Key words:
automatic diagram drawing,
UML diagrams
Genetic Algorithms

Paweł HOFMAN*

Maciej PIASECKI*

AUTOMATIC IMPROVEMENT OF UML DIAGRAMS LAYOUT

In the paper an approach to automatic drawing of UML diagrams based on Genetic Algorithms is presented. The approach is a development of the ideas proposed in [[4]][[9]]. Our solution is oriented on the domain of UML diagrams, where the main emphasis is put on Class Diagrams and Use Case Diagrams. A set of aesthetic rules is introduced. The rules are used for the assessment of the layout of diagram elements. Some heuristics increasing the efficiency of the search for a solution are proposed. The heuristics allow for individualisation of the designing process of a diagram layout to the user needs. The results of an usability evaluation of the proposed approach are presented.

1. INTRODUCTION

One of the most tedious activities performed during UML [[1]] based modelling is continues correcting of the layout of diagram elements while introducing changes to a model. Corrections are necessary in order to preserve the clarity of the presentation.

The correction process seems to strongly depend on a direct involvement of the user, its ability to find the proper solution and its sense of aesthetics. The proper layout is important for information exchange in a design group and can influence the quality of a design. However, there are many attempts to automatic drawing of diagrams, e.g. [[3],[6],[7],[8],[11]], that are based on formally stated constraints and heuristics. Unfortunately, general solutions do not take into account the specific features of UML and the produced layouts are very unnatural for practitioners of UML based modelling.

* Institute of Applied Informatics, Wrocław University of Technology,
ul. Wybrzeże Wyspiańskiego 27, 55–370 Wrocław, Poland

General purpose approaches are based on the assumption, that a diagram is a *graph*. Elements of a diagram correspond to graph vertices, the diagram links to graph arcs. It is assumed very often that one distinguishes some *order* (direction) *of reading* for a graph, i.e. some levels of vertices can be defined according to the *paths* constructed from arcs. Moreover, a direction of traversing of a graph is often assumed, where there are a selected vertices of the begging and the end. According to these assumptions, a graph is drawn on a surface. Thus, a proper selection of basic assumptions and constraints is very important to the final shape of drawing for this class of approaches.

We propose a slightly modified approach. The general order of reading is not presumed. More emphasis is put on the set of aesthetic rules and their role is increased. Moreover, the user can play an active role by dynamically introducing some preferences *during layout* design. He has a chance to eliminate some nonsense solutions. The user is asked for selecting elements that will be subjects of optimisation. Taking this counter wise, the user has a chance to point out elements whose positions should not be changed. Some elements can be manually positioned and their positions are additional constraints to the algorithm.

UML diagrams consists of many different elements. For the sake of simplicity, we dived them into two broad classes of: *blocks* and *links*. Blocks are elements of two dimensions, are represented as rectangles, and are used for representing UML symbols of: *objects*, *classes*, *tables*, *states* etc. Links represent all *named* and *unnamed UML associations* between two blocks. In the case of UML *classes of association*, where we have a triple of UML elements connected, one link represents the association and the second one connects a block representing the class of association to the first link.

2. RULES OF AESTHETICS

For the general case of diagram drawing, there is no source of preferences for one layout before the others, except some obvious degenerated forms. It is due to the two basic facts: each user has his own preferences and different types of diagrams in different domains of applications are subjects of different conventions. For example, in the case of a hierarchical diagram, it is natural to expect a single element on the top as a source of down going arcs together with organising the rest of elements into subsequent layers. It is less common to find a single element at the bottom (however for some types of diagrams it is possible). However, such hierarchical organisation is contradictory with the rules of presenting UML Use Case Diagrams: even in the case of one *actor* it is always put on one side, while the Use Cases on the opposite side.

For example, for the four blocks presented in Fig. 1, there are some typical layouts expected for a UML diagram. The initial layout a) is incorrect, each next is acceptable as a final one.

Concentrating our attention on the two selected types of UML diagrams, namely Class Diagrams and Use Case Diagrams, we are going to formulate constraints expressing aesthetics preserved by designers. We want the constraints to express the decisions made by the user manually organising elements of a diagram. However, the rules should be easily implemented.

Generally, well organised diagram is such that the values of the following features are minimised (e.g. [[3]]):

- the number of crosses between two links,
- the number of crosses between a link and a block,
- the sum of lengths of all links,
- the area occupied by the diagram (correlated with the above).

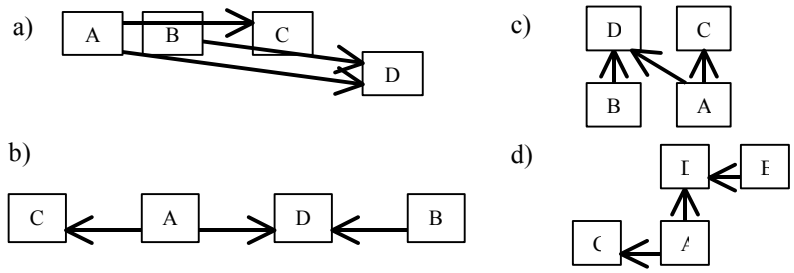


Fig. 1. Possible layouts, where a) is the initial layout

We propose to extend the set of the general constraints by the following ones:

- the number of the *segments* of a link should be minimal, where a segment is a part of link defined by the two subsequent changes in the direction of a link (the tendency to use straight lines),
- only one of the coordinates of the beginning and the end of a line should be different (once again straight lines are preferred),
- each link should have a length greater than some minimal length (the dimensions of a diagram should be too small).

