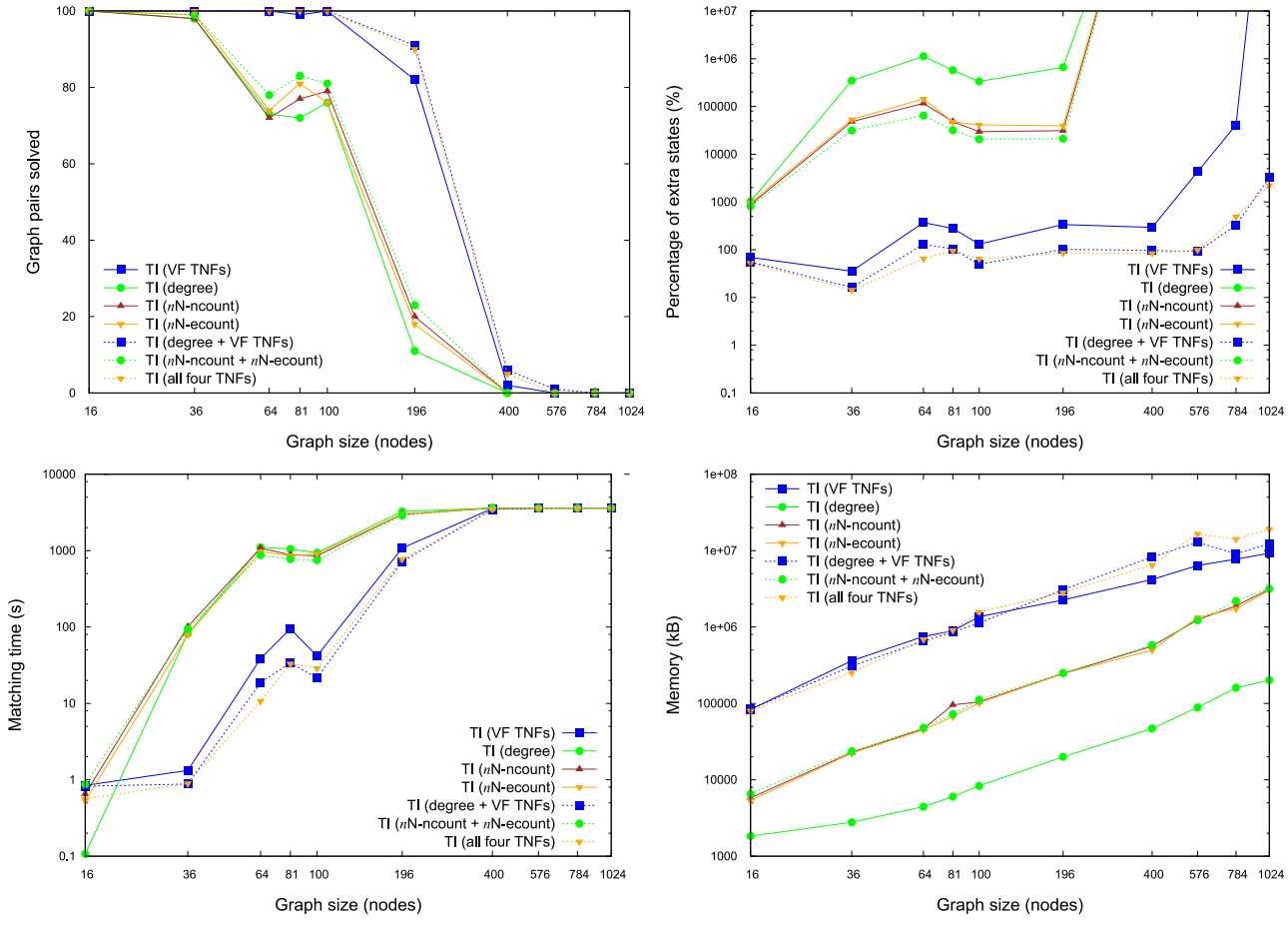

Fig. 7. *Mesh* graph (si6_m2Dr4) scalability results as a factor of graph size, for tree index, strengthening each TNF.

