Software Tools and Optimal Design

ADVANCED COURSE ON STRUCTURAL CONTROL AND HEALTH MONITORING
SMART'01 – (PP.151–172) – WARSAW, MAY 22-25, 2001.

Graph grammars in conceptual design

A. BORKOWSKI

Polish Academy of Sciences
Institute of Fundamental Technological Research
Świętokrzyska 21, 00-049 Warszawa, Poland
e-mail: **aborkQippt.gov.pl**

This paper deals with new perspectives for conceptual design in engineering that are opened by graph grammars and graph transformations. Due to their expressive power and natural visualization these knowledge representation tools allow the designer to consider general properties of the artifact prior to be occupied by details. Two fields of Civil Engineering were chosen for validating the proposed methodology. The first one is the design of floor layouts for single-family houses. The second area is the optimum design of the layout of trusses. Despite their apparent dissimilarity both problems can be efficiently tackled by the graph-based approach. The paper considers the theoretical background and the computer implementation issues of the above-mentioned problems.

Key words: *CAD, Graph grammars, Conceptual design.*

1. Introduction

It is well known that a good concept is worth months of improving details of the project. In the ancient Greece many beautiful temples were erected but they all followed rather crude concept of a column-beam structure (Fig. 1a). As the result, Greek temples were filled with grids of columns standing at a distance of less than 5 m from each other. Reinventing the Sumerian arch in the ancient Rome brought revolutionary change in architecture. Roman aqueducts and bridges gained larger spans and the magnificent Pantheon (Fig. 1b) with its 43 m of free space inside makes us feel impressed even today.

In the course of maturing and collecting experience of many generations of architects and builders the domain of Civil Engineering became less and less susceptible to revolutionary changes. For example, the introduction of skeletal structures made from the cast iron in the 19th century (Fig. 1c) increased the span only by the order of 40% as compared to the Pantheon. Nevertheless, it is much more than the typical gain of 5 to 10% achievable by the optimization within the frame of given concept.

The progress achieved during the last decades in CAD-tools is remarkable. It is easy to generate 2D and 3D drawings of the designed building, to analyze its response to external actions, to evaluate its construction costs and to plan the process of construction. On the other hand, there seem to be no computer-based counterpart of the sheet of paper on which the designer makes conceptual sketches. Moreover, there is an apparent temptation to skip the phase of conceptual design and to begin directly with composing

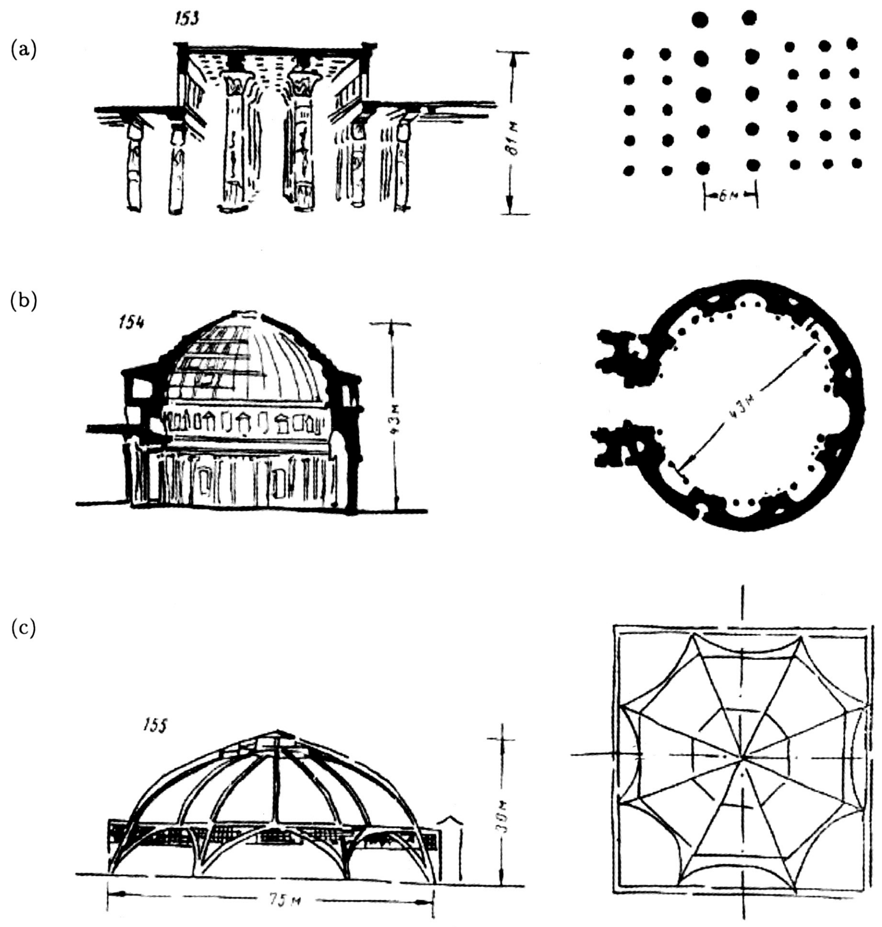


FIGURE 1. Conceptual changes in structures: (a) ancient Greek time; (b) Roman time; (c) 19th century.

the building of the routine library components. In that sense, the computerized design office has the tendency towards reproduction of known solutions instead of looking for novelty.

The necessity to stimulate creativity of the designer is obvious and many researchers look for ways to exploit the advantages of computer in that task (compare the proceedings of a conference devoted to the creative design [20]). Perhaps the most known software is the Invention Machine based upon the research done in the former Soviet Union by H. Altshuller [1]. Most of "invention machines" are based on the following pattern. First, the user is encouraged to formulate the problem that has to be solved. Then associations to the basic principles of mathematics and physics are revealed. If it is not possible to apply those principles directly, then the system presents solutions of similar problems obtained by other inventors. The strength of such systems lies in their case bases. Therefore, one could name this methodology the case-based creativity support. It is important that the cases are indexed by primary principles used to solve them. This allows the user to find solutions existing in domains being far from the area of his work. The reasoning through comparison and analogy is probably one of the most important ingredients of the human creativity.

Other possibility of looking for novelty is random or partly random search. Since computer can generate large number of trial solutions, there is a chance that one of them might prove to be useful as such or trigger the designer's imagination and let him develop previously unknown artifact. This way of reasoning is followed by creativity support systems that adopt the genetic algorithms. Perhaps pioneering work in this area was done by M.-L. Maher and P.X. Wu [32]. They looked for novel structural schemes by applying genetic operators for standard solutions. Similar results were obtained by G. Hliniak and B.Strug [27] in the domain of graphic design.

In this paper we want to confine the attention of reader to an interesting alternative in the computer-assisted creative design. Pioneered by N. Chomsky [13], the linguistic approach to world modeling found applications in many areas. The core idea in this methodology is to treat certain primitives as letters of an alphabet and to interpret more complex objects and assemblies as words or sentences of a language based upon the alphabet. Rules governing generation of words and sentences define a grammar of the concerned language. In terms of the world modeling such a grammar generates a class of objects that are considered plausible. Thus, grammars provide very natural knowledge representation formalism for computer-based tools that should aid the design.

2. Graph-based design knowledge representation

2.1. Graphs and graph transformations

Since G. Stiny [39] has developed the shape grammars, many researchers showed how such grammars allow the architect to capture essential features of a certain style of the building (e.g. a Victorian house or a Roman villa). However, the primitives of shape grammars are purely geometrical which restricts their descriptive power. Substantial progress was achieved after the graph grammars were introduced and developed (compare, e.g. [41]). Graphs are capable to bear much more information than linear strings or shapes. Hence, their applicability for CAD-systems was immediately appreciated [21].

