Fast Subgraph Isomorphism Detection for Graph-Based Retrieval

Markus Weber[1,2], Christoph Langenhan[3], Thomas Roth-Berghofer[4,1]
Marcus Liwicki[1], Andreas Dengel[1,2], and Frank Petzold[3]

[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, Germany

[3] Chair of Architectural Informatics, Faculty of Architecture,
Technical University of Munich, Arcisstrasse 21, 80333 Munich, Germany

[4] Explanation-aware Computing Systems, Institute of Computer Science,
University of Hildesheim, Marienburger Platz 22, 31141 Hildesheim, Germany

`{firstname.lastname}@dfki.de,{lastname}@tum.de`

Abstract. In this paper we present a method for a graph-based retrieval and its application in architectural floor plan retrieval. The proposed method is an extension of a well-known method for subgraph matching. 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.

Key words: architecture, graph theory, retrieval

1 Introduction

A graph is a mathematical representation that consists of vertexes and edges and can be applied to representations that capture relationships between any two elements. Thus they offer a number of advantages over traditional feature vector approaches [1]. Unlabeled graphs have no fixed labels, thus the only applicable similarity methods are ones that search for identical subgraphs. This subgraph isomorphism problem is a well-known problem in literature and is known to be NP-complete [2]. Similarity assessment for labeled graphs in general is domain-specific. Although being polynomial, graph representations do have a significant computational cost. Fortunately, there are a number of methods and techniques aimed at reducing this problem [1].

However, subgraph isomorphism still is a powerful general similarity measure which also could be applied without any specific domain knowledge. In order to reduce computational cost in subgraph isomorphism, index based approaches have been introduced. Such a method has been proposed by Messmer et al. [3]. It builds an index using the permutated adjacency matrix of the graph. The real-time search is then based on a tree. While the method has shown to be effective for a reference set with small graphs, it is unfeasible 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 method has been applied in the architectural floor plan retrieval. In [4] a graph-based structure for representing the spatio-relational content of a floor plan has been introduced. The case-based design approach for floor plan retrieval described in [5] deals with semantic and topological information of spatial configurations not regarding procedural information. The topological and functional inner structure, the orientation of spaces as well as the urban integration, and the relation of buildings to each other is vital for a retrieval of valuable reference to support architects during the early stages of designing a building. Thus a digital fingerprint was proposed in [6] that contains a clear spatial description of an architectural dataset. The proposed system offers a computational approach to extract a few characteristic and prominent features of a floor plan which are then used to generate a digital fingerprint. In the paper, we examine the development of graph-based methods to provide a sketch-based submission and retrieval system for publishing and researching building layouts.

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 and Section 3.2 the modified retrieval algorithm. Next, Section 4 will show that the number of computational steps will be significantly decreased and Section 5 discusses the application in the architectural domain. Finally, Section 6 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 [8] 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 [9, 1] 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. Another way is by applying graph edit costs, 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 as Hopfield networks, Kohonen maps or Potts MFT neural nets. Finally, genetic algorithms, the usage of Eigenvalues, and linear programming are applied.

Graph matching is challenging in presence of large databases [10, 1]. 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 [11], any clustering algorithm can be applied. Graph indexing can be performed by the use of decision trees.

Messmer and Bunke [3] 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.

As topologies are crucial to describe the relation of spaces, graphs are widely used to store information about buildings. The EsQUIsE project focuses on the the early design stages of buildings to support architects with a pen-based interface to sketch ideas and simulate certain aspects of the design. Juchmes et al. [12] proposes a floor plan like input strategy to use adjacency, dimension and orientation of space to build a 3D model and a graph structure. The PROBADO-framework intends to integrate multimedia content to digital libraries. Apart of indexing methods for music and 3D shape retrieval [13] a room connectivity graph [14] was proposed. To build up the graph structure of the spatial relation of a non semantic 3D-Model the storeys are detected, the room per storey, doors and windows determined. Using a sketch-based retrieval interface 3D-models will be retrieved by inputting schematic topologies of a spatial configuration.

3 Definitions and Notations

Basic definitions that are used throughout the paper are a labeled graph $G = (V, E, L_v, L_e, \mu, \upsilon)$ with its common representation as an adjacency matrix M:

Definition 1 *An* ***adjacency matrix*** *is $n \times n$ matrix M.*

$$M = (m_{ij}), i, j = 1, ..., n, \textit{ where}$$

$$m_{ii} = \mu(v_i)$$

and

$$m_{ij} = v((v_i, v_j)) \textit{ for } i \neq j.$$

Thus a graph G can also be represented by its adjacency matrix M. But the matrix M is not unique for a graph G. If M represents G, than any permutation of M is also a valid representation of G.

