Indexing with Well-Founded Total Order for Faster Subgraph Isomorphism Detection

Markus Weber[1,2],
Marcus Liwicki[1,2], and Andreas Dengel[1,2]

[1] Knowledge Management Department,
German Research Center for Artificial Intelligence (DFKI) GmbH
Trippstadter Straße 122, 67663 Kaiserslautern, Germany

[2] Knowledge-Based Systems Group, Department of Computer Science,
University of Kaiserslautern, P.O. Box 3049, 67653 Kaiserslautern

{firstname.lastname}@dfki.de

Abstract. In this paper an extension of index-based subgraph matching is proposed. This extension significantly reduces the storage amount and indexing time for graphs where the nodes are labeled with a rather small amount of different classes. In order to reduce the number of possible permutations, a weight function for labeled graphs is introduced and a well-founded total order is defined on the weights of the labels. Inversions which violate the order are not allowed. A computational complexity analysis of the new preprocessing is given and its completeness is proven. Furthermore, in a number of practical experiments with randomly generated graphs the improvement of the new approach is shown. In experiments performed on random sample graphs, the number of permutations has been decreased to a fraction of 10^{-18} in average compared to the original approach by Messmer. This makes indexing of larger graphs feasible, allowing for fast detection of subgraphs.

Keywords: Graph isomorphism; Subgraph isomorphism; Tree search; Decision tree; Indexing

1 Introduction

Graphs play a major role in structural pattern recognition. An important task in this field is to find similar structures (error-tolerant graph matching) or the same structure (exact graph matching). The focus of this paper is on the latter task, which is important if exactly the same structure or sub-structure needs to be retrieved.

Exact graph matching is needed when the user searches for specific constellations in molecules [11], in computer vision for the recognition of 3-D objects [8, 14], shape matching in image analysis [6, 2], or room-constellations in floor plans [13]. In most applications, the retrieval result should be available in real-time and the database of reference structures does not change too often. For

those situations it is advisable to build an index of the reference structures in advance.

Such a method has been proposed by Messmer et al. [9]. It builds an index using the permutated adjacency matrix of the graph. The real-time search is then based on a tree based. While the method has shown to be effective for reference set with small graphs, it is infeasible for graphs with more than 19 vertices.

In this paper we propose a method to overcome this problem. Assuming that the number of labels for the nodes is relatively small, we introduce a well-founded total order and apply this during index building. This optimization decreases the amount of possible permutations dramatically and allows building indexes of graphs with even more than 30 vertices.

The rest of this paper is organized as follows. First, Section 2 gives an overview over related work. Subsequently, Section 3 introduces definitions and notations which are used and Section 3.1 describes the new preprocessing step. Next, Section 4 will show that the number of computational steps will be significantly decreased. Finally, Section 5 concludes the work.

2 Related Work

In [7], Goa et al. give a survey of work done in the area of graph matching. The focus in the survey is the calculation of error-tolerant graph-matching; where calculating a graph edit distance (GED) is an important way. Mainly the GED algorithms described are categories into algorithms working on attributed or non-attributed graphs. Ullman's method [12] for subgraph matching is known as one of the fastest methods. The algorithm attains efficiency by inferentially eliminating successor nodes in the tree search.

Bunke [3, 4] discussed several approaches in graph-matching. One way to cope with error-tolerant subgraph matching is using the maximum common subgraph as a similarity measure. Furthermore the application of graph edit costs which is an extension of the well-known string edit distances. A further group of suboptimal methods are approximate methods, they are based on neural networks, such Hopfield network, Kohonen map or Potts MFT neural net. Moreover methods like genetic algorithms, the usage of Eigenvalues, and linear programming are applied.

Graph matching is challenging in presence of large databases [1, 4]. Consequently, methods for preprocessing or indexing are essential. Preprocessing can be performed by graph filtering or concept clustering. The main idea of the graph filtering is to use simple features to reduce to number of feasible candidates. Another concept clustering is used for grouping similar graphs. In principle, given a similarity (or dissimilarity) measure, such as GED [5], any clustering algorithm can be applied. Graph indexing can be performed by the use of decision trees.

