Structural Similarity and Adaptation

Katy Börner[1], Eberhard Pippig[1], Elisabeth-Ch. Tammer[1], Carl-H. Coulon[2]

[1] HTWK Leipzig, FB IMN, PF 30066, 04251 Leipzig, GERMANY
katy | eberhard | tammer@informatik.th-leipzig.de
[2] FIT, AI Research Division, 53754 Sankt Augustin, GERMANY
coulon@gmd.de

Abstract. Most commonly, case-based reasoning is applied in domains where attribute value representations of cases are sufficient to represent the features relevant to support classification, diagnosis or design tasks. Distance functions like the *Hamming-distance* or their transformation into similarity functions are applied to retrieve past cases to be used to generate the solution of an actual problem. Often, domain knowledge is available to adapt past solutions to new problems or to evaluate solutions. However, there are domains like architectural design or law in which *structural case representations* and corresponding *structural similarity functions* are needed. Often, the acquisition of adaptation knowledge seems to be impossible or rather requires an effort that is not manageable for fielded applications. Despite of this, humans use cases as the main source to generate adapted solutions. How to achieve this computationally? This paper presents a general approach to structural similarity assessment and adaptation. The approach allows to explore structural case representations and limited domain knowledge to support design tasks. It is exemplarily instantiated in three modules of the design assistant FABEL-Idea that generates adapted design solutions on the basis of prior CAD layouts.

1 Introduction

To provide support in a complex real world domain like design, case-based reasoning (CBR) has been suggested as an appropriate problem solving method [14, 17, 13, 15]. In CBR, a new problem is solved analogously to past experiences (cases). That is, cases similar to the problem are retrieved, the set of best cases is selected, a solution is derived and evaluated and the new problem along with its solution is stored in memory [17, 25]. In most CBR applications past experiences have no inherent structure and are described by fixed sets of attribute value pairs. Traditionally, case adaptation is guided by static libraries of hand-coded adaptation rules. If model-based knowledge about the domain is available it may be used to constrain the reasoning process (retrieval as well as adaptation) or to evaluate solutions.

In design, cases correspond to arrangements of physical objects represented by CAD layouts and refer to parts in real buildings. The topological structure (which does not reflect the function or behaviour of objects) inherent in such layouts needs to be considered during reasoning. Complex case representations, however, increase the computational expense in retrieving, matching, and adapting cases. Efficient memory organization directly tailored to analogical reasoning becomes essential. Additionally, design is a weak theory domain. Hardly any information about the relevance of features guiding the selection of similar cases is available. The adaptation of prior layouts mainly corresponds to adding, eliminating, or substituting physical objects. Because of the variety and the possible number of combinations of these modifications, adaptation knowledge is difficult to acquire by hand. In the project FABEL approaches have been developed that define the *structural similarity* [6, 16] between structured case representations (graphs) by their maximal common subgraphs (*mcs*). Given a new problem the structurally most similar case(s) are retrieved. One out of several *mcs* is transferred and the remaining case parts are taken over as needed. Additionally, the *mcs* may be used to represent and access classes of cases in an efficient manner. The remaining case parts can be seen as proper instantiations of *mcs*, i.e., as a special kind of adaptation knowledge that allows to adapt past cases to solve new problems. In such a way adaptation knowledge can be automatically extracted out of past cases reducing the effort needed for knowledge acquisition enormously.

This paper outlines the application domain and task and introduces the basic functionality required to support various design tasks. Based on this three concrete approaches are presented that apply structural similarity assessment and adaptation to provide this functionality in an efficient way. The approaches differ by focusing on specific parts of the CBR-scenario. Finally, related work will be discussed and conclusions will be drawn.

2 Application Domain and Task

The application domain used to motivate, illustrate, and evaluate the approaches to structural similarity and adaptation is architectural design. In particular we are concerned to support the design of rectangular building layout. Here, past experiences (cases) correspond to arrangements of physical objects represented by CAD layouts and refer to parts in real buildings. Each object is represented by a set of attributes describing its geometry and its type (e.g. fresh air connection pipe). Concentrating on the design of complex installation infrastructures for industrial buildings, cases correspond to pipe systems that connect a given set of outlets to the main access. Pipe systems for fresh and return air, electrical circuits, computer networks, phone cables, etc. are numerous and show varied topological structures. As for the retrieval, transfer, and adaptation of past cases to new problems not the geometry and type of single objects but their topological relations are important.

Due to this, objects and their (topological) relations need to be represented and considered during reasoning. Therefore, the approaches described in this article use a *compile* and *recompile* function to translate attribute value representation of objects and their relations into graphs. In general, objects are represented by vertices and relations between objects are represented by edges.

Reasoning, i.e. structural retrieval and adaptation proceeds via graph-based representations. A *recompile* function translates the graph-based solution into its attribute-based representation that may be depicted graphically to the architect. Concentrating on different aspects of structural similarity assessment and adaptation different compile functions are appropriate, resulting in different graph representations of cases and corresponding expressibility power and reasoning complexity. They are explained in detail in section 4.1 to 4.3. As an example see Fig. 1 which shows a problem and its solution. Some of the spatial relations (*touches, overlaps, is_close_to*) used by TOPO to represent cases structurally are visualized by arrows.