A special form of graph-based representation has been developed by E. Grabska [22, 23]. This formalism distinguishes the composition graphs (CP-graphs) that describe the structure of the object from the realization schemes describing the visualization. In 1996-98 Grabska's model served as the basic knowledge representation scheme in the research project [10] aimed at developing intelligent design-assisting tools for engineering. The results of that project were reported at the conferences in Stanford [24], Ascona [7] and Wierzba [8].

It turned out that by introducing an additional functionality graph into the original Grabska's 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 (compare, e.g. [14]). Such methodology allows the designer to distract himself from details and to consider the functionality of the designed object, the constraints and the requirements to be met and the possible ways of selecting optimum alternatives.

2.2. Composite representation

This model of knowledge representation consists of three main ingredients: the composition graph (CP-graph), the realization scheme and the control diagram. Figure 2 shows an example of the CP-graph: the objects a and b are linked by an edge corresponding to certain relation. The left end of this edge is attached to an *out-bond* of the node a. The opposite end enters an *in-bond* of the node b. Distinguishing in- and out-bonds is useful in transformations that convert one graph into another.

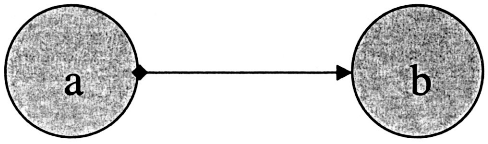


FIGURE 2. Simple CP-graph.

The mostly used transformation is a production or graph rewrite rule:

$$p : L \to R \tag{1}$$

Here p is the production label, L stands for the left-hand side of the rule and R is its right-hand side. The operator $\to$ indicates that the graph L is transformed into the graph R.

CP-graphs are used for storing structural information about objects. Data related to geometry and visualization are described by realization schemes. In particular, such schemes can contain predicates that constrain certain features of the described objects. Libraries of primitive objects are also attached to realization schemes.

Finally, all information related to the generation of objects is stored in control diagrams. These diagrams describe the sequence in which particular productions are applied. The result of applying a control diagram is generating a class of objects. In the context of design these are the objects plausible from the point of view of the designer. Further details on the composite representation can be found in [22].

2.3. Building functionality graph

The experience gained in the project [10] showed that the composition graphs should be further subdivided into the functionality graphs and the structural graphs. The nodes of functionality graph correspond to specific functions that have to be carried out by the designed object. The edges of this graph describe the relations between those functions. The functionality graph describes an object at the highest level of abstraction: no physical components are attached to this graph. Such components correspond to the nodes of structural graph. The edges of the latter describe relations between components. Thus, the structural graph reflects sub structuring of the designed artifact.

It is a common opinion among designers that the analysis of functionality should precede the sub structuring. Such a methodology allows the designer to consider carefully the most general aspects of the usage of the object prior to starting the detailed design. The prototype software developed at the Institute of Computer Science of the Jagiellonian University in Cracow contains a graphical editor for the purpose of functionality analysis. Figure 3 shows the copy of its screen.

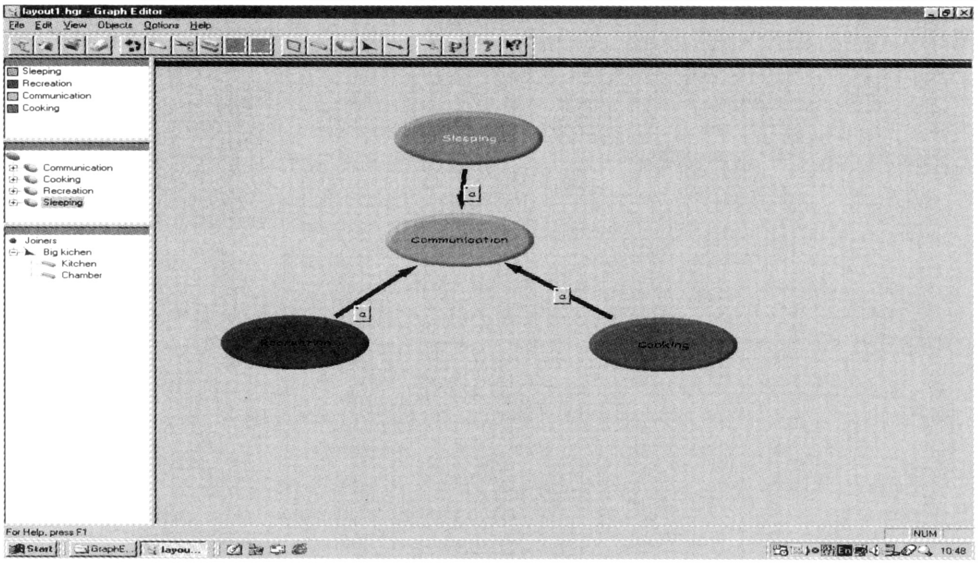


FIGURE 3. Editor of functionality graphs.

The list of functions selected by the user is shown on the left panel. These turn out to be the typical functions of a single-family house. The main panel contains the functionality graph. The functions listed at the left are displayed in different colors. After the user attaches a particular function to the node, this node is displayed in the proper color. The tools displayed in the menu bar allow the user to create new nodes and edges, to delete them, to split a node into two nodes or to merge two nodes into one. Each operation is mirrored to the symbolic description of the graph that is maintained in the background.

The editor separates the designer from the technicalities of the model. He or she can concentrate attention on the substantial features of the designed object, namely, on its functions and the relations between them.

2.4. Building structural graph

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 artefact. 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 before the satisfactory solution is found (Fig. 4).

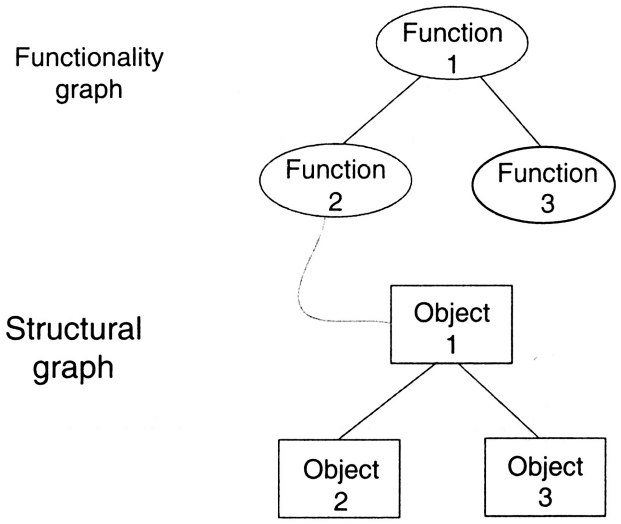


FIGURE 4. Editor of functionality graphs.

3. Floor layout design[1)]

Let us clarify the idea of our approach on an example of designing a single-family house. The functional requirements for such a house are straightforward. Given the number of inhabitants, consisting of adults, children and guests, one has to arrange the space in the house so that they can sleep, prepare and consume meals, meet each other and rest conveniently. It is natural, therefore, to consider the house as an assembly of the sleeping area, the social area, the communication area, the guest area, etc. The size of each area depends upon the number of relevant inhabitants. The total area of the house is given as an a priori constraint.

The user of our systems begins with defining functional requirements. This is accomplished by means of a graphical editor. Figure 5 shows its window. 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 edit the edges representing functional relations.

In the second step the structural graph is generated. The nodes of this graph represent the rooms of the house; the edges between the nodes represent the accessibility relations between the rooms. After the structural graph has been generated, the designer evaluates it. Note that at this stage the level of abstraction remains high: the

[1)]This Section is based upon results obtained by J. Szuba in the course of preparing his Ph.D.-thesis.