Messmer and Bunke [9] proposed a decision tree approach for indexing the graphs. They are using the permutated adjacency matrix of a graph to build a decision tree. This technique is quite efficient during run time, as a decision tree is

generated beforehand which contains all model graphs. However, the method has to determine all permutations of the adjacency matrices of the search graphs. Thus, as discussed in their experiments, the method is practically limited to graphs with a maximum of 19 vertices. The main contribution of this paper is to improve the method of Messmer and Bunke for special graphs by modifying the index building process.

3 Definitions and Notations

Basic definitions used throughout the paper are already defined in [9], such as a labeled graph $G = (V, E, L_v, L_e, \mu, \upsilon)$, adjacency matrix ($M$), and permutations on adjacency matrices ($A(G)$). Besides, definitions for orders on sets are needed.

Definition 1 *A **total order** is a binary relation $\leq$ over a set P which is transitive, anti-symmetric, and total, thus for all a, b and c in P, it holds that:*

- *if $a \leq b$ and $b \leq a$ then $a = b$ (anti-symmetry);*
- *if $a \leq b$ and $b \leq c$ then $a \leq c$ (transitivity);*
- *$a \leq b$ or $b \leq a$ (totality).*

Definition 2 *A partial or total order $\leq$ over a set X is **well-founded**, iff $(\forall\ Y \subseteq X\ :\ Y \neq \emptyset \rightarrow (\exists y \in Y\ :\ y$ minimal in Y in respect of $\leq))$.*

Additionally, a weight function is defined which assigns weight to a label of a graph.

Definition 3 *The **weight function** σ is defined as: $\sigma\ :\ L_v\ \rightarrow\ \mathbb{N}$.*

Using the weight function, a well-founded total order is defined on the labels of graph, for example $\sigma(L_1) < \sigma(L_2) < \sigma(L_3) < \sigma(L_4)$. Thus the labeled graph can be extended in its definition.

Definition 4 *A **labeled graph** consists of a 7-tuple, $G = (V, E, L_v, L_e, \mu, \upsilon, \sigma)$, where*

- *V is a set of vertices,*
- *$E \subseteq V \times V$ is a set of edges,*
- *L_v is a set of labels for the vertices,*
- *L_e is a set of labels for the edges,*
- *$\mu : V \rightarrow L_v$ is a function which assigns a label to the vertices,*
- *$\upsilon : E \rightarrow L_e$ is a function which assigns a label to the edges,*
- *$\sigma : L_v\ \rightarrow\ \mathbb{N}$ is a function which assigns a weight to the label of the vertices,*

and a binary relation $\leq$ which defines a well-founded total order on the weights of the labels:

$$\forall x, y \in L_v\ :\ \sigma(x) \leq \sigma(y)\ \vee\ \sigma(y) \leq \sigma(x)$$

3.1 Algorithm

The algorithm for subgraph matching is based on the algorithm proposed by Messmer and Bunke [9], which is a decision tree approach. Their basic assumption is that several graphs are known a priori and the query graph is just known during run time. Messmer's method computes all possible permutations of the adjacency matrices and transforms them into a decision tree. At run time, the adjacency matrix of the query graph is used to traverse the decision tree and find a subgraph which is identical.

Let $G = (V, E, L_v, L_e, \mu, \upsilon)$ be a graph from the graph database and M the corresponding $n \times n$ adjacency matrix and $A(G)$ the set of permuted matrices. Thus the total number of permutations is $|A(G)| = n!$, where n is the dimension of the permutation matrix, respectively the number of vertices.

Now, let $Q = (V, E, L_v, L_e, \mu, \upsilon)$ be a query graph and M' the corresponding $m \times m$ adjacency matrix, with $m \leq n$. So, if a matrix $M_P \in A(G)$ exists, such that $M' = S_{m,m}(M_P)$, the permutation matrix P which corresponds to M_P represents a subgraph isomorphism from Q to G, i.e

$$M' = S_{m,m}(M_P) = S_{m,m}(PMP^T).$$

Messmer proposed to arrange the set $A(G)$ in a decision tree, such that each matrix in $A(G)$ is classified by the decision tree. However, this approach has one major drawback. For building the decision tree, all permutations of the adjacency matrix have to be considered. Thus, for graphs with more than 19 vertices the number of possible permutations becomes intractable. In order to overcome this issue, the possibilities of permutations have to be reduced. One way is to define constraints for the permutations. Therefore a weight function σ (see Definition 3) is introduced which assigns a weight for each vertex according to its label. Thus each label has a unique weight and a well-founded total order (see Definition 1 and Definition 2) on the set of labels which reduces the number of allowed inversion for the adjacency matrix. Figure 1 illustrates an example for the modified matrices and the corresponding decision tree. Let us consider the following weights for the nodes:

$$\begin{aligned} L_v &= \{L_1, L_2, L_3\} \\ \sigma(L_1) &= 1, \\ \sigma(L_2) &= 2, \\ \sigma(L_3) &= 3. \end{aligned}$$