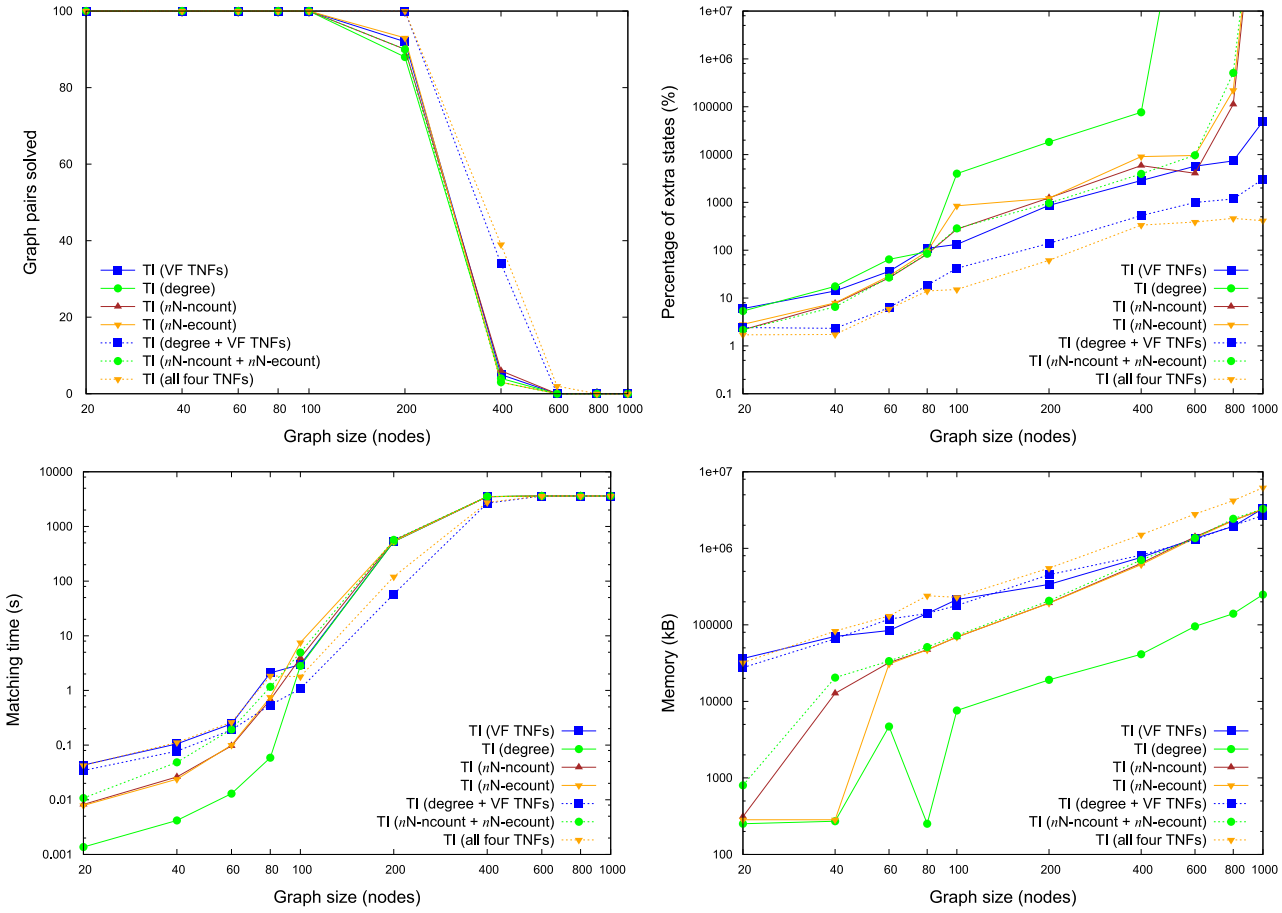

Fig. 8. *Bounded valence* graph (si6_b03m) scalability results as a factor of graph size, for tree index, strengthening each TNF.