Definition 2 *A $n \times n$ matrix $P = (p_{ij})$ is called a* ***permutation matrix*** *if*

1. $p_{ij} \in 0, 1$ *for i,j = 1, ..., n*
2. $\sum_{i=1}^{n} p_{ij}$ *= 1 for j = 1, ..., n*
3. $\sum_{i=1}^{n} p_{ij}$ *= 1 for i = 1, ..., n*

Furthermore, a so called row-column representation is given. Each matrix can be represented by its row-column elements a_i, where a_i is a vector of the form

$$a_i = (m_{1i}, m_{2i}, ..., m_{ii}, m_{i(i-1)}, ..., m_{ii}).$$

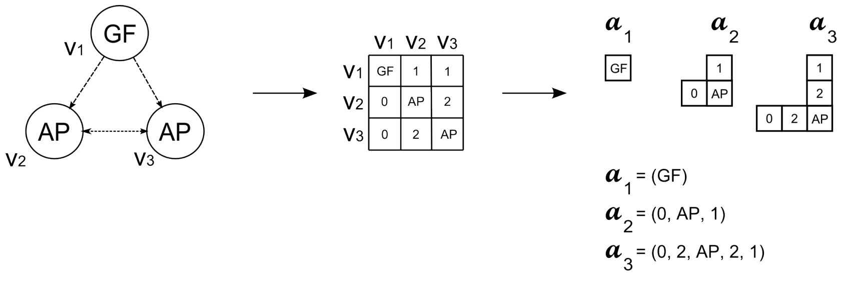

Fig. 1: The row-column representation of an adjacency matrix

In order to compare two adjacency matrices with different dimensions, a notation $S_{k,m}(M)$ is required which reduces the dimension of the matrix with the higher dimension.

Definition 3 *Let $M = (m_{ij})$ be a $n \times n$ matrix. Then $S_{k,m}$ denotes the $k \times m$ matrix that is obtained from M by deleting rows k + 1,..., n and columns j = 1,...,m where $k, m \leq n$. That is, $S_{k,m}(M) = (m_{ij})$; i = 1,...,k and j = 1,...,m.*

Besides, definitions for orders on sets are needed.

Definition 4 *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 5 *A partial or total order $\leq$ over a set X is **well-founded**, iff $(\forall\ Y \subseteq X\ :\ Y \neq \emptyset \rightarrow (\exists y \in Y\ :\ y \text{ minimal in } Y \text{ in respect of } \leq))$.*

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

Definition 6 *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 7 *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 [3], 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_1 and G_2 be graphs with their adjacency matrices M_1 and M_2 of dimension $m \times m$ and $n \times n$ and $m \leq n$. The problem of finding a subgraph isomorphism from G_1 to G_2 is equivalent to finding a permutation matrix, so the subgraph isomorphism is given, iff there is a $n \times n$ permutation matrix P such that

$$M_1 = S_{m,m}(PM_2P).$$

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 6) 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 4 and Definition 5) on the set of labels which reduces the number of allowed inversion for the adjacency matrix. Figure 2 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 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. $\sigma(\mu(v_1)) \leq \sigma(\mu(v_2)) \leq \sigma(\mu(v_3))$

