On Graph-Based Knowledge Representation in Design

Adam Borkowski[1], Ewa Grabska[2] and Janusz Szuba[3]

Summary

Based upon the graph-oriented knowledge representation two problems are considered: transforming functional requirements into layouts of houses and searching for optimum topology of trusses. It is shown that both problems can be conveniently formulated and then efficiently solved be means of specialized graph transformation rules.

Keywords: graph transformations, knowledge representation, floor layout, optimum topology

Introduction

Prior to designing any artifact one has to develop clear understanding of the purpose that the object should serve. In software engineering this understanding is achieved by working out a detailed description of functional requirements to be met by the designed system. The use case diagrams and the activity diagrams provided by the Unified Modeling Language (UML) proved to be very helpful in accomplishing the specification of system's functionality. In the first part of the paper we show how the same tools can be applied for describing the functionality of a building.

The main aim of this project is to create the system aiding the early phase of architectural designing and to accomplish that by means of graph transformations. Contrary to conventional expert systems proposed previously in this domain (Flemming et at, 1999) our system can be seen as a knowledge-based front-end of a CAD-tool. We restrict our consideration to the rather simple

[1] Professor, Institute of Fundamental Technological Research, Polish Academy of Sciences, Warsaw, Poland, abork@ippt.gov.pl
[2] Dr. Habil., Institute of Computer Science, Jagiellonian University, Cracow, Poland, ui-grabsk@cyf-kr.edu.pl
[3] Doctoral Candidate, Institute of Fundamental Technological Research, Polish Academy of Sciences, Warsaw, Poland, jszuba@ippt.gov.pl

example of designing a single-store family house but the methodology remains valid for designing other types of buildings. The system, which we called GraCAD, should help the architect to create designs that fulfill specified requirements. The GraCAD is integrated with the commercial program for architects (ArchiCAD, 2000).. The graph rewrite tool that we use is the PROGRES developed at the RWTH Aachen (Schürr, Winter and Zündorf, 1995).

A special form of graph-based representation has been developed by the second author (Grabska, 1994). It turned out that by introducing an additional functionality graph into the original model one can conveniently reason about conceptual solutions for the designed object. The functionality analysis of as the starting point of the conceptual design has been proposed by several researchers (Borkowski, Grabska and Hliniak, 1999), (Cole, 1998).

In the second part of the paper we consider the topological optimization of structural systems. An exhaustive presentation of the state-of-art in the topological optimization can be found in (Kirsch, 1995).

Reasoning About Function

Our proposal amounts to giving the designer a computer-based tool that allows him to reason about functional requirements and to transform them into the structural scheme of the designed object. In order to achieve that, we distinguish four phases of the design process:

1) Specifying functional requirements for the designed object.
2) Transforming them into the structure of the object.
3) Visualising the object.
4) Working out the detailed design.

We propose the initial two steps to be graph-based. Firstly, the designer takes the list of required functions and constructs a functionality graph. The nodes of this graph correspond to the functions that the considered artifact has to fulfill. The edges connecting the nodes depict functional relationships.

In the second phase the functionality graph is mapped into a structural graph of the object. The nodes of the structural graph correspond to the components of the object; the edges represent relations between the components. Thus, the structural graph describes a physical decomposition of the artifact. By assigning the functions to the components one aims at satisfying all the functional requirements by the object viewed as an assembly of its components. The transition from the functionality graph to the structural graph is neither unique nor straightforward. Hence, the designer needs usually several iterative loops over the steps 1, 2 before the satisfactory solution is found.

The step 3 – the visualization of the object described by the structural graph – is performed automatically. Our system generates the floor layout of the building in the format accepted by the ArchiCAD. This system allows the user to visualize the building in 2D or 3D mode, to make any desired cross-section of the building and to assign its components the elements of the library of standard units, like the doors, the windows, etc. Usually the visualization reveals several drawbacks of the designed object. Such errors are removed by revisiting the steps 1 and 2. After

several loops the final solution is found and the project enters the step 4. The detailed design is accomplished in usual manner by means of the CAD-tool.