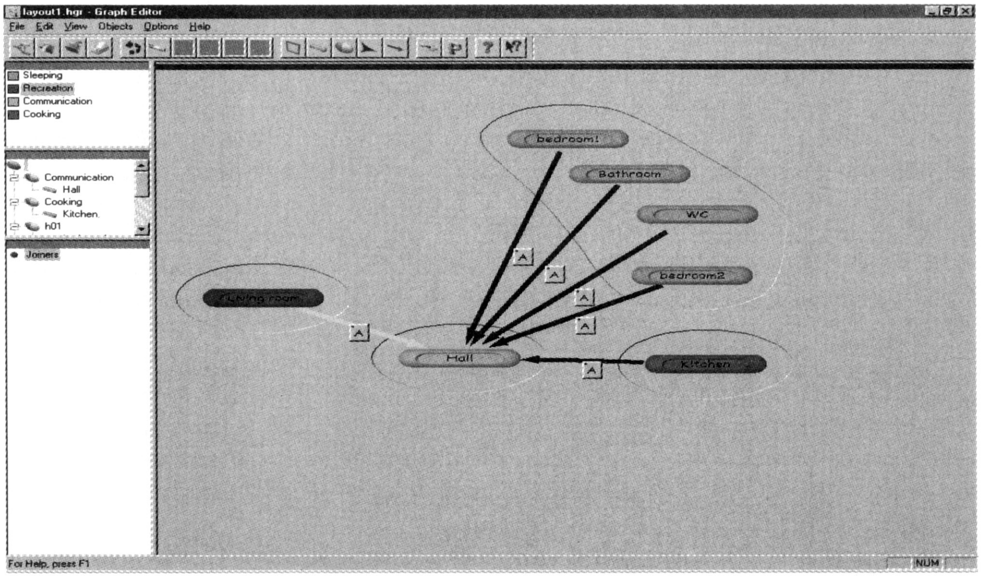


FIGURE 5. Editor of functional decomposition.

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. Adding (deleting) the node is for him adding (deleting) the room in the house, adding (deleting) the edge is adding (deleting) the passage from one room to another, merging two nodes is merging two rooms into one (e.g., joining bathroom and WC), splitting two nodes is making two rooms out of one (e.g., dividing the hall into two parts), changing the node type is changing the function of the room (e.g., changing a dining room into a sleeping room), changing the label of edge is changing the relation between rooms. Architects are trained in visual reasoning. Therefore, they find this tool quite intuitive.

In the sequel we present results obtained in co-operation with E. Grabska, M. Nagl and A. Schürr under a joint research project [11] The aim of this project is to develop prototype software that will assist an architect in the design of the layout of building. Contrary to conventional expert systems proposed previously, like [18], our system can be seen as a conceptual pre-processor for an architecture-oriented CAD-tool. It allows the user to specify functional requirements for a single-family house in terms of graphs, generates a proper graph grammar and translates the result into the input file for the CAD-system ArchiCAD [2]. The architect obtains a draft layout of the house that can be visualized and presented to the investor.

Fast prototyping is important in many areas and architecture is no exception. It enables the designer to present the draft of the design to the client in short time and to achieve approval or disapproval of the presented proposition. In this way the designer knows the intention of the client. If the client accepts the general conception of the design, the architect can start working on details.

Our system is generative: it does not produce a single layout but an entire family of plausible layouts described by the graph grammar. In order to achieve that we apply the Unified Modelling Language (UML) [5] for the specification purposes and we take advantage of the FUJABA [19] – a convenient Java-based graph editor. It is our intention to replace in the future FUJABA by the more powerful generative system PROGRESS [37] developed at the RWTH, Aachen. The results described in this Section can be viewed as the further development of the research reported previously in [25] and [40].

FUJABA contains graph grammar language called story diagrams. This language uses the UML class diagrams for specifying graph schemes. The UML activity diagrams serve for the representation of control structures and the UML collaboration diagrams provide the notation for graph rewriting rules. Story diagrams are internally translated into Java classes and methods allowing seamless integration of the object-oriented and graph-grammar-specific parts of the system.

People knowing the UML easily understand story diagrams. Class diagrams that allow the user to define attributes, methods and relations between classes represent graphs. Methods are implemented be means of story diagrams. Each story diagram may have formal parameters for passing attribute values and object references. Story diagrams follow the UML-notation of activity diagrams to represent graphically the control flow. Thus, the basic structure of a story diagram consists of a number of activities shown by rectangles with rounded left and right sides. Activities are connected by transitions that specify the execution sequence. The execution of the story diagram starts at the unique start activity represented by a filled circle. It proceeds by following the outgoing transition(s). Multiple outgoing transitions are guarded by mutually exclusive Boolean expressions shown in square brackets. Diamond-shaped activities express branching.

The first step in describing the specification of an object is constructing a class diagram where the objects that will be transformed and the relations between them are defined. The class diagram relevant for the specification of the single-family house is shown in Fig. 6. The most important class in this specification is the class *House* that represents the entire building. The class *Area* is the base class for the subclasses *SleepingArea*, *CommunicationArea*, *RelaxationArea*, *EatingArea*, *GuestArea* , *CleaningArea*. These subclasses bear the main functions of the house. The class *Room* is the base class for subclasses *BedRoom*, *BathRoom*, *WCBathRoom*, *WC*, *Kitchen*, *DinningRoom*, *Hall*, *LivingRoom* that represent physical decomposition of the building. Finally, the class *User* is the base class for subclasses *Adult*, *Child*, *Guest* that represent types of inhabitants.

The following relations are defined between classes:

1. *HouseContainsArea* – between *House* and *Area*,
2. *AreaContainsRoom* – between *Area* and *Room*,
3. *Uses* between *User* and *Room*,
4. *AreaAccessibility* between *Area_ 1* and *Area_ 2*,
5. *RoomAccessibility* between *Room_ 1* and *Room_ 2*

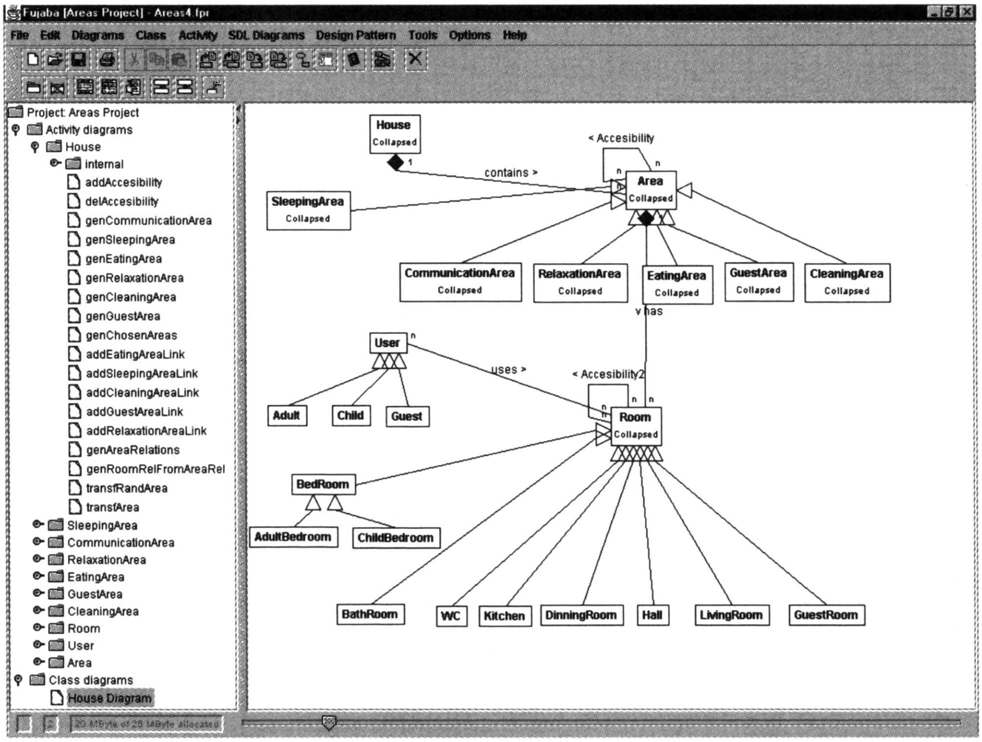


FIGURE 6. Class diagram of single-family house in FUJABA.