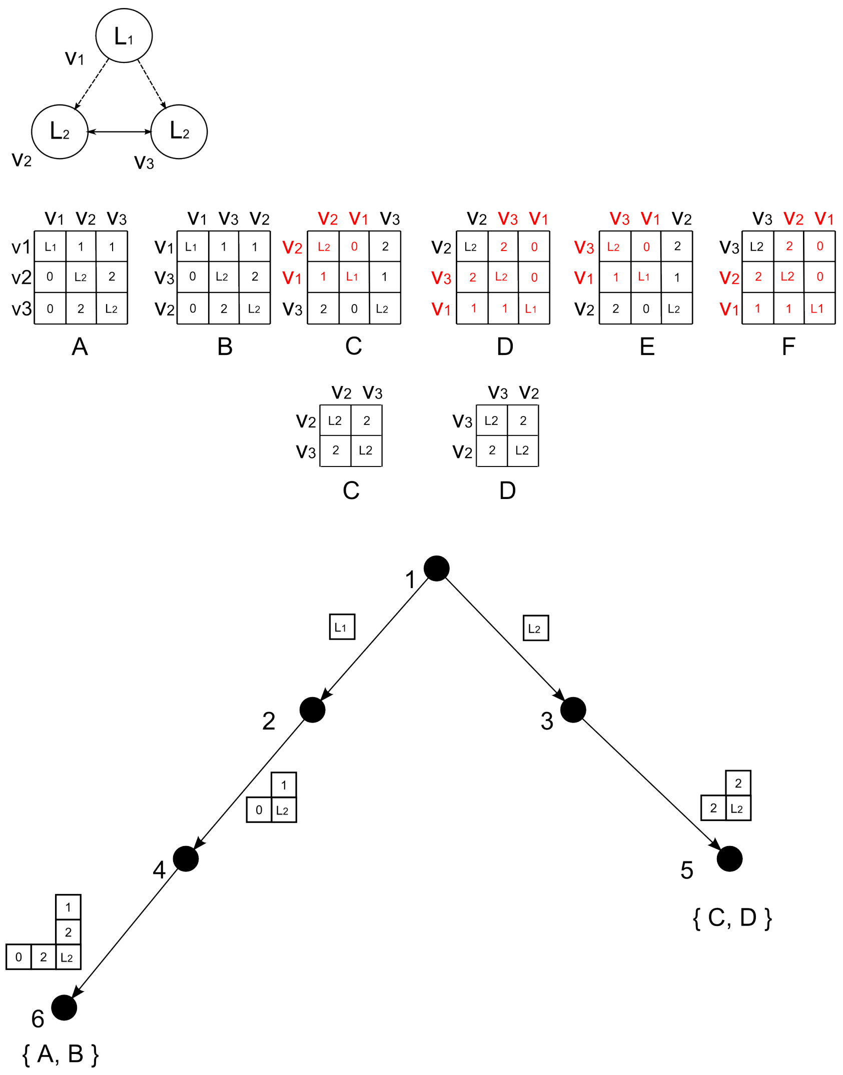

Fig. 2: Modified decision tree for adjacency matrices

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.

3.2 Retrieval Algorithm

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 Q the adjacency matrix M 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. The pseudo code of Algorithm 1 displays how the results are retrieved by traversing the tree:

Algorithm 1 RETRIEVAL(Q = (V, E, μ, υ, σ, L_v, L_e), Tree)

```
Require: Unsorted set $V$ of vertices, $\mu$ labeling function, $\sigma$ weight function
 1: sort(Q, $L_v$, $\mu$, $\sigma$)
Ensure: Vertices V are sorted according to the defined order.
 2: Let $R$ be an empty sorted set which will contain all results sorted by the similarity
    value $Sim$.
 3: Determine adjacency matrix $M$ from graph $Q$.
 4: Determine row-column list $RCL$ from $M$.
 5: for $i \leftarrow 1$ to $|V| - 1$ do
 6:     for $j \leftarrow 1$ to $|RCL|$ do
 7:         Find best match for row-column vector $a_i \in RCL$ in tree at $level_i$.
 8:         Update $R$ and $Sim$.
 9:     end for
10:     Remove element $V_i$ from $V$.
11: end for
12: return R.
```

3.3 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 $A(G)$, $M \in A(G)$.

Algorithm 2 BUILD_INDEX($G = (V, E, L_v, L_e, \mu, \upsilon, \sigma)$, Tree)

```
Require: Unsorted set V of vertices, $\mu$ labeling function, $\sigma$ weight function.
 1: sort(V, $L_v$, $\mu$, $\sigma$)
Ensure: Vertices V are sorted according to the defined order.
 2: Let $O$ be an empty list.
 3: for all $l_i \in L_V$ do
 4:     Let $\{v_a, \dots, v_b\}$ contain all $v$ with $\mu(v) = l_i$
 5:     $O_i \leftarrow VARIATIONS(\{v_a, \dots, v_b\})$
 6: end for
 7: Let $AG \leftarrow O_1 \times \dots \times O_{|Lv|}$.
 8: for all $m_i$ in $AG$ do
 9:     Add row column vector for sequence of $m_i$ to Tree.
10: end for
```

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

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

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 2 shows how the index is built. Algorithm 3 and Algorithm 4 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 $\{v_a, \dots, v_b\}_i$ all variations with respect to the order have to be determined. These variations are computed in Algorithm 5, by determining all combinations of the interval $\{v_a, \dots, v_b\}_i$ including the empty set and calculating all permutations for these combinations. Algorithm 3 and Algorithm 4 realize the algorithm proposed by Rosen [15] 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, where k is $L_{interval} =$ length of interval; $k = 1 \dots L_{interval}$, 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

Algorithm 4 $ROTATE(V, begin, end, R)$

```
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 5 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.
```

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 5 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.