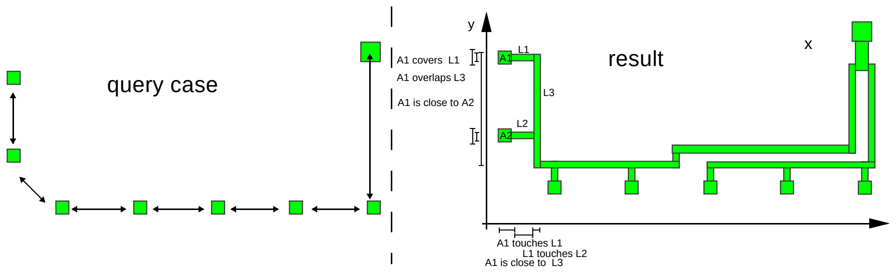

Fig. 1. Domain example: A problem and its solution

Given a base of cases represented by graphs, reasoning proceeds as follows: First, one or a set of cases that show a high structural similarity to the problem needs to be *retrieved* out of the case base. Here structural similarity is defined via the maximal common subgraph (*mcs*) of a case and a problem. Assuming that cases and problem may share more in common than their maximal common subgraph, structural *adaptation* proceeds by transferring and combining case parts that connect unconnected problem objects to the *mcs*.

3 Required Functionality

The required functionality (informally defined in the last section) will be defined via the mappings needed to transfer a set of cases and a problem into one or a set of problem solutions.

We give some basic notions and notations first. A graph $g = (V^g, E^g)$ is an ordered pair of vertices V^g and edges E^g with $E^g \subseteq V^g \times V^g$. Let $mcs(G)$ denote the set of all maximal common subgraphs of a set of graphs G, with respect to some criteria. If there is no danger of misunderstanding, the argument of mcs will be left out. Let Γ be the set of all graphs, and O be a finite set of objects represented by attribute values for geometry and type. $\mathcal{P}(\Gamma)$ denotes the powerset of Γ, that means the set of all subsets of Γ.

The mappings needed to accomplish the required functionality are depicted in Fig. 2 and are explained subsequently. In order to access and interact via CAD layouts (that are represent by attribute value pairs) but to reason via topological structure there must be a way of translating attribute value representations of cases into graph representations and reverse. Therefore, a *compile* function has to be defined that maps the attribute value representation of a set of objects representing a case or a problem into its structural representation:

$$compile : \mathcal{P}(O) \rightarrow \Gamma.$$

Inversely, the function *recompile* maps the graph representation of a set of objects denoting a solution into their attribute value representation:

$$recompile : \Gamma \rightarrow \mathcal{P}(O).$$

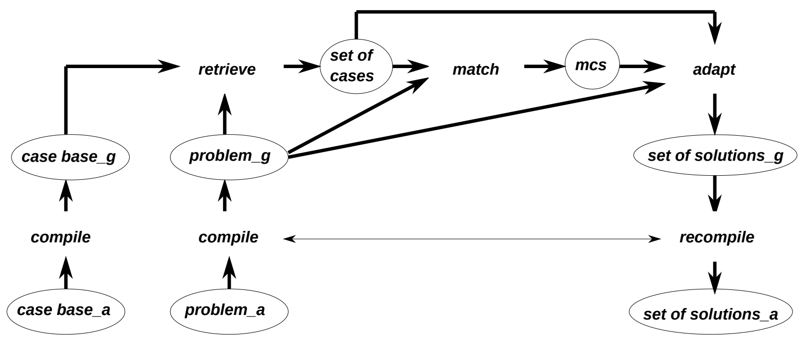

Fig. 2. Mappings required to accomplish the required functionality

The concept of structural similarity allows the selection of one or more cases, that are suitable for solving a problem. It is used by the function *retrieve*, that maps a set of cases (i.e. the case base) and a problem into a set of candidate cases that are applicable to solve the problem:

$$retrieve : \mathcal{P}(\Gamma) \times \Gamma \rightarrow \mathcal{P}(\Gamma).$$

Retrieve itself uses (sometimes repeatedly) a function named *match* that maps two graphs into their maximal common subgraphs *mcs*:

$$match : \Gamma \times \Gamma \rightarrow \mathcal{P}(\Gamma).$$

As for adaptation, vertices and edges of the selected set of candidate cases are transferred and combined to complement the problem into a set of solutions:

$$adapt : \mathcal{P}(\Gamma) \times \Gamma \times \Gamma \rightarrow \mathcal{P}(\Gamma).$$

The function *adapt* also itself will be influenced by the function *match* such that a single element of *mcs* is used for adaptation.
Because of the finity of the object set and the existence of the *compile* function, we can restrict the last four mappings to finite domains, i.e., Γ may be replaced by $compile(\mathcal{P}(O))$.

4 Three Approaches to Structural Similarity and Adaptation

This section introduces three approaches that provide the defined functionality using structural similarity assessment and adaptation. The approaches differ in the compile and recompile functions applied. Different graph representations (trees to arbitrary graphs) are used to represent cases illustrating the tradeoff between expressibility and match complexity. Different ways of memory organization are used and several retrieval and adaptation strategies are proposed. The approaches are compared at the end of this section.

4.1 TOPO

The main feature of TOPO is the case-based extension and correction of rectangular layouts.

Compile and Recompile: The *compile* function used by TOPO detects binary topological relations of various types. The type of a relation is determined by the application dependent attributes of involved objects and their 3-dimensional topological relation. TOPO's *compile* function projects each layout to the three axes and the 3-dimensional relation is a combination of the relations detected for each dimension. For each projection 8 different directed relations can be detected. They are similar to the temporal relations of [1], but additionally distinguish several classes of distances between disjoint intervals. Therefore, a given object may be in one of 16 relationships for each dimension leading to $16^3 = 4096$ different 3-dimensional relationships [12].

Building a graph out of objects and relations, one must decide which ones should be the vertices and which ones the edges. Depending on this decision different subgraphs of graph representations of two layouts become isomorph. As described and discussed in [12] TOPO uses the relations as vertices and the objects as edges because it allows to detect weaker, but larger, correspondencies between two layouts.