Until now we deliberately stripped our objects off the geometry: they were dimensionless and the only thing that matters were their mutual functional relations. After this phase of the design process is accomplished, one needs to proceed with the second phase: the generation of the drawing of designed object. In our case this means the drawing of the floor plan of the house.

The layout generator should take into consideration important requirements that were not covered by the functional graph and the structural graph. Some of them, taken from the guides on design in architecture [31] and [29], are listed below:

1. The outer contour of the house may be influenced either by the shape and dimension of the available piece of land or by the preferences of the client;
2. The orientation of the house with respect to the north-south axis must be taken into account when placing the rooms best location of the room in the house with regard to world directions (e.g., a living room should be located in the south, whereas a kitchen in the north);
3. There are minimal values of areas for particular types of rooms (e.g., the area of living room should not be less than $18\,\mathrm{m}^2$);
4. Most rooms should be kept as close as possible to square shape (exceptions are communication areas and special purpose units);

5. Main dimensions of the building, like the distances between load carrying walls, the height of the floors, etc., are standardized and must follow certain modular system;
6. Components like doors or windows are taken from the library and also have modular dimensions.

The present prototype of floor layout generator works under the assumption of "rectangular world": the outer contour of the building is a rectangle and the house consists of rectangular rooms. This assumption simplifies the strategy of composing the floor layout. It will be abandoned in the next version of the program. The advanced version of the floor generator will include also the positioning of the house over the land piece and its orientation with respect to the north-south direction. At present this orientation and the choice of the position of main entrance are done manually.

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. The

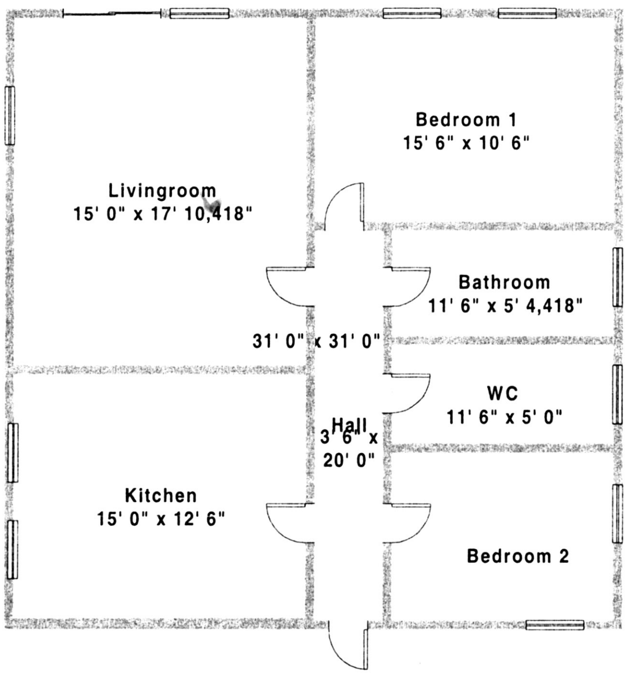


FIGURE 7. Generated layout of house.

rooms in small houses are placed around a corridor leading from the main entrance. For medium size houses this corridor is replaced by a hall and for big houses – by an internal garden or atrium.

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. Finally the windows are placed in the outer walls. Their size and position follows from the illumination requirements for rooms of the specific size and function.

Figure 7 shows the floor layout obtained by means of the prototype generator. Usually the floor layout generated automatically serves only as a raw material for further improvement. At present the architect itself within the ArchiCAD must do all changes. Our aim is to equip the next version of the system with a possibility of transferring changes back to the level of graph-based functional-structural description. This will significantly improve the flexibility and the assisting power of the tool.

4. Structural optimization

4.1. Formulation of problem

The problem of structural optimization is usually formulated as follows: given a set of design variables and a set of constraints find a structure that is optimal in certain sense and simultaneously feasible. The condition of optimality is fulfilled if a given function of the design variables attains its extreme (scalar optimization) or if a compromise between several conflicting design goals is found (vector optimization). The feasibility condition requires the design variables to satisfy all constraints imposed of the structure.

Depending upon the type of design variables two alternative approaches to the considered problem can be distinguished. In the smooth or continuum approach (Fig. 8a) one looks for an optimum shape and, possibly, for optimum distribution of internal properties of the body. In this case the design variables are smooth functions of the coordinates, the constraints may include differential equations and the design goal is usually expressed in the form of integral over the body. Finding a function that brings an integral over a given domain to its extreme is the well known problem of the variation calculus. Unfortunately, few practically important problems can be solved this way.

An alternative is to consider the structure being composed of a finite number of components. The truss shown in Fig. 8b is a natural example: it consists of 6 nodes

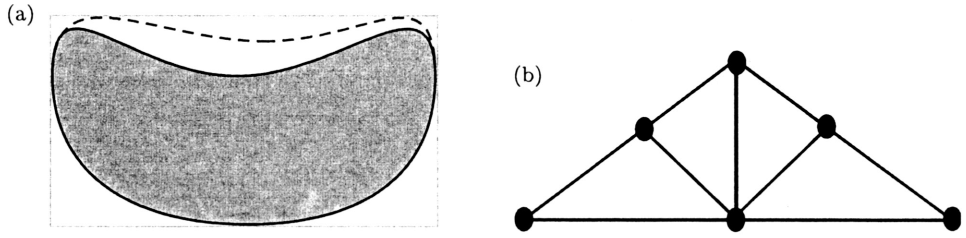


Figure 8. Alternative approaches in optimum structural design: (a) smooth; (b) discrete.

connected by 9 bars. The design variables are this time pure numbers and the design goals can be expressed in terms of functions as opposed to functionals present in the smooth formulation.

Simplicity of the discrete case is misleading: discrete problems are computationally expensive due to their combinatorial nature. The full formulation of such problems should cover the following levels of optimization:

1. the layout[2)] level,
2. the geometry level,
3. the member level.

At the layout level one has to decide how many components are to be used and how these components are to be interconnected. The natural formalism for expressing structural layouts is the graph theory. It allows the user to construct, evaluate and manipulate graphs that bear all necessary information.

At the geometry level we assume the layout to be fixed and we consider the best positioning of individual components. In the case of truss depicted in Fig. 8b this means looking for the optimum positions of the nodes. Certain constraints may appear at this level: usually the supports have given a priori positions, sometimes the contour of the structure must conform to certain pattern, etc.

The lowest level is related to the properties of individual members. Given the layout and the geometry, one may still look for the best sizes of cross sections or for the most suitable distribution of material properties (the Young module, the yield stress, etc.).

If one considers only the levels 2 and 3, i.e. if one optimizes the geometry and the structural components under the given layout, then the problem falls into the class of mathematical programming problems [6]:

$$\min\{f(x_k) \mid g_l(x_k, y_j) \leqslant 0\} \tag{2}$$

and can be solved by means of a suitable numerical method [36]. Here the design goal is to minimize the function $f(x_k)$called the cost function, x_k stand for the design variables and the constraints g_l include not only the design variables but the state variables y_j as well. This type of structural optimum design has very rich literature and can regarded as more or less exhausted as a topic of research.