The advantages of using graph-based representation during the steps 1, 2 seem to be obvious. The designer is encouraged to think firstly in the abstract terms of functions and their relations. After the functionality graph is established, the designer tries to decompose the object into physical units assigning particular functions to them. The entire process is performed graphically by means of the editor that allows the designer to manipulate graphs conveniently.

Let us clarify the idea of our approach on an example of designing a single-family house. The user of our systems begins with defining functional requirements. This is accomplished by means of a convenient editor (Figure 1). The left part of it contains the list of functions arranged in a tree. The right part shows the functionality graph. The user can add or delete nodes representing functions, as well as perform editing of the edges representing functional relations.

In the second step the structural graph of the building is generated. The nodes of this graph represent the rooms of the house; the edges represent accessibility relations between rooms. After the structural graph has been generated, the designer evaluates it. Note that at this stage the level of abstraction remains high: the geometry is still irrelevant, what matters is the assignment of functions to particular rooms.

If the evaluation falls negative, the designer can modify the structure. In terms of graphs this means adding or deleting a node, adding or deleting an edge, merging two nodes into one, splitting a node into two, changing the type of node or changing the label of edge. Given the editor the architect is not bothered by the technicalities of the graph theory. Architects are trained in visual reasoning. Therefore, they find this tool quite intuitive.

Generating Floor Layout

After the designer finds the object structure acceptable, the layout of the house is automatically generated and displayed by the ArchiCAD system. The floor layout generator scans the structural graph of the house node by node and creates "embryos" of rooms: the objects that have already walls but are of square shape with minimum allowable area. Such embryos are placed inside the preliminary contour of the floor according to heuristic rules. Then they are allowed to expand until there is no free space between adjacent rooms. The preliminary outer contour is allowed to adjust itself in order to accommodate all necessary rooms.

The initial placement of room embryos follows three predefined patterns. The choice of pattern depends upon the global area of the house requested by the client. After the position and the size of each room have been found, the generator begins to place doors and wall openings. These elements are located according to the accessibility edges given in the structural graph. Additional requirements coming from the codes of practice are also taken into account.

Figure 2 shows two alternative floor layouts obtained by means of the prototype generator. Both solutions were derived according to the "small house"

pattern. They differ in the number of bedrooms governed by the number of inhabitants and in the position of the main entrance.

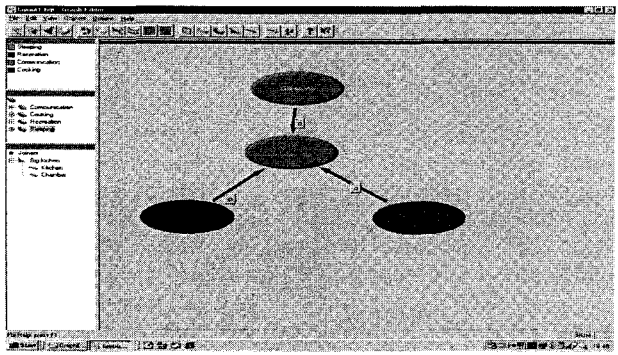


Figure 1: Editor of functionality graph.

Specifying Project

The graph rewrite system PROGRES (Schürr, Winter and Zündorf, 1995) serves in our prototype the following purposes:

1) It provides access to the domain knowledge stored in the form of graph grammar. At present this grammar encompasses single-family houses but the knowledge base can be easily extended to cover other domains.
2) It provides means of constructing graphs that describe the currently designed object.
3) It allows the user to trigger rules that check whether the current design fulfils architectural norms and other constraints.

a) b)

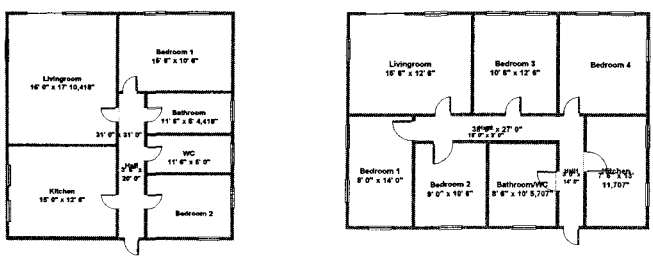
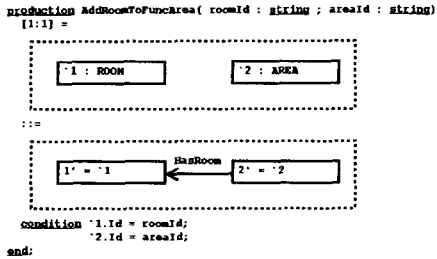


Figure 2: Two alternative floor layouts.