Each inversion that violates the ordering is not allowed. Thus just the vertices which have the same label, respectively the same weights, have to be permuted and if the labels have a different weight, just the variations are required. Given

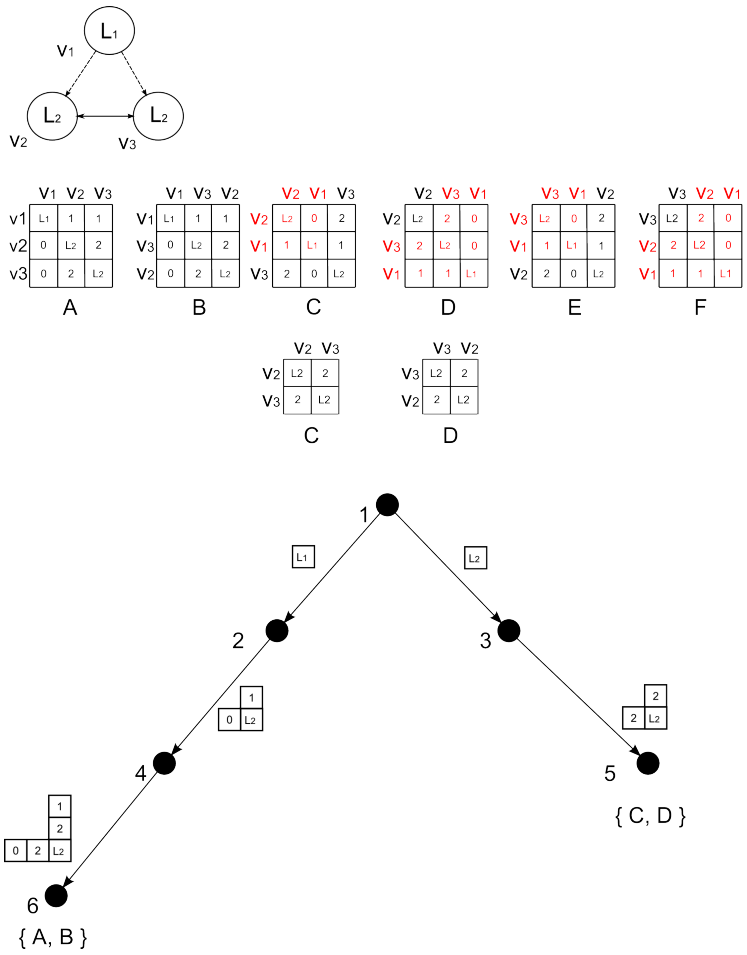

Fig. 1: Modified decision tree for adjacency matrices

the graph G, the following labels are assigned to the vertices,

$$\begin{aligned} V =& \{v_1, v_2, v_3\} \\ \mu(v_1) =& L_1, \\ \mu(v_2) =& L_2, \\ \mu(v_3) =& L_2. \end{aligned}$$

Hence, the only valid permutations are:

1. $\sigma(\mu(v_1)) \leq \sigma(\mu(v_2)) \leq \sigma(\mu(v_3))$
2. $\sigma(\mu(v_1)) \leq \sigma(\mu(v_3)) \leq \sigma(\mu(v_2))$
3. $\sigma(\mu(v_2)) \leq \sigma(\mu(v_3))$
4. $\sigma(\mu(v_3)) \leq \sigma(\mu(v_2))$