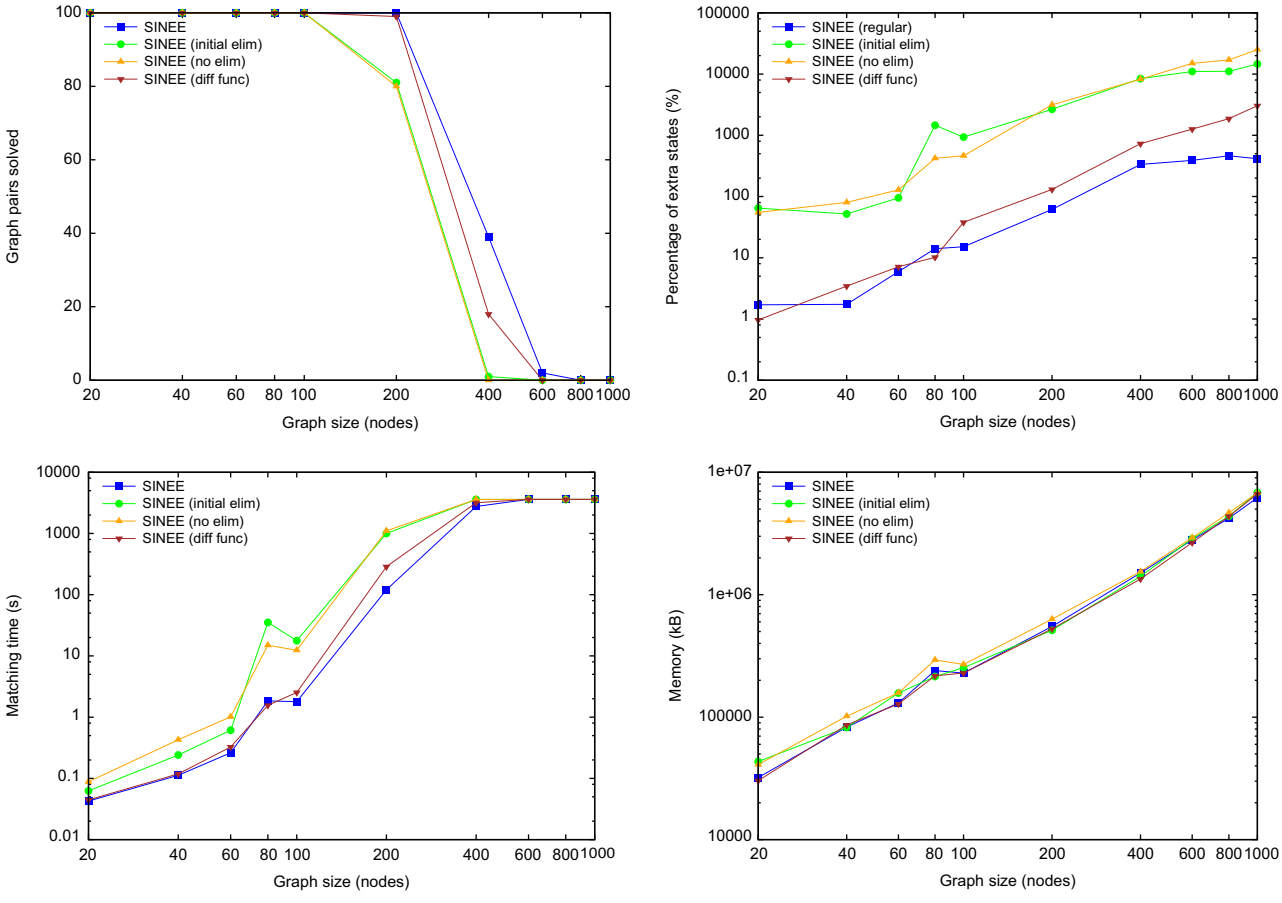

Fig. 9. *Bounded valence* graph (si6_b03m). Matching results for SINEE and subsets with initial elimination only (initial elim), with no eliminations (no elim), and without the SINEE node mapping selection function (diff func).

not. Configurations utilising the VF TNFs took almost 10 × longer per state on these graphs. However for graph sizes up to 400, they explored less than 400% extra states, while other configurations explored over 10 000% on all graph sizes larger than 16. This is likely due to the fact that the VF TNFs encode the matching state in addition to the graph structure, which is highly symmetric on mesh graphs.

6.5.4. SINEE decomposition

The experiments thus far have shown the effectiveness of the TNFs and strengthening techniques for pruning the search tree. However, as all of these experiments have utilised the SINEE algorithm, the pruning contribution of the SINEE algorithm is unclear. To investigate this contribution, Fig. 9 compares SINEE against various subsets of SINEE, each containing a restricted set of pruning modules. These SINEE subsets test SINEE without node or edge elimination, with only initial node and edge elimination (INE with edge elimination added), and without the SINEE node mapping selection function. When the SINEE node mapping selection function is not used, it is instead replaced with the standard node mapping selection function from VF2.

Observing the results in Fig. 9, the impact of the various SINEE modules is clear. The most important factor for SINEE is the online node and edge elimination, as can be seen by the relatively poor performance of the tests conducted with only initial eliminations, or with no eliminations. Initial eliminations generally improve the results over no eliminations, however in some cases the results are similar, or even marginally worse. Likewise, the SINEE node mapping selection function generally, but not always, outperforms its VF2 counterpart, especially in the larger graph sizes. It should be noted that, since node mapping selection functions use simple heuristics to determine the best search tree paths, suboptimal decisions are common. In such cases, an algorithm that uses less pruning may in fact search less states, as pure chance may give it a more optimal path. Such an example can be seen when comparing the initial eliminations to the no elimination results for graph size of 80.