4.2. Previous attempts on layout optimization

Contrary to that, it is still an open question how to find the best structural layout. Many researchers devoted their effort to it (compare, e.g., the reviews [30] and [42]) but the results are still far from being satisfactory. Perhaps the best method for today, called adaptive optimization [3], was inspired by the living nature. A bone is the main load carrying component of the human body. It develops its sophisticated and highly efficient internal structure (Fig. 9) during the phase of growth. The external surface remains smooth and resembles a tube whereas the internal skeleton follows the lines of principal stresses. Notably, this process is reversible: the astronauts suffer from their bones becoming weaker after a long stay in the space. At the absence of loading the unnecessary material is removed from the bone.

[2)]Many authors substitute "topology" for the layout. This should not be done since mathematicians already use the term "topology" for different purpose.

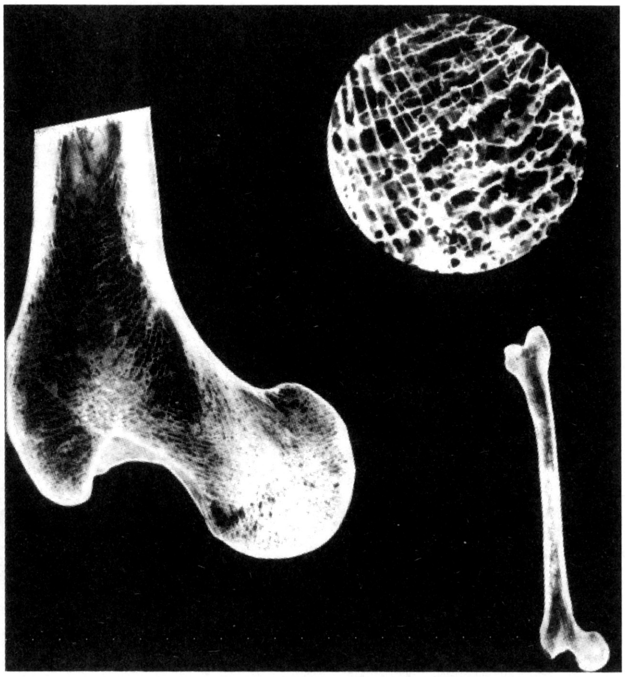


FIGURE 9. Structure of bone.

Figure 10 taken from the paper by K.-U. Bletzinger, S. Kimmich and E. Ramm [4] shows typical result of the adaptive optimization. The solid block visible at the left side was taken as an initial guess for the cantilever loaded by a single vertical force. The hollow shape at the right is the final result. Notably, this result requires heuristic post processing since the adaptive process itself ends up with a discrete truss-like structure, as can be seen in the lower part of the figure.

The adaptive method circumvents the difficulty in solving directly the layout optimization problem. It starts with a homogeneous solid of an arbitrary simple shape and relies upon the users ability to extract optimum topology from the truss-like skeleton. Another interesting indirect method has been proposed by D.M. Stal and G.M. Turkiyyah [38]. This method introduces the notion of the structural backbone that is allowed to develop in the gradual optimization process.

Probably the first direct method of the layout optimization was the ground structure concept developed in early 1960's by W. S. Dorn, R. E. Gomory and H. J. Greenberg [16]. It was applicable only for trusses and excluded optimization of nodes. Figure 11 shows the ground structure for the rectangular truss. The positions of all nodes are fixed and all possible connections between them are taken into account. Such ground structure serves as an initial solution for the numerical optimizer – the simplex method of linear programming – with the cross sectional areas of the bars as the design variables. The optimum solution includes bars with cross sectional areas close to zero. Removing them one obtains the optimum layout. Though simple in principle, this method proved to be numerically unstable and is not used today.

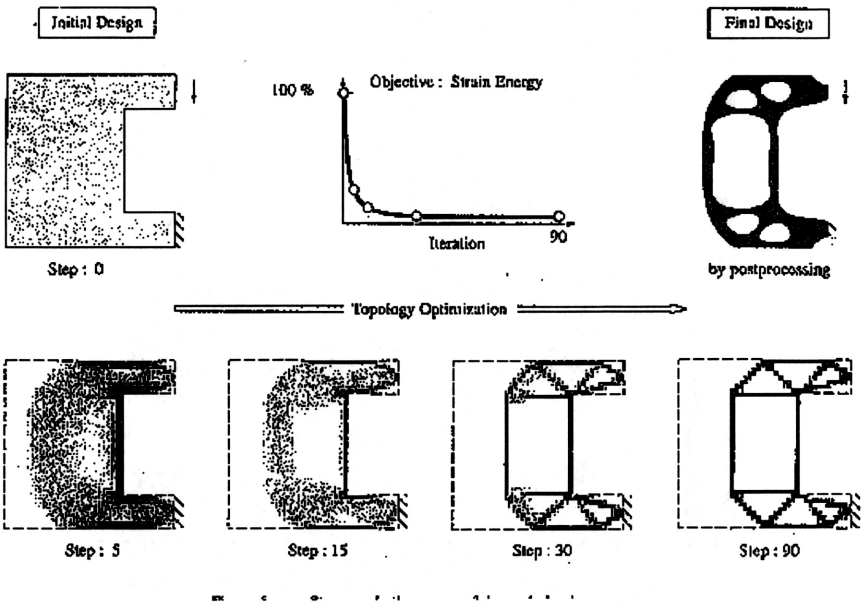


FIGURE 10. Example of adaptive optimization.

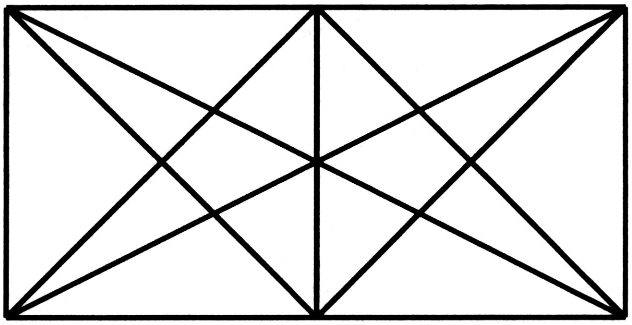


FIGURE 11. Example of adaptive optimization.

A novel concept of topological derivative was proposed recently by Z. Mróz and D. Bojczuk [35]. They define a set of basic transformations like introducing a new node (Fig. 12a) or splitting an existing node (Fig. 12b) and consider the influence of such transformation upon the cost function. The derivative with respect to the design variable s_j gives the user information about the slope of the cost surface and allows him to look for the optimum by means of the gradient-based search.