Let $VA(G)$ be the set of all valid permutations. The decision tree is built according to the row-column elements of the adjacency matrices $M_P \in VA(G)$ and should cover all graphs from the database. So, let R be the set of semantics $R = \{G_1, G_2, ..., G_n\}$, where n is the total number of graphs in the repository, with their sets of corresponding adjacency matrices $VA(G_1)$, $VA(G_2)$, ..., $VA(G_n)$. Now, each set of adjacency matrices has to be added to the decision tree. An obvious advantage of the method is that the whole process can be done a priori. The decision tree acts as an index for subgraphs. So, during run time the decision tree has been loaded into memory and by traversing the decision tree, the corresponding subgraph matrices are classified. For the query graph the adjacency matrix is determined following the constraints defined by ordering. Afterwards the adjacency matrix is split up into row-column vectors a_i. For each level i the corresponding row-column vector a_i is used to find the next node in the decision tree using an index structure. As query q_1 ends in a leaf of the decision tree, the labels of the leaf are the results, query q_2 stops in a node, thus the labels of all leafs beneath the node combine the result.

3.2 Proof of Completeness

For the proposed modified algorithm it has to be proven that the algorithm finds all solutions. The algorithm elaborated in the previous section reduces the number of valid permutations. So, it has to be shown that by leaving out permutations, no valid solution is lost.

Let $G = (V, E, L_v, L_e, \mu, \upsilon, \sigma)$ be a well-founded total ordered graph and let $A(G)$ be the set which contains all valid permutations of the graph's adjacency matrices. To be complete, the algorithm must find a solution if one exists; otherwise, it correctly reports that no solution is possible. Thus if every possible valid subgraph $S \subseteq G$, where the vertices of S fulfill the order, every corresponding adjacency matrix M has to be an element of the set $A(G)$, $M \in A(G)$.

For this reason to proof that the algorithm is complete it has to be shown that the algorithm generates all valid subgraphs $S \subseteq G$. Therefore the pseudo code of Algorithm 1 shows how the index is build. Algorithm 2 and Algorithm 3 are helping functions for calculating all variations of the set of vertices in an interval. The generation of the index starts with an unsorted set of vertices. By sorting the vertices with their associated labels using the well-founded total order, the set is ordered according to the weights of the labels.

Now, the algorithm iterates over all intervals of vertices $\{v_a, ..., v_b\}$ where the labels have the same weights, $\sigma(\mu(v_a)) == \sigma(\mu(v_b))$. For each interval $\{v_a, \ldots, v_b\}_i$ all variations with respect to the order have to be determined. These variations are computed in Algorithm 4, by determining all combination of the interval $\{v_a, \ldots, v_b\}_i$ including the empty set and calculating all permutations for these combinations. Algorithm 2 and Algorithm 3 realize the algorithm proposed by Rosen [10] which computes all permutations for a defined interval. It has been proven that Rosen's algorithm computes all permutations. In combinatorial mathematics, a k-variation of a finite set S is a subset of k distinct elements of S. For each chosen variation of k elements,

Algorithm 1 BUILD_INDEX($G = (V, E, L_v, L_e, \mu, \upsilon, \sigma)$, Tree)

Require: Unsorted set V of vertices, μ labeling function, σ weight function.
sort(V, L_v, μ, σ)
Ensure: Vertices V are sorted according to the defined order.
Let O be an empty list.
for all $l_i \in L_V$ **do**
    Let interval $\{v_a, \ldots, v_b\}$ contain all v with $\mu(v) = l_i$
    $O_i \leftarrow VARIATIONS(\{v_a, \ldots, v_b\})$
end for
Let $AG \leftarrow O_1 \times \ldots \times O_{|Lv|}$.
for all m_i in AG **do**
    Add row column vector for sequence of m_i to Tree.
end for

Algorithm 2 $PERMUTE(V, begin, end, R)$

Require: Sorted set V of vertices and $begin < end$, with V_{end-1} being last the element.
Adding sequence of vertices V to R.
for $i \leftarrow end - 2$ to $begin$ **do**
    for $j \leftarrow i + 1$ to $end - 1$ **do**
        Swapping position i and j in V.
        Call $PERMUTE(V, i + 1, end, R)$.
    end for
    Call $ROTATE(V, i + 1, end, R)$.
end for