Retrieval: The *matching* function searches for the maximal common subgraphs of two graphs. In order to solve a similar task, the problem of finding a maximal clique of a graph, various NP-complete algorithms has been developed [2]. A clique is a complete subgraph of a graph (every vertex is connected with every other one). Instead of searching for a common subgraph of two graphs TOPO searches for a maximal clique in one graph representing all possible matchings between the two graphs, called their *combination graph.*

Building the combination graph: Using the transformation in [3], the vertices in the combination graph represent all matchings of compatible vertices in the source graphs. Figure 3 shows an example. The source graphs f and g contain objects of type a and b connected by directed relations. The type of a relation is defined by the types of its source and target objects. Two vertices are connected in the combination graph if and only if the matchings represented by the vertices do not contradict one another. The matchings (R_2(a,b)⇔R_8(a,b)) and (R_5(b,b)⇔R_{10}(b,b)) are connected because both relations occur in both source graphs in the same context. Both are connected by a shared object of type b. (R_2(a,b)⇔R_8(a,b)) and (R_1(b,a)⇔R_6(b,a)) are not connected because the matched relations share an object of type a in graph f but do not share any object in graph g.

The maximal clique in this combination graph and the corresponding maximal subgraphs are marked in black.

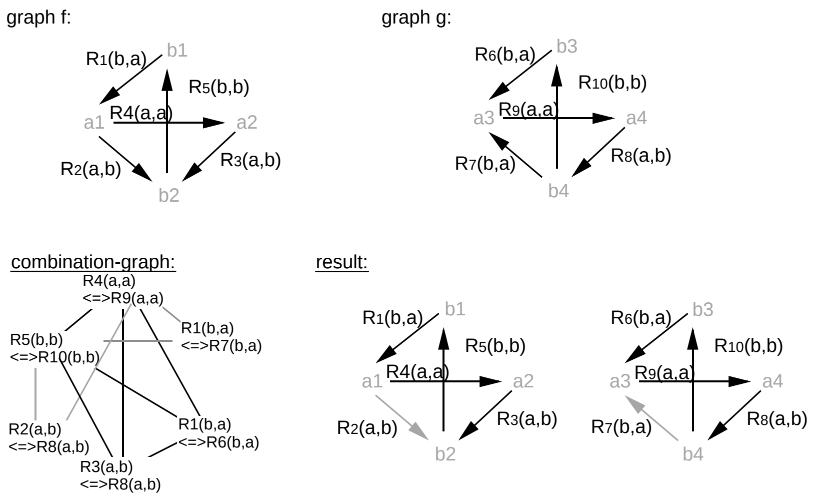

Fig. 3. Transformation of the problem of finding the maximal common subgraph to the problem of finding a maximal clique in a graph. The maximal clique and the corresponding matching are marked in black.

A general maximal clique algorithm: The algorithm of [11] (for further use called *max-clique*$_{BK}$) finds all cliques in a graph by enumerating and extending all complete subgraphs. It extends complete subgraphs of size k to complete subgraphs of size $k + 1$ by adding iteratively vertices which are connected to all vertices of the complete subgraph.

As an improvement [12] describes, how to reduce the search space by searching for matchings of connected subgraphs only and combining them in a second step.

The function retrieval: TOPO includes no own retrieval function but supplies the retrieval module ASPECT [20] with two different similarity functions. Given a similarity function and a query case, ASPECT guarantees to find the most similar source case without the need to compare all cases.

Given two cases, the first similarity function returns the size of the maximal common subgraphs relative to the minimum size of both graphs. Because a maximal common subgraph cannot be larger than the smaller one of both graphs the result is always a rational number between 0 and 1. In order to avoid the np-complete search for the maximum common subgraphs a second similarity function is defined. It compares the sets of relations ocuring in both cases. The result is the number of compatible relations relative to the minimum number of detected relations of both cases leading to a result, which is also between 0 and 1. The second similarity function obviously returns an upper bound of the first similarity function, but has no np-complexity.

Adaptation: TOPO extends, refines and corrects layouts by case adaptation. Because there exists no theory dividing a layout statically into problem part and solution part, TOPO uses the heuristic that every object of the source case which is not found in the query case might be part of the solution. After determining the common subgraph of the graph representation of a query and a case it offers to transfer all objects from the source case to the query case which are connected to the common part by a path of topological relations. In order to use this path of topological relations to determine the position of the transferred object in the query, TOPO uses the recompile functions. The user may specify the parts to be transferred by selecting types of desired objects.

During transfer TOPO may change the size of transferred objects to preserve topological relations. For example a window is resized in order to touch both sides of a room. To avoid geometries which are impossible for an object, TOPO limits the resizing to geometries which occurred in the case base.

The adaptation described so far is additive only and this is the main aim of TOPO. A second way of using TOPO is transformative. TOPO searches in the query case for objects constituting an unusual topological relationship. For this reason TOPO creates a statistic about the frequency of topological relations occurring in the case base for each type of objects. For example, 100 percent of the outlets in the case base touch pipes and 2 percent of the chairs touch a shelf. In case of detecting unusual relationships between objects of a query, TOPO asks the user whether the position of one or both of the objects should be changed or not. If the position of an object is confirmed to be changed, TOPO searches for an object of the source case compatible to the object of the query which is related to compatible objects, but by more frequent topological relations. Using the recompile functions of those more frequent relations the new position of the object of the query case is determined. As an example let us suppose, that TOPO detected the unusual relation *used air outlet inside used air connection pipe* in a query case like in Fig. 5 and the user confirmed the position of the outlet to be changed. TOPO searches for a part of the source case, where there is an object which is compatible to the outlet and related to another object which ist compatible to the connection pipe. Probably it would find an outlet which touches a connection line. Therefore, it would transfer the relation *touching* and accordingly reconstruct the position of the outlet of the query case.