Mróz and Bojczuk adopted gradual growth of the structural length as the main strategy. They were able to show that at certain characteristic values of the length

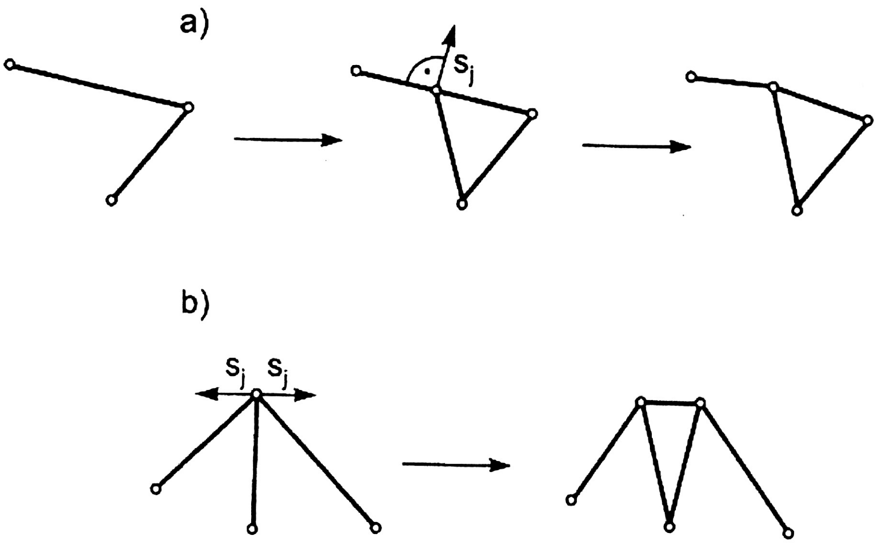


Figure 12. Concept of topological derivative.

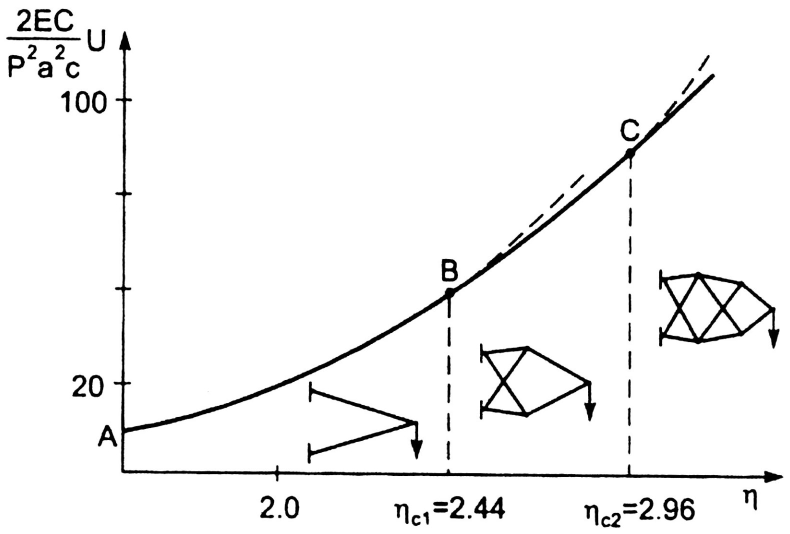


Figure 13. Topological bifurcations.

the optimum layout changes and called this phenomenon the topological bifurcation (Fig. 13). Changes in the layout may introduce discontinuities in the cost function. Therefore the topological derivative must be understood in the generalized sense.

The Mróz-Bojczuk method leads to interesting and practically meaningful results. Figure 14 shows the evolution of cantilever truss optimum in the sense of minimum compliance. The buckling constraints were neglected in this case. After introducing them and changing the design goal to the minimum weight, one obtains substantially different evolution depicted in Fig. 15.

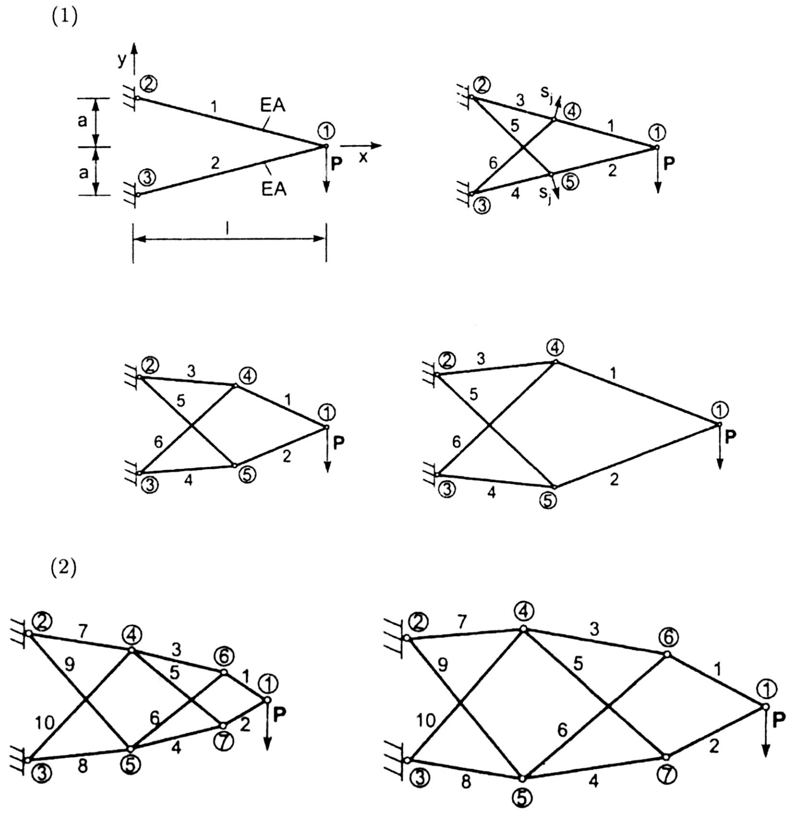


FIGURE 14. Evolution of truss optimal in sense of minimum compliance: (1) initial stage; (2) continuation.

It is well known that the problem of layout optimization for a minimum compliance has been solved analytically by A.G.M. Michell [34] already in 1904. His solution disregards buckling and consists of an infinite number of logarithmic spirals that split into two buckets originating at the supports. The discretized form of this solution is shown in Fig. 16. The solution obtained by Mróz and Bojczuk differs from the Michell's truss. The probable cause of this discrepancy is the non-differentiability of the cost function in the layout optimization problem. The generalized derivative introduced in the Mróz-Bojczuk method may lead to sub optimal solutions.