Fig. 9 shows only results from the si6_b03m subset, however testing was also performed on the si6_m2Dr4 and si6_r001 subsets, which showed similar trends.

7. Conclusions

In this paper we proposed some nN topological node features, introduced strengthening techniques for TNFs, and presented a new matching algorithm designed to effectively utilise such pruning techniques. Without any strengthening techniques applied, these nN TNFs, with the exception of nN-nwalksp$_k$ (for the reasons mentioned in Section 6.4.1) were shown to have more pruning power than any of the existing TNFs.

The strengthening indices, and especially the tree index, were shown to be able to significantly improve the pruning ability of a wide range of TNFs including regular TNFs, nN TNFs, and the VF TNFs extracted from the VF2 algorithm.

Repetitive updating and maintaining of the nN-strengthened indices resulted in excessive algorithm overhead, which made them inefficient when compared to the regular strengthening indices. It should be noted that while these techniques are not as efficient as their regular counterparts, with a maximum iteration number of three or less, they still improved the total matching times for degree. Such techniques may still have considerable use in the preprocessing stage [27].

The pruning efficiency of both the nN TNFs, as well as the strengthening techniques, was significantly amplified by the novel SINEE algorithm. By utilising every opportunity to eliminate a node, edge, or mapping, the SINEE algorithm allowed these pruning techniques to be refined more frequently, resulting in pruning that would not have been possible otherwise.

This paper also highlights the importance of the node mapping selection function, a critical part which is often overlooked or marginalised. A thorough investigation into node mapping selection functions would be an important contribution to the literature.

Conflict of interest statement

None declared.

Acknowledgements

This work was supported by the Australian Postgraduate Award and National ICT Australia Research Scholarship.

References

[1] E. Estrada, M. Fox, D.J. Higham, G.-L. Oppo (Eds.), Network Science: Complexity in Nature and Technology, Springer, London, 2010.
[2] L. Wiskott, J.-M. Fellous, N. Kuiger, C. von der Malsburg, Face recognition by elastic bunch graph matching, IEEE Trans. Pattern Anal. Mach. Intell. 19 (1997) 775–779.
[3] J. Shi, J. Malik, Normalized cuts and image segmentation, IEEE Trans. Pattern Anal. Mach. Intell. 22 (2000) 888–905.
[4] P. Willett, J.M. Barnard, G.M. Downs, Chemical similarity searching, J. Chem. Inf. Comput. Sci. 38 (1998) 983–996.
[5] X. Yan, J. Han, gspan: graph-based substructure pattern mining, in: 2002 IEEE International Conference on Data Mining, 2002, pp. 721–724.
[6] H. Jeong, S.P. Mason, A.-L. Barabási, Z.N. Oltvai, Lethality and centrality in protein networks, Nature 411 (2001) 41–42.
[7] S. Wasserman, Social network analysis: methods and applications, 8, Cambridge University Press, Cambridge, 1994.
[8] B. McBride, Jena: a semantic web toolkit, IEEE Internet Comput. 6 (2002) 55–59.
[9] D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition, Int. J. Pattern Recognit. Artif. Intell. 18 (2004) 265–298.
[10] H.-C. Ehrlich, M. Rarey, Maximum common subgraph isomorphism algorithms and their applications in molecular science: a review, Wiley Interdiscip. Rev.: Comput. Mol. Sci. 1 (2011) 68–79.
[11] S. Ndiaye, C. Solnon, CP models for maximum common subgraph problems, in: J. Lee (Ed.), Principles and Practice of Constraint Programming, Lecture Notes in Computer Science, 6876, Springer, Berlin, 2011, pp. 637–644.
[12] P. Foggia, G. Percannella, M. Vento, Graph matching and learning in pattern recognition in the last 10 years, Int. J. Pattern Recognit. Artif. Intell. 28 (2014) 1450001.
[13] B.B. McKay, Practical graph isomorphism, Congr. Numer. 30 (1981) 45–87.
[14] S. Sorlin, C. Solnon, A parametric filtering algorithm for the graph isomorphism problem, Constraints 13 (2008) 518–537.
[15] J.E. Hopcroft, J.K. Wong, Linear time algorithm for isomorphism of planar graphs (preliminary report), in: Proceedings of the Sixth Annual ACM Symposium on Theory of Computing, ACM, 1974, pp. 172–184.
[16] E.M. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time, J. Comput. Syst. Sci. 25 (1982) 42–65.
[17] S. Fankhauser, K. Riesen, H. Bunke, P. Dickinson, Suboptimal graph isomorphism using bipartite matching, Int. J. Pattern Recognit. Artif. Intell. 26 (2012) 1250013.
[18] M. De Santo, P. Foggia, C. Sansone, M. Vento, A large database of graphs and its use for benchmarking graph isomorphism algorithms, Pattern Recognit. Lett. 24 (2003) 1067–1079.
[19] J.R. Ullmann, An algorithm for subgraph isomorphism, J. ACM 23 (1976) 31–42.
[20] L. Cordella, P. Foggia, C. Sansone, M. Vento, A (sub)graph isomorphism algorithm for matching large graphs, IEEE Trans. Pattern Anal. Mach. Intell. 26 (2004) 1367–1372.
[21] N. Dahm, H. Bunke, T. Caelli, Y. Gao, Topological features and iterative node elimination for speeding up subgraph isomorphism detection, in: Proceedings of the 21st International Conference on Pattern Recognition, 2012, pp. 1164–1167.
[22] J.R. Ullmann, Bit-vector algorithms for binary constraint satisfaction and subgraph isomorphism, J. Exp. Algorithmics 15 (2011) 1.6:1.1–1.6:1.64.
[23] J. Larrosa, G. Valiente, Constraint satisfaction algorithms for graph pattern matching, Math. Struct. Comput. Sci. 12 (2002) 403–422.
[24] S. Zampelli, Y. Deville, C. Solnon, Solving subgraph isomorphism problems with constraint programming, Constraints 15 (2010) 327–353.
[25] C. Solnon, All different-based filtering for subgraph isomorphism, Artif. Intell. 174 (2010) 850–864.
[26] J. Hopcroft, R. Karp, An n5/2 algorithm for maximum matchings in bipartite graphs, SIAM J. Comput. 2 (1973) 225–231.
[27] N. Dahm, H. Bunke, T. Caelli, Y. Gao, A unified framework for strengthening topological node features and its application to subgraph isomorphism