4.2 MACS

In order to apply our structural concept the module MACS uses an appropriate representation of domain objects which focuses on their structure. Therefore, domain objects translated into unconstrained graphs. The preference of MACS involves to structure the case base dynamically based on the structural similarity of graph-based cases and to realize a fast retrieval over such a structured case base.

Compile and Recompile: Analogous to the other approaches the function *compile* guarantees the transformation of an attribute value represented case into its graph representation (see [4] for examples). We represent cases very close to the graphical representations which are in accordance with former reflections about the graph structure of the domain elements. There are mainly two ways to interpret a layout fragment:

- All architectural objects which appear in the layout are represented as vertices. It is possible but not necessary to label the vertices with a type name of the object and with additional qualitative and quantitative attributes. The edges are used for expressing the topological relationships between neighboring vertices and, possibly, are labeled with a type name of the topological relationship together with additional attributes.
- If the layout contains a certain amount of spanning objects like pipes or beams then it is more suitable to represent the spanning objects as edges and the remaining objects as vertices and, if it is necessary, to label with their type name and necessary additional attributes.

The function *recompile* transforms vertices and edges of an adapted graph into objects and relations of a concrete layout. In case of labeled graphs this mapping is unique. Otherwise the problem of nonunique mappings has to be solved.

Organization of the Case Base: In this approach, we consider arbitrary graphs which may be either directed or undirected and, possibly, labeled or unlabeled. Computing structural similarity is essentially based on graph matching which actually means computing the maximal common isomorphic subgraphs. The module MACS uses a backtracking algorithm [23] that realizes the function *match* to compute maximal common subgraphs of two arbitrary graphs [24]. A maximal common subgraph of two graphs denotes their *structural similarity.* Because of our key concept of structural similarity, for any collection of graphs $C \in \mathcal{P}(\Gamma)$, we use $mcs(C)$ to denote the set of maximal common subgraphs of all graphs in C with respect to counting vertices and edges (cf. [24]).

In general, the structural similarity of a set of graphs is not unique. Therefore, it may be represented by a set of graphs. Let us introduce some *selection operator* $E : \mathcal{P}(\Gamma) \rightarrow \Gamma$ to determine a unique representative $\Theta \in mcs(C)$ for each class C.

Usually, the peculiarities of the application domain lead to a couple of preferable selection operators. There are some illustrative possibilities, e.g.

- *other similarity concepts* based on labels of vertices or edges of graphs within $mcs(C)$,
- *graph–theoretic properties* which characterize structures preferred in the domain.

When $mcs(C)$ and E are given, the structural similarity of any class C of graphs may be written as $\sigma(C) = E(mcs(C))$ in the sense of [10].

Because of NP-completeness of the subgraph isomorphism problem, that is the computing of a common isomorphic subgraph, classical case retrieval is extremely expensive, if every member of a usually huge case base is potentially queried. The performance of this search can be increased by prestructuring the case base in case classes, i.e. clustering.

Preferably, a given case base CB is clustered with respect to structural similarity, i.e. graphs of a close structural relationship are grouped together and represented by a graph describing their structural similarity. Let us assume that a case base CB is separated in n partitions or classes where the finite number n may be approx. $\sqrt{N}$ with $N = \mid CB \mid$. Each class CB_i $(i = 1, ..., n)$ consists of a set of graph-represented cases and is determined by a graph Θ_i which is called its *representative.*

Fast Retrieval over a Structured Case Base: We may assume that the case base is partioned in n finite classes and that a representative of every class is computed by $\Theta_i = \sigma(CB_i)$, $(i = 1, ..., n)$ with an optimal value of $n \approx \sqrt{N}$.

There is a basic scenario for retrieving similar cases to a given problem (the query case g). The *fast retrieval* proceeds in two steps:

1. The retrieval yields a maximal similar representative Θ_i to the given query case. The resulting class is indexed by some i^*. (Note that there may be several maximal results being mutually incomparable.)
2. Over the set CB_{i^*} of preferred cases, retrieval is performed again to return the ultimate result.

The fast retrieval allows to reduce the number of necessary comparisons to $2n$ in the average case. Usually, the result is a set of cases of CB_{i^*} having a structural similarity of maximal size. Those are returned for further reasoning procedures like case adaptation. The advantage over conventional approaches is quite obvious.

In case of MACS, the result of retrieval is a set of *source* cases denoted by $S_{i^*} \subseteq CB_{i^*}$. Each case $g^j_{i^*} \in S_{i^*}$ has the property that each element of $mcs(\{g^j_{i^*}, g\}) = match(g^j_{i^*}, g)$ has the equal number of vertices and edges, i.e. the size of structural similarity of each graph of S_{i^*} and the query case is identically.

Adaptation: The selected cases are proposed one after the other to a checking algorithm or to the user, who selects the most suitable one.

In general we have smaller query cases than source cases, hence it is possible that a chosen source case solves a problem itself. Otherwise, the selected source case have to be adapted to the problem. Normally, in CBR *adaptation* means the transformation of a chosen source case using informations which are included in a knowledge-base and several kinds of substitutions of a query case.

The module MACS realizes a structure modification of query cases doing simple substitutions exclusively which result is the supplemented query case. We may distinguish two basic scenarios to realize the function *adapt* for a query case g:

1. We select a source case $g'_{i^*} \in S_{i^*}$. Further, let h be a structural similarity of g and the chosen source case g'_{i^*}. Using the result of $match(g, g'_{i^*})$, we get a mapping list of corresponded vertices of g and g'_{i^*} which define the graph h. The *adapted graph* (solution) is generated from graph g by adding all walks in g'_{i^*}, which have not an isomorphic mapping in g but which begin and end with vertices of h. These walks may be sequences of edges which are incident with vertices not corresponding to a vertex of g excluding the begin and end vertices.
2. We use the whole set S_{i^*} for adaptation. Let h be a structural similarity of g and S_{i^*}. An *adapted graph* (solution) is generated from graph g by adding all walks of graphs of S_{i^*} that have the same property as described above. The user will have to choose between using all graphs of S_{i^*} or special graphs only.