In our system every operation during specification of functional requirements or creating the prototype design is invoking the PROGRES graph transformation that modifies the graph of the house structure. For example, assigning a function to the room is equivalent to invoking the following PROGRESS rule:



Note that in ArchiCAD the layout of the building is coded in terms of walls, whereas the main building block in our system is a room. Therefore, seamless transformation between both formats of knowledge representation must be granted for the user. Each time when the user inserts a room into the prototype design window the system creates four instances of the class *Wall*. Their attributes contain data defining the location of each wall in the global co-ordinate system. The relation between the room and its walls is established by connecting *Wall* nodes with the *Room* node by means of the *contains* edges. The graph of house structure created this way is passed to the *RoomAdjuster* and to the rule checkers. The latter are implemented as graph transformation rules in PROGRESS.

Reasoning About Topology

Let us turn our attention now to the optimum design of skeletal structures. Conventional optimization assumes that the layout, i.e. the number of nodes, their positions as well as the connections between the nodes, is given. Selecting the best values for the attributes of structural components leads usually to the gain in cost or weight of the order of 5 to 10 %. On the other hand, finding the most favorable configuration of the structure may diminish the same cost indicator by 20 or 30 %.

The above-mentioned circumstance inspired many researchers to look for a way of finding the best structural layout (topology) automatically. In particular, the sensitivity analysis developed during last decades seemed, at the first glance, to enable the search for optimum topology in a gradient-based manner. Unfortunately, this way turned out to be much more difficult than expected (Sokołowski and Żochowski, 1999). The main obstacle in calculating sensitivities with respect to topological changes is the non-smooth nature of the problem. Introducing a new structural element or changing the way in which elements are connected brings abrupt changes in the cost function.

The first attempts to employ genetic optimizers that require no sensitivity data were quite promising (Achtziger, 1995), (Eschenauer, Kobelev and Schumacher, 1994). In the second part pf this paper we report preliminary results

on developing the graph-based methodology for the topological optimization. A prerequisite for applying genetic search for the considered problem was an introduction of the novel definitions of the crossover operator and the mutation operator (Hliniak and Strug, 2000). The new operators work on graphs and take into account constraints introduced by dependency on the context. This allows siblings to inherit certain semantics from parents. As a result, the search is less burdened by meaningless solutions.

Structural optimum design can be considered at three levels: 1) the component level; 2) the shape level; 3) the topology level. These levels build a hierarchical ontology leading towards higher abstraction. Strictly speaking, all three levels are mutually coupled but a brute force attempt to solve the problem in one step would be too expensive computationally. Therefore, in practice the coupling between levels is either completely or partially neglected. Solutions obtained this way are sub-optimal as compared to the exact solution.

For the sake of simplicity we prefer to illustrate all notions on planar trusses since generalization to the 3D-space and to the systems in bending is straightforward. Let the optimized structure consist of m elements (bars) connected at n nodes. Following the object-oriented methodology we represent the properties of an object by its attributes. The attributes of a generic element will be its length, cross-section, material, etc. The main attributes of a generic node are its co-ordinates in the global reference frame.

Instead of treating each attribute of an individual structural element as an independent design variable, it is preferable to split the entire set $\mathcal{M}$ of structural members into relatively small number of subsets $\mathcal{M}_k$ grouping homogeneous elements. The attributes of the k-th group are taken then as the design variables at the component level of optimum design. Similar reasoning leads to imposing internal structure on the entire set $\mathcal{N}$ of nodes. Firstly, this set is divided into the subsets $\mathcal{N}_{fix}$ and $\mathcal{N}_{var}$ containing the nodes that are fixed and variable, respectively, with respect to their position. As a rule, $\mathcal{N}_{fix}$ includes the supports of the structure but the points of loading can also enter this subset.