where k is $L_{interval} =$ length of interval; $k = 1 \ldots L_{interval}$, again all permutations have to be considered. Now, assuming there would be a valid subgraph $Q = (V', E', L'_v, L'_e, \mu, \upsilon, \sigma)$, respectively the corresponding adjacency matrix A which depends on the alignment of the vertices. To be a valid subgraph, V' has to be a subset of V, $V' \subseteq V$. Furthermore the alignment of the vertices V' according to their labels has to fulfill the defined order, $\sigma(\mu(v_i)) \leq \sigma(\mu(v_{i+1}))$. For the alignment the intervals $\{v'_a, \ldots, v'_b\} \in V'$ where the weights of the labels have the same value $\sigma(\mu(v'_a))) == \sigma(\mu(v'_b)$ are important as they can vary. The Algorithm 4 determines all variations for intervals with the same weights for labels, thus the alignment $\{v'_a, \ldots, v'_b\}$ is considered. This holds for each interval, thus algorithm produces all valid permutations according to the well-founded total order. As the query graph Q also has to fulfill the order, its adjacency matrix A will be an element of $A(G)$, if Q is a valid subgraph of G. Thus, the solution will be found in the decision tree.

3.3 Complexity Analysis

The original algorithm by Messmer [9] as well as the proposed algorithm need an intensive preprocessing, the compilation of the decision tree. Messmer's method has to compute all permutations of the adjacency matrix of the graph, thus the

Algorithm 3 $ROTATE(V, begin, end, R)$

1: Let $temp \leftarrow V_{end-1}$.
2: Shift elements in V in from position *begin* to $end - 1$ one position right
3: Set $V_{begin} \leftarrow temp$.
4: Add sequence of vertices V to R.

Algorithm 4 VARIATIONS($\{v_a, \ldots, v_b\}$)

Require: Sorted set $V = \{v_a, \ldots, v_b\}$ of vertices, $a \leq b$.
1: Let O be an empty list.
2: Determine all combinations C for $\{v_a, \ldots, v_b\}$ including the empty set.
3: **for all** c in C **do**
4: Call $PERMUTE(c, 0, |c|, O)$.
5: **end for**
6: Return O.

compilation of the decision tree for a graph $G = (V, E, L_v, L_e, \mu, \upsilon, \sigma)$ has a run time complexity of $\mathcal{O}(|V|!)$.

Due to space limitations, we omit the detailed listing of all calculations. The final result for the complexity of our proposed approach is

$$\mathcal{O}(((n_{max} + 1)!)^{|L_v|}),$$

where n_{max} is the maximum number of vertices with the same weight. Thus for the worst case - where all vertices have the same label - $n_{max} = |V|$, $\mathcal{O}((|V|+1)!)$ which would be worse than the method proposed by Messmer and the best case - where all vertices have different labels ($n_{max} = 1$) is $\mathcal{O}(2^{|V|})$ To find the average case of the algorithm the distribution of the labels in the graph has to be considered. This distribution varies according to the represented data.

4 Evaluation

In order to examine run time efficiency of the modified subgraph matching experiments on randomly generated graphs were performed. The modified decision tree algorithm has been implemented in Java using a Java 6 virtual machine. The experiments ran on a Intel Core Duo P8700 (2.53 GHz) CPU with 4 GByte main memory. For the experiment 100 random graphs were generated with 15 to 30 vertices. It compares Messmers's algorithm with its required permutations to the modified algorithm. The permutations for the modified algorithm were determined according to the algorithm discussed in Section 3.1 and the formula in Section 3.3:

$$\prod_{i=1}^{|L_v|} \left(\sum_{j=1}^{n_i} \binom{n_i}{j} \cdot j! \right)$$

and as the original has to be calculate the permutations for all vertices ($|V|!$ permutations). In the second experiment the time to add a graph to the deci-

Table 1: Results of graph experiments (first 10 graphs).

Graph	**Vertices**	**Permutations**	**Permutations**	**Same lables**
	#	**(modified)**	**(original)**	**(max.)**
1	17	3.26×10^{6}	3.55×10^{14}	5
2	21	3.59×10^{9}	5.10×10^{19}	8
3	17	1.08×10^{7}	3.55×10^{14}	5
4	20	2.50×10^{8}	2.43×10^{18}	6
5	24	1.64×10^{12}	6.20×10^{23}	10
6	17	1.63×10^{6}	3.55×10^{14}	3
7	21	2.04×10^{7}	5.10×10^{19}	3
8	30	1.39×10^{12}	2.65×10^{32}	5
9	22	8.01×10^{8}	1.12×10^{21}	6
⋮	⋮	⋮	⋮	⋮
100	23	1.00×10^{9}	2.58×10^{22}	6
⌀	23.05	1.09×10^{13}	3.73×10^{31}	5.23