The module MACS provides the first variant of adaptation exclusively. It would require a different scenario to realize the second variant of adaptation. Especially, the creation of some choosing heuristics based of a different graph representation or the specification of extensive user interaction would be necessary.

4.3 CA/SYN

Conceptual analogy (CA) is a general approach that relies on conceptual clustering to facilitate the efficient use of past cases in analogous situations [7]. CA divides the overall design task into *memory organization* and *analogical reasoning* both processing structural case representations. In order to ground both processes on attribute value representations of cases a *compile function* and a *recompile function* need to be defined.

Compile and Recompile: The function *compile* guarantees the unique transformation of an attribute value represented case into its structural normal form, i.e., a tree. Especially suited for the design of pipelines, *compile* maps outlets into vertices and pipes into edges. Inversely, *recompile* maps vertices and edges into outlets and pipes. Geometrical transformations like rotation are considered.

Representing the main access by a square, outlets by circles, interconnecting points by circles of smaller size and pipes by line segments, Fig. 4 (left bottom) illustrates six cases representing pipe systems. Each of them shows a tree like structure. The main access corresponds to the root (R), outlets correspond to leave (L). Crosspoints of pipes or connections of pipe segments are represented by internal vertices (I). Pipes correspond to edges. Each object is placed on the intersecting points of a fixed grid (not shown here) and can be uniquely identified by its x and y coordinates and its $type \in \{R, I, L\}$. Pipes connect objects horizontally or vertically. Thus a case can be represented by a set of vertices and a set of edges representing *connected_to* relations among these objects. Formally, a *case* $c = (V^c, E^c)$ is a tree. A *case base* CB is a finite set of cases. A typical design *problem* provides the main access, the outlets, and perhaps some pipes, i.e., it is a forest. A *solution* of a problem contains the problem objects and relations and eventually adds intermediate vertices from past cases and provides the relations that are required to connect all outlets to the main access.

Memory Organization starts with a case base (CB) providing a significant amount of cases as well as a structural similarity function σ.

To explain memory organization some basic definitions will be given first. A *case class* CC is a nonempty subset of CB. Let $mcs(CC)$ be the (unique) maximum common, connected subgraph of the cases in CC containing the root vertice. The structural similarity[3] is defined as $|E^{mcs}|$ divided by the total number of edges in the cases of CC:

$$\sigma(CC) = \frac{|E^{mcs}|}{|\cup_{c \in CC} E^c|} \in [0,1].$$

Given a case base and a similarity function σ a *case class partition* CCP is a set of mutually disjoint, exhaustive case classes CC: $CCP = \{CC_i \mid \bigcup_i CC_i = CB \wedge \forall i \neq j (CC_i \cap CC_j = \emptyset) \wedge \forall i \neq j ((c_1, c_2 \in CC_i \wedge c_3 \in CC_j) \rightarrow \sigma(c_1, c_2) \geq \sigma(c_1, c_3) \wedge \sigma(c_1, c_2) \geq \sigma(c_2, c_3))\}$. A *case class hierarchy* CCH is the set of all partitions CCP^v of $CB = \{c_1, .., c_N\}$: $CCH = (CCP^0, CCP^1, ..., CCP^{n-1})$.

Given a set of cases $CB = \{c_1, .., c_N\}$, represented by trees, memory organization starts. Nearest-neighbor-based, agglomerative, unsupervised conceptual clustering is applied to create a hierarchy of case classes sharing cases of similar structure. It begins with a set of singleton vertices representing case classes, each containing a single case. The two most similar case classes CC_1 and CC_2 over the entire set are merged to form a new case class $CC = CC_1 \cup CC_2$ that covers both. This process is repeated for each of the remaining $N-1$ case classes, where N is the number of cases in CB. Case class merging continues until a single all-inclusive cluster remains. Thus at termination a uniform, binary hierarchy of case classes is left.

Subsequently, a concept description $K(CC)$ is assigned to each case class CC. The concept represents the $mcs(CC)$ (named prototype) of the cases in

[3] Note that the structural similarity function is commutative and associative. Thus it may be applied to a pair of cases as well as to a set of cases.

CC and a set of instantiations thereof, along with the probability of these instantiations. The probability of an instantiation corresponds to the number of its occurrences in the cases of CC divided by the total number of cases in CC. The *mcs* denoting the structure relevant for similarity comparisons will serve as an index in the case base. The instantiations (subtrees) denote possibilities for adaptation. Probabilities will direct the search through the space of alternative instantiations.

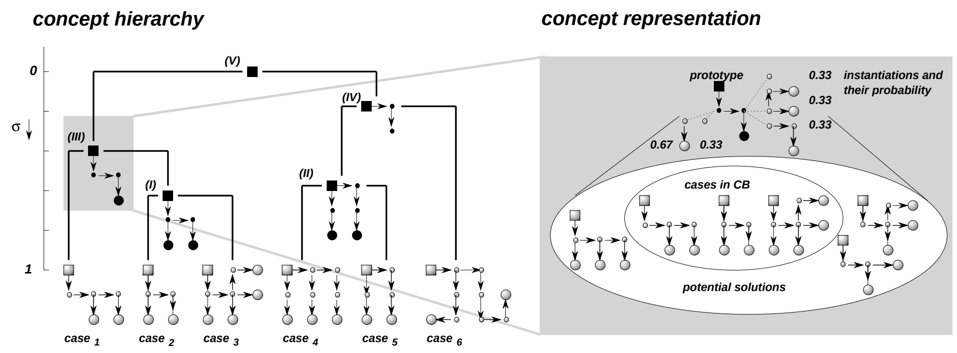

Fig. 4. Concept hierarchy and concept representation

In such a way, large amounts of cases with many details can be reduced to a number of hierarchically organized concepts. The concrete cases, however, are stored to enable the dynamic reorganization and update of concepts.