detection, in: W.G. Kropatsch, N.M. Artner, Y. Haxhimusa, X. Jiang (Eds.), Graph-Based Representations in Pattern Recognition, Lecture Notes in Computer Science, 7877, Springer, Berlin, 2013, pp. 11–20.
[28] H.L. Morgan, The generation of a unique machine description for chemical structures—a technique developed at chemical abstracts service, J. Chem. Doc. 5 (1965) 107–113.
[29] H.W. Kuhn, The Hungarian method for the assignment problem, Naval Res. Logist. Q. 2 (1955) 83–97.
[30] B. Weisfeiler, A.A. Lehman, A reduction of a graph to a canonical form and an algebra arising during this reduction, Nauchno-Tech. Inf. (1968) 12–16 (in Russian).
[31] N. Shervashidze, K.M. Borgwardt, Fast subtree kernels on graphs, in: Y. Bengio, D. Schuurmans, J. Lafferty, C.K.I. Williams, A. Culotta (Eds.), Advances in Neural Information Processing Systems, vol. 22, 2009, pp. 1660–1668.
[32] N. Shervashidze, P. Schweitzer, E.J. van Leeuwen, K. Mehlhorn, K.M. Borgwardt, Weisfeiler–Lehman graph kernels, J. Mach. Learn. Res. 12 (2011) 2539–2561.
[33] P. Foggia, C. Sansone, M. Vento, A database of graphs for isomorphism and sub-graph isomorphism benchmarking, in: Proceedings of the 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, 2001, pp. 176–187.
[34] G. Csardi, T. Nepusz, The igraph software package for complex network research, InterJ. Complex Syst. (2006) 1–9.

Nicholas Dahm is a Ph.D. student in the School of Engineering, Griffith University, Australia. He received his B.IT (Hons.) in 2009 from Griffith University. His primary research areas are Computer Vision and Graph Matching.

Horst Bunke received his Ph.D in 1979 from the University of Erlangen, Germany. Since then he has served on numerous editorial boards, received the King-Sun Fu Prize in 2010, and is ranked as one of the 150 most prolific authors, according to the DBLP Computer Science Bibliography. Currently, he is an emeritus Professor at the University of Bern, Switzerland.

Terry Caelli received his Ph.D in 1975 from the University of Newcastle, Australia. He has been a prominent member of the research community and has held Professor positions in Germany, Canada, and Australia. His research interests include computer vision and machine learning for biomedical technologies and environmental monitoring.

Yongsheng Gao received B.Sc. and M.Sc. degrees in Electronic Engineering from Zhejiang University, China, in 1985 and 1988, respectively, and a Ph.D. degree in Computer Engineering from Nanyang Technological University, Singapore. Currently, he is a Professor at the School of Engineering, Griffith University, Australia. His research interests include face recognition, biometrics, image retrieval, computer vision, and pattern recognition.