At the second level of optimum design we are looking for the optimum shape of the structure. Let S be a curve describing such shape. This curve allows us to partition $\mathcal{N}$ into the subset $\mathcal{N}_{ext}$ of external, i.e. lying on S, nodes and the subset $\mathcal{N}_{int}$ of internal nodes. A common assumption adopted in the shape optimization is $\mathcal{N}_{int} = \emptyset$ which implies $\mathcal{N}_{ext} = \mathcal{N}_{fix} \cup \mathcal{N}_{var}$. Describing parts of S by Bsplines or NURBS allows us to diminish further the dimension of the shape optimization problem.

Both the component optimization and the shape optimization are well represented in the literature on optimum structural design. Let us treat them as being satisfactorily solved and let us confine our attention to the highest level of abstraction, namely, to the problem of topological optimization. Let $\mathcal{P}$ be a set of primitives – building blocks that constitute our structure. We understand then as topology a graph $\mathcal{T}$ describing connections between primitives. A brute force approach would be to take $\mathcal{P} = \mathcal{M}$ and to look for $\mathcal{T}_{opt}$. Note that in order to evaluate topology in terms of the structural weight one has to solve both lower level problems: a) find optimal

positions for nodes; b) find optimal attributes for components. We are looking at present for reasonable means of decoupling the above-mentioned sub-problems since otherwise the computational burden is too high.

It seems, however, that better results could be obtained if certain meta-knowledge is applied prior to starting the topological optimization. Structures used in practice exhibit substantial regularity: they are made of a finite number of topologically identical units - panels. Thinking in terms of panels reduces the dimension of the search space but finding the optimum topology still remains non-trivial task. The connectivity graph C applies now for panels. Even in the case of the popular cantilever truss principally different solutions are possible. Figure 3.a shows a conventional linear sequence of panels. The well known analytical solution found by Michell is depicted in Figure 3.b. This topology consists of three levels. At the highest level the "onion" type structure of the layers A, B, C and D is seen. Each of those layers is subdivided into the panels 0, 1, 2, L at the intermediate level. The lowest level corresponds to the triangles of the type shown in Figure 4.a that are substructures of individual panels.

Searching For Solution

In our system a CP-graph is used as a genotype and a realization scheme serves as means for transforming it into a phenotype. When a crossover is performed on two graphs G_1 and G_2 the sub graphs g_1 and g_2, respectively, are selected in these graphs. Then each sub graph is removed from a parent graph and put into the sibling graph. As a result, two new topological solutions are generated. However, there usually exist edges connecting nodes belonging to a selected sub graph with nodes not belonging to it. Such edges are called embedding of a sub graph. Hence, removing a sub graph from a graph and putting it into another one requires a method allowing for a proper reconnection of these edges. The underlying idea is that all edges should be re- connected to similar nodes they were connected to in the graph from which they were removed. There is probably more than one possibility of defining which nodes are similar. In this paper two nodes are said to be similar if they have identical labels. More formally a crossover operator is defined as a 6-tuple (G_1, G_2, g_1, g_2, T, U), where G_1, G_2, g_1, g_2 are hierarchical graphs and their sub graphs respectively. The crucial elements of this operator are T and U that are called embedding transformations. They describe how edges of the embedding are to be re-connected. Embedding transformations are sets of pairs of the form (b, b'), where b denotes a bond to which an edge was connected originally and b' – the one to which it will be re-connected.

It is important to notice, however, that the graphs to be crossed-over and their respective sub graphs are selected during the execution of the genetic algorithm. Therefore, the embedding transformations can not be defined a priori as it is done in graph grammars (Rozenberg, 1997). The most difficult problem is to find a method allowing these transformations to be established on-fly. We developed an algorithm that automatically generates the transformations T and U for any given two graphs and their sub graphs.

The mutation operator can be relatively easily defined for the graph-based representation. We consider changing at random the values of attributes of a

single node (local mutation), changing them for all nodes (global mutation), removing nodes or inserting them. So, while crossover allows us to generate artifacts being combinations of previously existing designs, mutations may introduce wholly new elements into the object being designed. Thus, they produce layouts that differ from the initial ones not only in sense of geometrical properties (like position) but also by structure (i.e. connections).