Analogical Reasoning is based on concepts exclusively. Given a new problem, it is classified in the most applicable concept, i.e., the concept that shares as many relations as possible (high structural similarity) and provides instantiations that contain the problem objects that are not covered by the prototype (adaptability)[4]. Thus instead of *retrieving* one or a set of cases, the function *classify* maps a concept hierarchy $K(CCH)$ and a problem p into the most applicable concept $K(CC)$.

Next, the *mcs* of the most applicable concept is transferred and instantiated. Instantiations of high probability are preferred. Each solution connects all problem objects by using those objects and edges that show the highest probability in the concept applied. In general, there exist more than one solution. The set of solutions for a problem and a case base may be denoted by $S_{CB,p}$. Instead of *adapting* one or more cases to solve the problem, the function *instantiate* maps the concept representation $K(CC)$ and the problem into a set of adapted solutions $S_{CB,p}$.

[4] Note that the most similar concept may be too concrete to allow the generation of a solution. See also [22] for a discussion and experiments on adaptation-guided retrieval.

Finally, the set of solutions may be ordered corresponding to a set of preference criteria: (1) max. structural similarity of the solution and the concept applied, (2) max. probability of edges transferred, and (3) min. solution size.

If the solution was accepted by the user, its incorporation into an existing concept changes at least the probabilities of the instantiations. Given that the problem already contained relations, it might add new instantiations or even change the prototype itself. If the solution was not accepted, the case memory needs to be reorganized to incorporate the user provided solution.

Fig. 4 (left) depicts the organization of cases into a concept hierarchy. N cases are represented by $2N-1$ case classes resp. concepts $K(CC)$. Leave vertices correspond to concrete cases and are represented by the cases themselves. Generalized concepts in the concept hierarchy are labeled (I) to (V) and are characterized by their *mcs* (prototype) denoted by black circles and line segments. The representation of concept no. (III) representing $case_1$ to $case_3$ is depicted on the right hand side of Fig. 4. The instantiation of its prototype results in $case_1$ to $case_3$ as well as combinations thereof. Given a new problem, the most applicable, i.e., most similar concept containing all problem objects is determined. The set of problems that may be solved by concept no. (III) corresponds to the set of all subtrees of either concrete or combined cases, containing the root vertice.

The general approach of *Conceptual Analogy* has been fully implemented in SYN, a module of a highly interactive, adaptive system architecture [8]. Its compile and recompile function is especially suited to support the geometrical layout of pipe systems. See [6, 9] for a detailed description of the implementation.

4.4 Example

Figure 5 provides an example of structural similarity assessment and adaptation. Depicted on the top is the query case or problem, the middle presents a source case used by TOPO to generate the adapted solution on the right hand side. The bottom line illustrates the application of a concept to generate a design solution. The problem contains a set of supply accesses, a main access as well as some pipes. In the source case, all accesses are connected by pipes.

As for the application of TOPO (middle), dark arrows denote spatial relations of the maximal common subgraph of the graphs of both cases. If there is no maximum common subgraph, one is chosen by chance. The paths of relations connected to the chosen common subgraph are transferred incrementally to generate the adapted case.

It is not always desirable to transfer all paths, as the ones leading to outlets A and H. Therefore, the designer is asked before transfer, or he can repair the layout, or domain-specific heuristics must be applied. During transfer TOPO may change the size of transferred objects to preserve spatial relations. For example a window is resized in order to touch both sides of a room. To avoid geometries which are impossible for an object, TOPO limits the resizing to geometries which occurred in the case base.

The application of MACS may retrieve to the same source case as the one used by TOPO requiring a higher match complexity and the handling of nonuniform mappings. The application of MACS *adapt* function transferes all pipes that are necessary to connect all problem outlets. Again the problem of non-unique mappings needs to be solved.

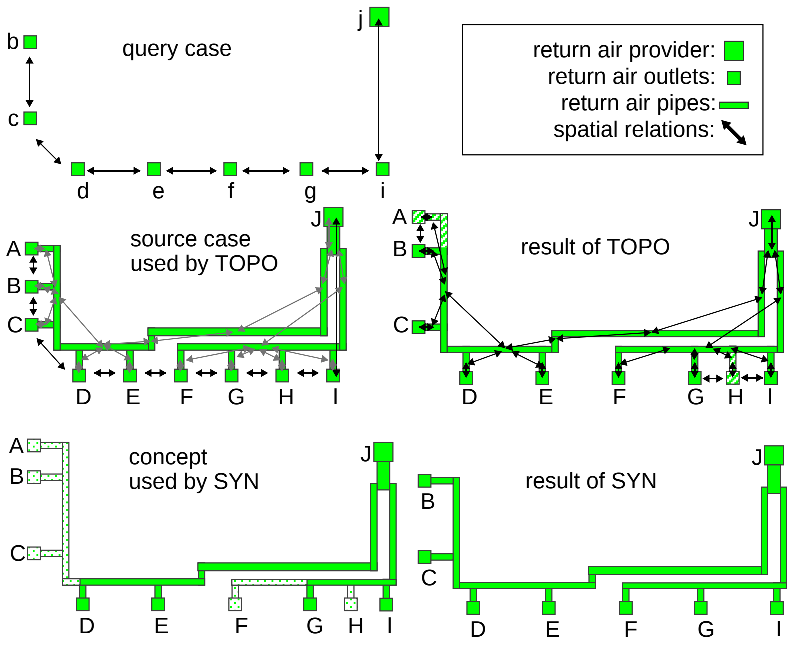

Fig. 5. Examples of structural similarity assessment and adaptation

Instead of transferring parts of a single case, SYN selects the most applicable concept of the respective concept hierarchy. This concept shows the highest structural similarity of its cases and contains all problem objects. It is characterized by its prototype (dark grey) and its instantiations (light grey) on the left hand side. The prototype is transferred to the query case connecting outlets D, E, G, and I to the main access J. The remaining problem outlets B, C, and F are connected via appropriate instantiations resulting in a correctly adapted solution.