3.4 Complexity Analysis

The computational complexity analysis discussed in this section will be based on the following quantities:

$$\begin{aligned}
N &= \text{the number of graphs in the floor plan graphs in the repository,}\\
M &= \text{the maximum number of vertices of a graph in the floor plan repository,}\\
M_v &= \text{the number of vertex labels for a graph in the floor plan repository,}\\
I &= \text{the number of vertices in the query graph,}\\
I_v &= \text{the number of vertex labels.}
\end{aligned}$$

The original algorithm by Messmer [3] 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 compilation of the decision tree for a graph $G = (V, E, L_v, L_e, \mu, \upsilon, \sigma)$ has a run time complexity of

$$\mathcal{O}(|V|!).$$

For the size of the decision tree Messmer determined the following bounds. The sum of nodes over all the levels (without the root node) is limited to

$$\mathcal{O}(l_v \sum_{k=0}^{M-1} \binom{M}{k} (l_2^e)^k) = \mathcal{O}(l_v(1 + l_e^2)^M),$$

and as the decision tree becomes linearly dependent on the size of the database N, the space complexity of the decision tree is

$$\mathcal{O}(N l_v (1 + l_e^2)^M).$$

The processing time for the new decision tree compilation algorithm is reduced. Algorithm 2 describes the new way to The basic idea of the algorithm is to take all labels L_v with the same weight which occur in the graph and omit their instances. So, considering the mathematical idea of the algorithm an approximation of the run time complexity would be:

$$\prod_{i=1}^{|L_v|} \left(\underbrace{|\{\forall v \in V | \mu(v) = l_i\}|!}_{n_i} + \sum_{j=1}^{n_i - 1} \binom{n_i}{j} \cdot j! \right).$$

The first term considers the permutations for all labels with the same weight, denoted by n_i. The second term describes the k-variations. As we have to consider the variations from $n_i - 1$ to 1, the sum is sufficient.

In order to simplify the equation, we can combine the two terms, as the $n_i!$ is equivalent to $\binom{n_i}{n_i} \cdot n_i! = 1 \cdot n_i! = n!$, thus we have:

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

Now, let n_{max} be the maximum number of vertices with the same weight. Then we have the upper bound of

$$\left(\sum_{j=1}^{n_{max}} \binom{n_{max}}{j} \cdot j! \right)^{|L_v|}.$$

A rather imprecise approximation of the sum $\sum_{j=1}^{n_{max}} \binom{n_{max}}{j} \cdot j!$ would be $(n_{max} + 1)!$, so the resulting complexity of the algorithm 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.

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

Graph	**Vertices #**	**Permutations (modified)**	**Permutations (original)**	**Same lables (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
$\varnothing$	23.05	1.09×10^{13}	3.73×10^{31}	5.23

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 3.4:

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

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 decision 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.

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

5 Application in Architecture

In [4], we already a graph representation for floor plans has been introduced. This representation is used to describe the spatio-relational content of the floor plan. The nodes of the floor plan graph represent functional rooms in a floor plan. Each label is an entity type which has been introduces in [4] and by using the edges the relation between these functional units is described.

So, if the architect is searching for a specific composition of functional rooms, a query is generated which describes this composition. Figure 3 illustrates a query graph Q, where the architect would search for two directly connected apartments (AP) in an attic floor (AT). Each leaf of the decision tree is a possible alignment of the adjacency matrix representing a graph. Thus all leafs beneath the last matching node of the search path, respectively the associated graphs, are added to the result set R. The number of exact row column vector matches N for the query graph Q divided through the number of nodes of the query graph $|V|_q$is used as a simple similarity measure.

$$Sim(q, R) = \frac{N}{|V|_q}$$

In our example the similarity would be $66,\overline{6}\%$ as only two of three comparisons match.

6 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.