Table 2: Run time for compiling the decision tree for each graph

Graph	**Vertices**	**Run time**	**Permutations**	**Same lables**
	#	**(minutes)**	**#**	**(max.)**
1	17	1.47	8.19×10^{5}	4
2	17	8.90	4.17×10^{6}	5
3	21	45.67	5.32×10^{7}	4
4	21	10.06	8.19×10^{6}	3
5	21	38.01	4.09×10^{7}	3

sion tree was measured and again the number of permutations of the adjacency matrix which were added to the decision tree. As the experiment was quite time-consuming on a desktop machine, only the performance for five smaller graphs was measured. The results of the experiment are listed in Tab. 2. The experiments show that the algorithm significantly reduces the number of permutations (see Tab. 1). Though, the time needed to compile the decision tree is still quite long even for small problem instance, as shown in Tab. 2. However, as the method is designed for an off-line preprocessing and considered to run on a server machine, it is still reasonable for practical applications.

5 Conclusions and Future Work

In this paper an extension for the method of Messmer's subgraph matching has been proposed. The original method is very efficient to perform exact subgraph matching on a large database. However, it has a limitation for the maximum number of vertices. The modification discussed in this paper enables to increase this limit depending on how the vertices are labeled. As the number of permutations in the preprocessing step depends on the vertices with the same labels, an analysis of the data that will be represented in graph is necessary. If there are

just a few vertices with the same label, e.g. less than five, even graphs with 30 vertices can be handled. It has been proven that the modification of the method does not affect its completeness.

Noteworthy, the proposed method can be applied in several areas, such as object recognition, matching of 2D or 3D chemical structures, and architectural floor plan retrieval. Future work will be to perform experiments on real graph data sets and research strategies for choosing appropriate weight functions. Furthermore, we plan to extend this method to provide a fast method for error-tolerant graph matching.

References

1. Bunke, H.: On a relation between graph edit distance and maximum common subgraph. Pattern Recognition Letters 18(8), 689 – 694 (1997)
2. Bunke, H., Messmer, B.: Efficient attributed graph matching and its application to image analysis. pp. 44–55 (1995)
3. Bunke, H.: Graph matching: Theoretical foundations, algorithms, and applications. Proc. Vision Interface (2000)
4. Bunke, H.: Recent developments in graph matching. Pattern Recognition, International Conference on 2, 2117 (2000)
5. Bunke, H., Shearer, K.: A graph distance metric based on the maximal common subgraph. Pattern recognition letters pp. 255–259 (1998)
6. Cheng, J., Huang, T.: Image registration by matching relational structures. Pattern Recognition 17(1), 149 – 159 (1984)
7. Gao, X., Xiao, B., Tao, D., Li, X.: A survey of graph edit distance. Pattern Analysis and Applications 13(1), 113–129 (Januar 2009)
8. Kim, W., Kak, A.: 3-d object recognition using bipartite matching embedded in discrete relaxation. IEEE Transactions on Pattern Analysis and Machine Intelligence 13, 224–251 (1991)
9. Messmer, B., Bunke, H.: A decision tree approach to graph and subgraph isomorphism detection. Pattern Recognition 32, 1979–1998 (1999)
10. Rosen, K.H.: Discrete mathematics and its applications (2nd ed.). McGraw-Hill, Inc., New York, NY, USA (1991)
11. Schomburg, I., Chang, A., Ebeling, C., Gremse, M., Heldt, C., Huhn, G., Schomburg, D.: BRENDA, the enzyme database: updates and major new developments. NUCLEIC ACIDS RESEARCH 32(Sp. Iss. SI) (JAN 2004)
12. Ullmann, J.: An algorithm for subgraph isomorphism. Journal of the ACM (JACM) 23(I), 31–42 (1976)
13. Weber, M., Langenhan, C., Roth-Berghofer, T., Liwicki, M., Dengel, A., Petzold, F.: a.SCatch: Semantic Structure for Architectural Floor Plan Retrieval. In: Advances in Case-Based Reasoning, Proc. of ICCBR 2010 (2010)
14. Wong, E.K.: Model matching in robot vision by subgraph isomorphism. Pattern Recogn. 25, 287–303 (March 1992)