4.5 Comparison

All three approaches use structural similarity assessment and adaptation to retrieve and transfer case parts to solve new problems. However, the approaches differ in their focus on different parts of the CBR-scenario. Subsequently the strength and limitations of each approach and its domain specific and domain independent parts are discussed.

First of all it must be noticed that the definition of the compile and recompile function strongly depends on the domain and task to support. The higher the required expressability of structural case representations the more complex are graph matching, i.e., the less efficient are retrieval and adaptation. The application of labeled graphs (TOPO) or trees (SYN) allows to invert the function *compile* to *recompile*. This can not be guaranteed for arbitrary graphs (MACS). Whereas the representation of cases by trees (SYN) guarantees unique *mcs*, this does not hold for graph representations, as in TOPO or MACS. Domain specific selection rules need to be defined or extensive user interaction is necessary to select the most suitable *mcs*. This may be advantageous during retrieval allowing the selection of different points of view on two graphs (being the *mcs* and a problem) but may not be acceptable for memory organization.

While TOPO does no retrieval at all, MACS and SYN retrieve a set of cases from a dynamically organized case base. MACS uses a two-level case organization. The lower level contains the concrete cases grouped into classes of similar cases. The upper level contains graphs describing the *mcs* of classes of similar cases. It does a two stage retrieval selecting the most similar *mcs* first and searching in its cases for the most similar concrete case(s). SYN uses a hierarchical memory organization, i.e., a case class hierarchy. Each case class is represented intentionally by a concept representing the unique *mcs* and *instantiations* which denote possibilities for adaptation as well as their *probabilities*. Given a new problem, the most applicable concept of the concept hierarchy is searched for.

In order to compare graph representations, MACS and TOPO need to apply graph matching algorithms (cliquen search and backtracking) that are known to be NP-complete. For this reason TOPO uses ASPECT for retrieval, reducing retrieval to one computationally expensive match between a selected case and the problem. MACS case organization allows to reduce the number of matches required to search through N cases to $N \times \sqrt{N}$. The restriction to trees (SYN) reduces expressibility but offers the advantage to match efficiently.

An important peculiarity of the module TOPO is to investigate the compatibility of object types and relation types of layouts. The case adaptation of TOPO searches for an object of the source case compatible to the object of the query which is related to compatible objects, but by more frequent topological relations. Using the recompile functions of those more frequent relations the new position of the object of the query case is determined.

The preference of MACS involves to structure the case base dynamically and to realize fast retrieval. MACS is suitable to investigate the possibilities of learning the partition of a case base with respect to the structural similarity and their representatives. MACS realizes a simple variant of case adaptation by adding all walks of the source case, which have not an isomorphic mapping in the query case but begin and end with vertices of their *mcs*.

CA/SYN definitely concentrates on efficient structural case combination. Therefore, the approach integrates the formation of hierarchically organized concepts (i.e., concept hierarchies) and the application of these concepts during analogical reasoning to solve new problems. It is unique in its representation of concepts by the *mcs* and its *instantiations* plus *probabilities*. Its definition of applicability allows the efficient selection of the most similar concept that is neither too general not to concrete and guarantees the generation of a problem solution. The instantiation of its *mcs* is guided by the probabilities of these instantiations, resulting in the optimal solution.

5 Related Work and Discussion

Interactive design systems as ARCHIE [19] or CADRE [15] do either support retrieval or adaptation of past designs. Systems like CASEY, KRITIK or IDEAL [5] integrate case-based and model-based reasoning to support retrieval as well as adaptation. They require model-based knowledge which is not available in our domain. Cases are most commonly represented by fixed sets of attribute value pairs. Static libraries of hand-coded adaptation rules are used to transform past cases into new solutions.

There is a number of CBR approaches that apply conceptual clustering techniques in organizing their case base. The *Prototype-Based Indexing System* (PBIS) proposed by [18] uses an incremental prototype-based neural network to organize cases into groups of similar cases and to represent each group of cases by a prototype. The JANUS CBR Shell [21] applies a Cohonen network to automatically organize cases into disjoint case classes corresponding to similar attribute values. It represents these classes by reference cases. Both systems do a two stage retrieval. Firstly, the prototype/reference case pointing to a case class is selected. Secondly, this case class is searched for the most similar concrete case. Its solution is presented as the actual classification. However, both systems are restricted to attribute value representations of cases and support classification tasks.

To our knowledge, the approaches introduced in this paper are unique in their handling of structural, i.e., graph-based case representations, their definition and application of structural similarity functions and the structural adaptation of cases without further domain knowledge. They enable to reuse highly structured cases of varying sizes to support design tasks. As for memory organization, conceptual clustering techniques can be advantageously applied to reduce reasoning complexity. The representation of case classes by their *mcs* or by concepts, effectively short cuts much of the memory retrieval effort that would be necessary to check each past case separately for structural similarity. The complexity of structural mappings can be handled in an economical manner and realistic response times become possible.