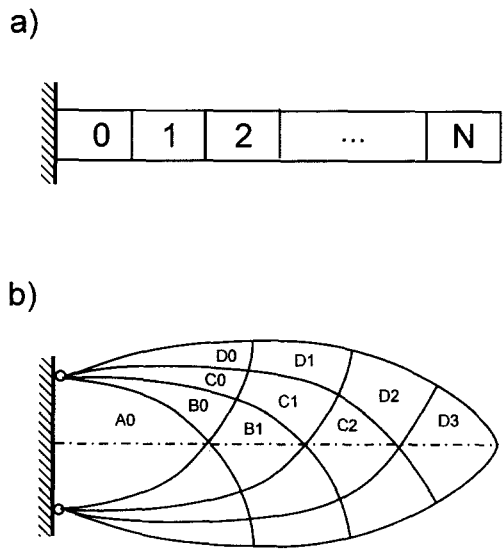


Figure 3: Alternative meta-level topologies for cantilever truss: a) linear solution; b) Michell's solution.

Conclusions

The interviews with designers confirm that an add-on to CAD-tool aiding the conceptual phase of design is worth developing. It increases the designer's productivity, shortens the way from the talk with the client to the prototype design, helps the designer to operate on the abstract conceptual level, facilitates the communication with the investor. The experience gained so far suggests that it is possible to hide the formalism of graph-oriented language from the user allowing him or her to work comfortably at the level of graphical interface. Further work has to be done in order to include more realistic design constraints and to enhance the power of the generative part of the system.

Our system for topological optimization is still under development. The results of numerical experiments will be presented at the workshop.

Acknowledgements

This project was partly supported by the Polish State Committee of Scientific Research (grant # 8T07A 01621) and the German Federal Ministry of Education and Research (joint Polish-German project "Graph-based tools for conceptual design in Civil Engineering") which is gratefully acknowledged.

References

Achtziger, W. (1995). "Multiplay load truss optimisation: properties of min-max compliance and two non-smooth approaches". In: *Proc. of the First World Congress of Structural and Multidisciplinary Optimization*, Pergamon Press, Oxford, 123-128.

Borkowski, A., Grabska, E. and Hliniak, G. (1999). "Function-structure computer-aided design model", *Machine GRAPHICS &VISION*, 9, 367-383.

Cole, E. L. Jr. (1998). "Functional analysis: a system conceptual design tool", *IEEE Trans. on Aerospace & Electronic Systems*, 34 (2), 354-365.

Eschenauer, H. A., Kobelev, V. V. and Schumacher, A. (1994). "Bubble method for topology and shape optimization of structure". *J. of Structural Optimization*, 8, 42-51.

Flemming, U., Coyone, R., Gavin, T., and Rychter M. (1999). "A generative expert system for the design of building layouts – version 2". In: B. Topping (Ed.), *Artificial Intelligence in Engineering Design*, Computational Mechanics Publications, Southampton, 445-464.

Grabska, E. (1994). "Graphs and designing". In: H. J. Schneider and H. Ehrig (Eds.), *Graph Transformations in Computer Science*, LNCS 776, Springer-Verlag, Berlin, 188–203.

Hliniak, G. and Strug, B. (2000). "Graph grammars and evolutionary methods in graphic design". *Machine GRAPHICS & VISION*, 9, 1 (2), 5–13.

Kirsch, U. (1995). "Reduction and expansion processes in topology optimization". In: *Proc. of the First World Congress of Structural and Multidisciplinary Optimisation*, Pergamon Press, Oxford, 95-102.

Rozenberg, G. (Ed.). (1997). *Handbook of Graph Grammars and Computing by Graph Transformation*, World Science, Singapore.

Schürr, A., Winter, A. and Zündorf A. (1995). "Graph grammar engineering with PROGRESS". In: W. Schäfer, P. Botella (Eds.), *Proc. 5th European Software Engineering Conference (ESEC'95)*, LNCS 989, Springer-Verlag, Berlin, 219-234.

Sokołowski, J. and Żochowski, A. (1999). „On topological derivative in shape optimization". *SIAM Journal on Control and Optimisation*, 37, No. 4, 1251-1272.