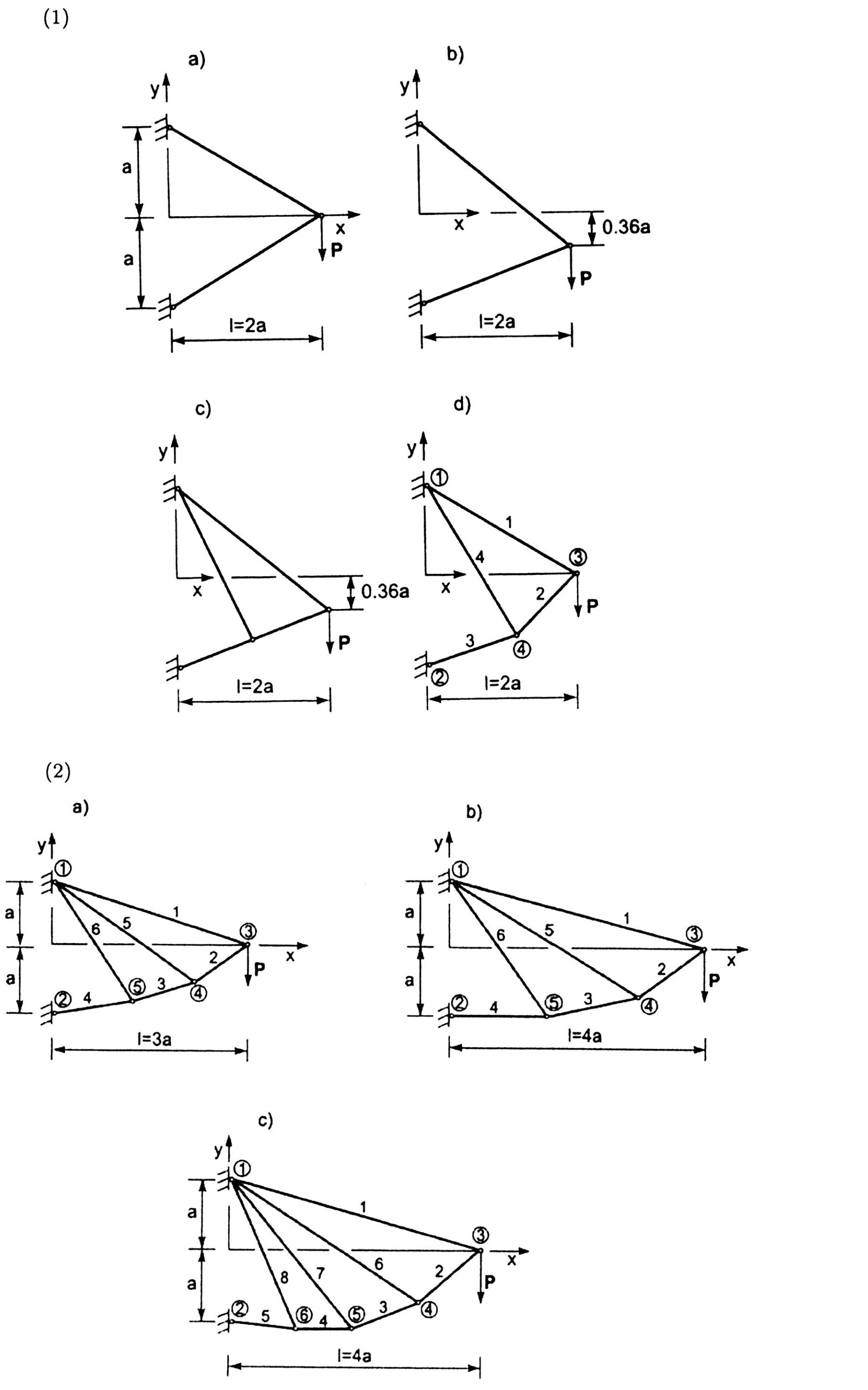


FIGURE 15. Evolution of truss optimal in sense of minimum weight: (1) initial stage; (2) continuation.

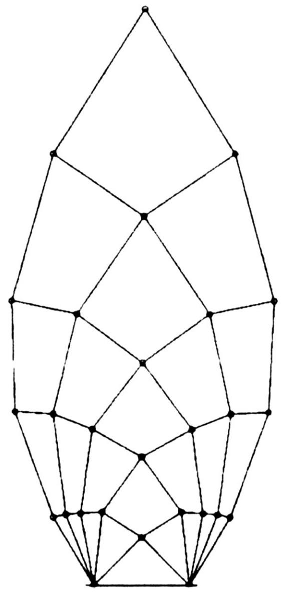


FIGURE 16. Michell's truss.

4.3. Graph-based approach combined with genetic search

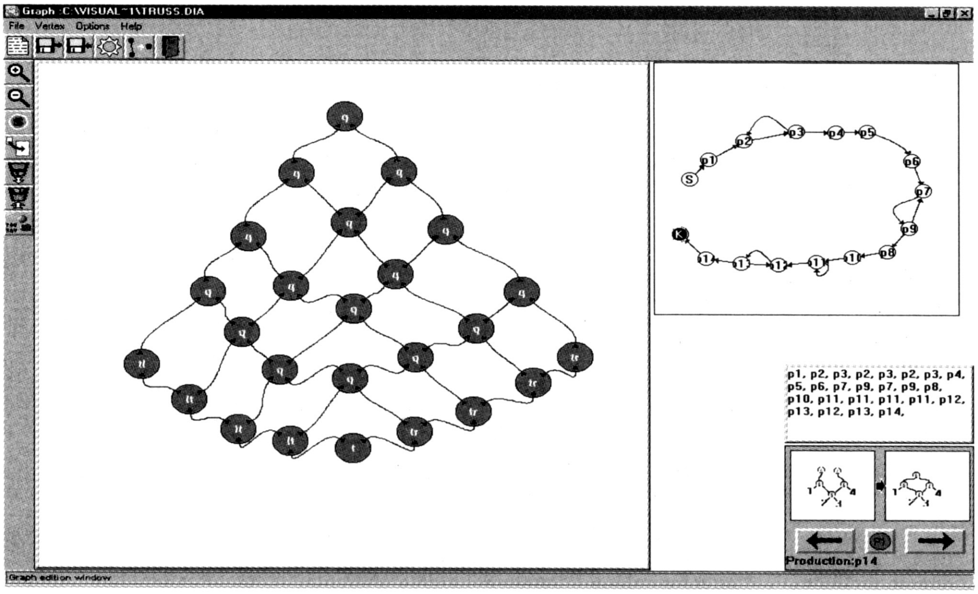
Since the optimization of layout leads to discontinuities of the cost function, natural candidates for numerical optimizers are methods that do not rely on the gradient of that function. A broad class of such methods is available in the theory of optimization, most of them including the component of random search in the space of solutions. The genetic algorithms (compare, e.g. the excellent monograph by Z. Michalewicz [33]) that became popular recently belong to this class.

The concept of genetic algorithm has been borrowed from the Darwin's evolution theory. In the classical form proposed by J.H. Holland [28] the properties of an optimized object are coded in the form of binary string called chromosome. Generating chromosomes at random forms an initial population of objects. The iterative process of evolution consists of mating objects taken from the current population (the parents), producing their off springs and allowing the best of them to form the next population. The objects are evaluated by the fitness function that resembles the cost function in conventional optimization.

Nowadays the theory of genetic search is quite elaborate: many ways were proposed regarding coding conventions for chromosomes, genetic operations that produce off springs, selection strategies, etc. Taking advantage of those proposals several authors applied the genetic search to the structural optimization (compare, e.g., the papers by T. Burczyński [12], K. De Jong and T. Arciszewski [15], H. Eschenauer and A. Schumacher [17] or P. Hajela and J. Lee [26]).

Despite the difference in the optimization tool, most of those papers followed the methodology of adaptive optimization proposed by Bendsoe and Kikuchi. This means that the layout was neither explicitly represented nor explicitly optimized. For example, the bubble method proposed in [17] starts with a random number of cavities (bubbles) introduced into a homogeneous solid and let those cavities evolve into the final layout.

The research project started recently by the author of present paper in the collaboration with E. Grabska aims at exploiting the advantages of graphs as the language for representing structural layouts. Figure 17 shows the graph grammar for Michell's truss. Similarly as it was done with the floor layouts, one can build a graph-based generative system that will produce plausible initial populations of structural layouts. Applying genetic evolution to such populations one may expect interesting optimal or sub optimal layouts to emerge. The first attempts made in the domain of industrial design [27] indicate that this approach may be quite successful.



```
p1, p2, p3, p2, p3, p2, p3, p4,
p5, p6, p7, p9, p7, p9, p8,
p10, p11, p11, p11, p11, p12,
p13, p12, p13, p14,
```

Production:p14

FIGURE 17. Michell's truss.

5. Conclusions

In this article we have concentrated on the passage from the functional requirements of a designed object to the object structure. We restricted our consideration to a rather simple example of designing a house since the methodology remains valid for any object. In our example it is easy to distinguish various kinds of areas, rooms and relations between them and to show that graphs and graph transformations are useful as knowledge representation in computer aided design. The presented methodology seems to be appropriate for architects because they often use graphs – sometimes being unaware of it.

The second part of the paper is devoted to the difficult problem of structural layout optimization. Having compared different indirect and direct methods proposed so far, we argue that the graph-oriented representation combined with the genetic search may lead to efficient methodology. At present this approach is at the preliminary phase and much effort is still needed before reaching the maturity for practical applications.