6 Acknowledgements

Thanks to D. W. Aha, K. P. Jantke, M. M. Richter, A. Voßand the anonymous referees for their many critical comments that helped shape this research. The research was supported by the German Ministry for Research and Technology (BMBF) within the joint project FABEL under contract no. 413-4001-01IW104.

Project partners in FABEL are German National Research Center of Computer Science (GMD), Sankt Augustin, BSR Consulting GmbH, München, Technical University of Dresden, HTWK Leipzig, University of Freiburg, and University of Karlsruhe.

References

1. J. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.
2. Luitpold Babel and Gottfried Tinhofer. A branch and bound algorithm for the maximum clique problem. *ZOR – Methods and Models of Operations-Research*, 34:207–217, 1990.
3. H. G. Barrow and R. M. Burstall. Subgraph isomorphism relational structures and maximal cliques. *Information Processing Letters*, 4:83–84, 1976.
4. Brigitte Bartsch-Spörl and Elisabeth-Ch. Tammer. Graph-based approach to structural similarity. In Angi Voß, editor, *Similarity concepts and retrieval methods*, pages 45–58. GMD, Sankt Augustin, 1994.
5. S. Bhatta and A. Goel. From design cases to generic mechanisms. *AI EDAM*, 10, 1996.
6. Katy Börner. Structural similarity as guidance in case-based design. In Wess et al. [25], pages 197–208.
7. Katy Börner. Conceptual analogy. In D. W. Aha and A. Ram, editors, *AAAI 1995 Fall Symposium Series: Adaptation of Knowledge for Reuse*, pages 5–11, November 10-12, Boston, MA, 1995.
8. Katy Börner. Interactive, adaptive, computer aided design. In Milton Tan and Robert Teh, editors, *The Global Design Studio – proceedings of the 6th international conference on computer-aided architectural design futuresB95*, pages 627–634, Singapore, 1995. University of Singapore.
9. Katy Börner and Roland Faßauer. Analogical Layout Design (Syn*). In Katy Börner, editor, *Modules for Design Support*, pages 59–68. GMD, Sankt Augustin, June 1995.
10. Katy Börner, Klaus P. Jantke, Siegfried Schönherr, and Elisabeth-Ch. Tammer. Lernszenarien im fallbasierten Schließen. Fabel-Report 14, GMD, Sankt Augustin, December 1993.
11. C. Bron and J. Kerbosch. Finding all cliques in an undirected graph. *Communications of the ACM*, 16:575–577, 1973.
12. Carl-Helmut Coulon. Automatic Indexing, Retrieval and Reuse of Topologies in Architectural Layouts. In Milton Tan and Robert Teh, editors, *The Global Design Studio – proceedings of the 6th international conference on computer-aided architectural design futures*, pages 577–586, Singapore, 1995. Centre for Advanced Studies in Architecture, National University of Singapore.
13. Eric A. Domeshek and Janet L. Kolodner. A case-based design aid for architecture. In *Proc. Second International Conference on Artificial Intelligence in Design*, pages 497–516. Kluwer Academic Publishers, 1992.
14. Ashok K. Goel. *Integration of case-based reasoning and model-based reasoning for adaptive design problem solving.* PhD thesis, Ohio State University, Columbus, Ohio, 1989.
15. Kefeng Hua and Boi Faltings. Exploring case-based building design – CADRE. *AI EDAM*, 7(2):135–144, 1993.

16. Klaus P. Jantke. Nonstandard concepts of similarity in case-based reasoning. In H. H. Bock, W. Lenski, and M. M. Richter, editors, *Information Systems and Data Analysis: Prospects–Foundations–Applications*, pages 29–44. Springer Verlag, 1994.
17. Janet L. Kolodner. *Case-Based Reasoning.* Morgan Kaufmann, San Mateo, 1993.
18. M. Malek and B. Amy. A pre-processing model for integrating CBR and prototype-based neural networks. Technical report, Working Paper TIMC-LIFIA-IMAG Grenoble, 1994.
19. M. Pearce, A. K. Goel, J. L. Kolodner, C. Zimring, L. Sentosa, and R. Billington. Case-based design support: A case study in architectural design. *IEEE Expert*, pages 14–20, October 1992.
20. Jörg Walter Schaaf. "Fish and Sink"; An Anytime-Algorithm to Retrieve Adequate Cases. In Manuela Veloso and Agnar Aamodt, editors, *Case-based reasoning research and development: first International Conference, ICCBR-95, proceedings*, pages 538–547. Springer, Berlin, October 1995.
21. Ingo Schiemann and Ansgar Woltering. Organisation großer Fallbasen in der TUB-JANUS Shell zum effizienten Retrieval geeigneter Fälle. In Richter M. M., editor, *Workshop Fallbasiertes Schließen: Grundlagen und Anwendungen, Deutsche Expertensystemtagung XPS-95*, LSA-95-02, pages 30–36, 1995.
22. Barry Smyth and Mark T. Keane. Retrieving adaptable cases: The role of adaptation knowledge in case retrieval. In Wess et al. [25], pages 209–220.
23. Kathleen Steinhöfel. Backtrack Algorithmus zur Suche des größten gemeinsamen Teilgraphen. HTWK Leipzig, 1995. Dokumentation.
24. Elisabeth-Ch. Tammer, Kathleen Steinhöfel, Siegfried Schönherr, and Daniel Matuschek. Anwendung des Konzeptes der Strukturellen Ähnlichkeit zum Fallvergleich mittels Term- und Graph-Repräsentationen. Fabel-Report 38, GMD, Sankt Augustin, September 1995.
25. Stefan Wess, Klaus-Dieter Althoff, and Michael M. Richter, editors. *Topics in Case-Based Reasoning – Selected Papers from the First European Workshop on Case-Based Reasoning (EWCBR-93)*, volume 837 of *LNAI*. Springer Verlag, 1994.