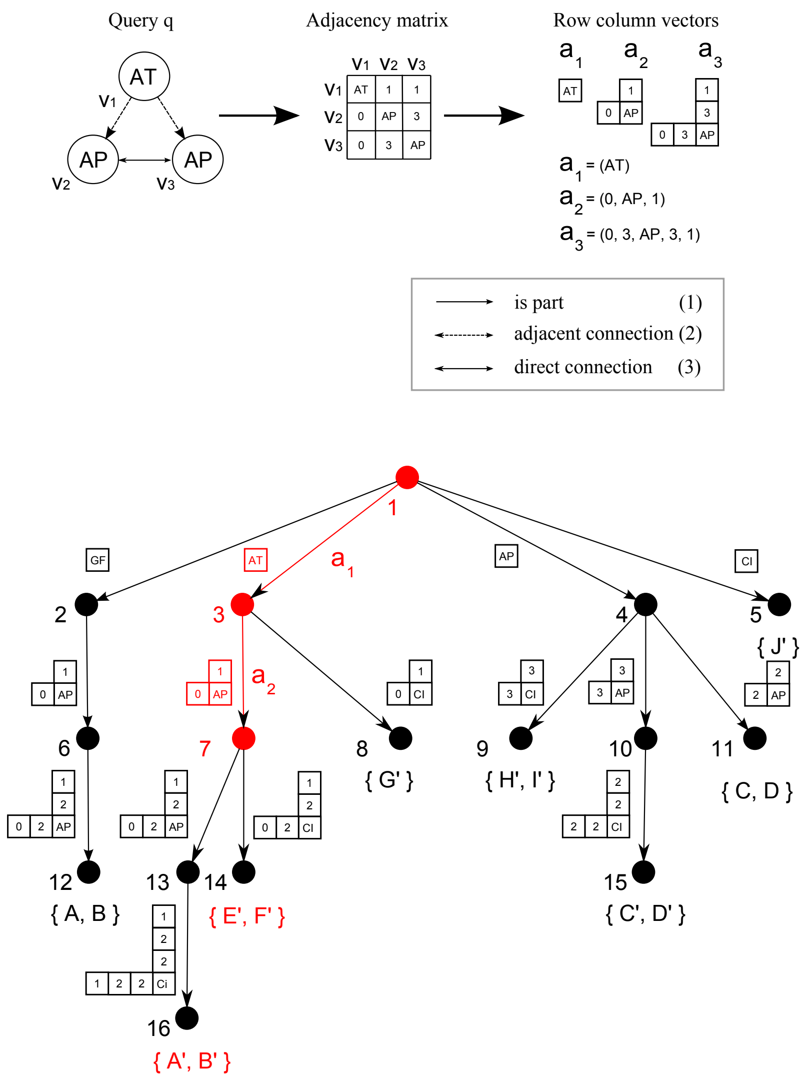

Fig. 3: Retrieval process.

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. In the paper we discussed how the method is applied on architectural floor plan retrieval and future work will be to perform experiments on real graph data sets of different domains 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.: Recent developments in graph matching. Pattern Recognition, International Conference on **2** (2000) 2117
2. Cook, S.A.: The complexity of theorem-proving procedures. In: STOC '71: Proceedings of the third annual ACM symposium on Theory of computing, New York, NY, USA, ACM (1971) 151–158
3. Messmer, B., Bunke, H.: A decision tree approach to graph and subgraph isomorphism detection. Pattern Recognition **32** (1999) 1979–1998
4. 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)
5. Langenhan, C., Weber, M., Liwicki, M., Petzold, F., Dengel, A.: Sketch-based Methods for Researching Building Layouts through the Semantic Fingerprint of Architecture. In: Computer-Aided Architectural Design Futures 2011: Proceedings of the International CAAD Futures Conference. (2011)
6. Langenhan, C., Petzold, F.: The Fingerprint of Architecture - Sketch-Based Design Methods for Researching Building Layouts Through the Semantic Fingerprinting of Floor Plans. In: International electronic scientific-educational journal: Architecture and Modern Information Technologies. (2010)
7. Gao, X., Xiao, B., Tao, D., Li, X.: A survey of graph edit distance. Pattern Analysis and Applications **13**(1) (Januar 2009) 113–129
8. Ullmann, J.: An algorithm for subgraph isomorphism. Journal of the ACM (JACM) **23**(I) (1976) 31–42
9. Bunke, H.: Graph matching: Theoretical foundations, algorithms, and applications. Proc. Vision Interface (2000)
10. Bunke, H.: On a relation between graph edit distance and maximum common subgraph. Pattern Recognition Letters **18**(8) (1997) 689 – 694
11. Bunke, H., Shearer, K.: A graph distance metric based on the maximal common subgraph. Pattern recognition letters (1998) 255–259
12. Juchmes, R., Leclercq, P., Azar, S.: A multi-agent system for the interpretation of architectural sketches. In: EG Workshop on Sketch-Based Interfaces and Modeling. (2004) 53–61
13. Blümel, I., Berndt, R., Ochmann, S., Vock, R., Wessel, R.: Supporting planning through content-based indexing and 3d shape retrieval. In: 10th Int. Conf. on Design & Decision Support Systems in Architecture and Urban Planning. (2010)
14. Wessel, R., Blümel, I., Klein, R.: The room connectivity graph: Shape retrieval in the architectural domain. In Skala, V., ed.: The 16-th Int. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision'2008, UNION Agency-Science Press (2008)
15. Rosen, K.H.: Discrete mathematics and its applications (2nd ed.). McGraw-Hill, Inc., New York, NY, USA (1991)