Acknowledgement

Partial financial support of the Alexander von Humboldt Foundation under the joint Polish-German research project "Graph-based tools for conceptual design in Civil Engineering" is gratefully acknowledged.

References

1. H. Altshuller, *Algorithm of Invention* (in Russian), Moskovskij Rabochij Publishing House, Moscow 1969.
2. *ArchiCAD 6.5 Reference guide*, Graphisoft, Budapest 2000.
3. M.P. Bendsoe, *Optimization of Structural Topology, Shape and Material*, Springer, Berlin 1990.
4. K.-U Bletzinger, S. Kimmich and E. Ramm, Efficient modeling in shape optimal design. in: B.W.E. Topping (Ed.), *Computational Structures Technology*, Herriot-Watt University, pp.1-15, Edinburgh 1991.
5. G. Booch, J. Rumbaugh and I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley Longman Reading, 1999.
6. A. Borkowski and St. Jendo, *Mathematical Programming*, Vol.2 of the Series "Structural Optimization" ed. by M. Save and W. Prager, Plenum Press, New York 1990.
7. A. Borkowski and E. Grabska, Converting function into object, in: I. Smith (Ed.), *Proc. 5th EG-SEA-AI Workshop on Structural Engineering Applications of Artificial Intelligence*, LNCS 1454, Springer-Verlag, pp.434-439, Berlin 1998.
8. A. Borkowski (Ed.), *Artificial Intelligence in Structural Engineering, Proc. 6th EG-SEA-AI Workshop*, Wierzba June 1999, WNT, Warszawa 1999.
9. A. Borkowski, E. Grabska and G. Hliniak, Function-Structure Computer-Aided Design Model, Machine GRAPHICS and VISION, Vol.9, pp.367-383, 1999.
10. A. Borkowski, E. Grabska and J. Pokojski, *Computer-Aided Innovative Design* (in Polish), Final Report of the Research Project No. 8T11F02611 financed by the Polish Committee of Research, IPPT PAN, Warszawa 1999.
11. A. Borkowski, E. Grabska, M. Nagl and A. Schürr, *Graph-based tools for conceptual design in Civil Engineering*, Joint research project financed by the BMBF, Warsaw-Cracow-Aachen-Munich 2000-2002.
12. T. Burczyński, Evolutionary algorithms: applications in mechanics (in Polish), in: T. Burczyński and W. Cholewa (Eds.), *Proceedings of Symposium on Methods of Artificial Intelligence in Mechanics and Mechanical Engineering (AI-MECH 2000)*, Gliwice 2000.
13. N. Chomsky, *Aspects of Theory of Syntax*, MIT Press, Cambridge 1965.
14. E.L. Cole Jr., *Functional analysis: a system conceptual design tool*, IEEE Trans. on Aerospace & Electronic Systems, 34 (2), pp.354-365, 1998.
15. K. DeJong and T. Arciszewski, An overview of evolutionary computation and its application to engineering design, in: A. Borkowski (Ed.), *Artificial Intelligence in Structural Engineering*, Proc. of the 6^{th} EG-SEA-AI Workshop, Wierzba June 1999, pp.9-22, WNT, Warszawa 1999.
16. W.S. Dorn, R.E. Gomory and H.J. Greenberg, Automatic design of optimal structures. *Journal de Mechanique*, 3, pp.25-52, 1964.

17. H.A. Eschenauer and A. Schumacher, Bubble method for topology and shape optimization of structures, *Structural Optimization*, 8, pp.42-51, 1994.

18. U. Flemming, R. Coyone, T. Gavin and M. Rychter, A generative expert system for the design of building layouts – version 2, Artificial Intelligence in Engineering: Design, in: B.H.V. Topping (Ed.), *Computational Mechanics Publications*, pp.445-464, Southampton 1995.

19. T. Fischer, J. Niere, L. Torunski and A. Zündorf, Story diagrams: a new graph rewriting language based on the Unified Modeling Language and Java, in: *Proceedings of TAGT'98 (Theory and Application of Graph Transformations)*, LNCS , Springer-Verlag, Berlin 1999.

20. J. Gero, M.-L. Maher and F. Sudweeks (Eds.), *Preprints of Computational Models of Creative design*, Key Center of Design Computing, Melbourne, 1995.

21. H. Göttler, J. Günther and G. Nieskens, Use graph grammars to design CAD-systems!, in: G. Rozenberg (Ed.), *4th International Workshop on Graph Grammars and Their Applications to Computer Science*, LNCS 532, pp.396-410, Springer-Verlag, Berlin 1991.

22. E. Grabska, *Theoretical concepts of graphical modelling. Part one: Realization of CP-graphs*, Machine GRAPHICS and VISION, 2(1), pp.3-38, 1993; *Part two: CP-graph grammars and languages*, Machine GRAPHICS and VISION, 2(2), pp.149-178, 1993.

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

24. E. Grabska and A. Borkowski, Assisting creativity by composite representation, in: J.S. Gero and F. Sudweeks (Eds.), *Artificial Intelligence in Design'96*, pp.743–760, Kluwer Academic Publishers, Dordrecht 1996.

25. E. Grabska and A. Borkowski, Generating floor layouts by means of composite representation, in: *Proc. Worldwide ECCE Symp. on Computers in the Practice of Building and Civil Engineering*, pp.154-158, 1997.

26. P. Hajela and J. Lee, Genetic algorithms in truss topological optimization, *Journal of Solids and Structures*, Vol.32, No.22, pp.3341-3357, 1995.

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

28. J.H. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor 1975.

29. E. Neufert, *Bauentwurfslehre*, Vieweg & Sohn, Braunschweig-Wiesbaden 1992.

30. U. Kirsch, Optimal topologies of structures, *Applied Mechanics Reviews*, 42, pp.223–239, 1989.

31. W. Korzeniewski, *Appartement Housing. Architect's Guide* (in Polish), Arkady, Warszawa 1989.

32. M.-L. Maher and P.X. Wu, Creativity through co-evolutionary design, in: J. Gero, M.-L. Maher and F. Sudweeks (Eds.), *Preprints of Computational Models of Creative Design*, Key Center of Design Computing, Melbourne, pp.244-259, 1998.

33. Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, New York 1994.

34. A.G.M. Michell, *The limits of economy of material in frame structures*, Philosophical Magazine, Ser.6, 8, 1904.

35. Z. Mróz and D. Bojczuk, Topological derivative concept in optimal design of structures, in: *Proc. 2nd World Congress of Structural and Multidisciplinary Optimization*, Amherst, May 1999.

36. K. Schittkowski, Software for mathematical programming, in: K. Schittkowski (Ed.), *Computational Mathematical Programming*, NATO Advanced Study Institute Series, Vol.F15, Springer-Verlag, Berlin 1985.

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

38. D.M. Stal and G.M. Turkiyyah, Skeleton-based techniques for the creative synthesis of structural shapes, in: J.S. Gero and F. Sudweeks (Eds.), *Artificial Intelligence in Design '96*, Kluwer Academic Publishers, pp.761-780, Dordrecht 1996.

39. G. Stiny, *Introduction to Shape and Shape Grammars, Environment and Planning B: Planning and Design*, Vol.7, p.343-351, 1980.

40. J. Szuba, E. Grabska and A. Borkowski, Graph visualization in ArchiCAD, in: M. Nagl, A. Schürr and M. Münch (Eds.), *Application of Graph Transformations with Industrial Relevance*, LNCS 1779, Springer-Verlag, pp.241-246, 2000.

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

42. G.I.N. Rozvany, M.P. Bendsoe and U. Kirsch, Layout optimization of structures, *Applied Mechanics Reviews*, 48, pp.41-119, 1995.