The proposed set of constraints is assumed to produce solutions that are better matching the characteristics of the UML diagrams.

3. GENETIC ALGORITHM

One can notice that all the constraints are in close correlation. It would be hard to apply them separately. For examples, decreasing size of a diagram, and the same the lengths of links, would increase the number of crossings of different types, while the crossings seem to be more serious mistakes than the size of the diagram. Thus, the analysed problem is a problem of multi-criteria optimisation with the unknown variables dependencies [[3],[10]]. The situation is even more complicated, as it is the present layout of blocks and relations on the diagram which that is the source of dependencies.

The task is not trivial and, probably, there is no efficient deterministic algorithm for finding an optimal solution, i.e. an efficient algorithm from the point of view of the user. The user should be able to apply an option of automatic layout any number of times during modelling.

Existing approaches often use sets of heuristics, that are often not compatible with the current situation on the diagram, but make the time of computations being reasonable.

We propose a development of the Genetic Algorithm (GA) applied in [[3]]. Some *mutation operators* that are specific for UML diagrams are added. The mutation operators are mainly responsible for browsing the space of solutions. Some additional heuristics are introduced increasing the efficiency of finding the solution and making possible user's intervention.

Application of GA guarantees [[2]]:

- probabilistic choice, i.e. any layout can be produced from the same starting point,
- a set of solutions is evaluated simultaneously and parts of the currently best solutions are exchanged to produce the next ones,
- each time full solutions are modified what gives a chance to break the process at any time (and the solution which is the best one up till that time can be presented to the user),
- the currently best solutions are preserved (assuming that the fitness function is stable),
- and the process of finding layout can be restarted from any point e.g. after introduction of some changes by the user.

The formulated aesthetic constraints can be expressed in the form of the following fitness function:

$$f_{eval} = \sum_{i=1}^{k} \alpha_i x_i \tag{1}$$

where k is the number of the constraints, α_i — the weight assigned for breaking the i-th criterion, the weights define heuristically the `seriousness' of breaking the corresponding constraints, χ_i — the degree of breaking the i-th criterion.

The fitness function defines the quality of a solution from the point of view of all constraints in the same time. In that way, the optimisation of layout is transferred to the domain of the optimisation of the fitness function.

The optimisation process follows the scheme of a standard GA [[2],[5]]. The possible solutions — layouts — are encoded as specimens.

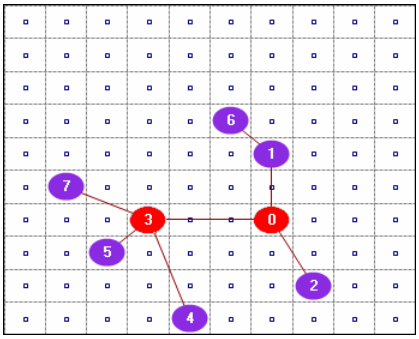


Fig. 2. Example of a specimen. The blocks: 0 and 3 (red) cannot be moved.

For the sake of better efficiency of the algorithm, the problem has been simplified by assuming that: blocks are cells in some grid and the links cannot be divided into segments. In that way, the two of the constraints are preserved in any generated solution.

A *specimen* contains a matrix of the size $N \times N$ (see Fig. 2), where N is chosen in such way, that the blocks do not occupy more than 10% of the cells. This encoding has been chosen after some initial tests and has been inspired by [[3]]. The information about links between blocks is also stored separately (as shared among specimens). But, additionally to locations of blocks, a specimen keeps three lists:

- of blocks, that can be moved (together with their coordinates in the matrix),
- of blocks that cannot be moved,
- and of all blocks.

The first two lists facilitates selection of blocks for GA operators, and the third facilitates calculation of the fitness function.

The fitness function takes in account three factors: the number of crossings between links, between links and blocks, and the size of the diagram. The probability of selection of specimens for the generation of the next population is directly related to their values of the fitness function.

Besides the set of offsprings generated by crossover and mutation, a new population includes also a set of the best specimens from the previous population. We introduced this to prevent the best solutions from being lost.

3.1 GA OPERATORS

Following [[3]], we introduced crossover and mutation operators dedicated to the domain. There are two crossover operators: *RectCrossover* and *TreeNodeCrossover*. *RectCrossover* chooses randomly some rectangle in the matrix. Next, each of the two offsprings takes some blocks from the interior of the rectangle of one of the parents and the rest of the blocks from the exterior of the second one. The operator tries to not change the positions of the blocks. Potential conflicts are solved by a random choice of a new position for a block.

TreeNodeCrossover chooses a connected subgraph of blocks of one of the parents and creates two descendants exchanging blocks according to the chosen subgraph. All potential conflicts are solved as before.

The initial set of mutation operators includes the following ones:

- *SingleMutate* — random choice of a block which is next moved to some randomly selected cell,
- *SmallMutate* — two rectangular areas are randomly chosen and the blocks inside them are exchanged (preserving their positions),
- *EdgeMutate* — a link is randomly chosen and all blocks attached to it are moved into some new positions,
- *EdgeMutate2* — similar to the previous one, except that during the movement of the block their relative positions are preserved.

After the preliminary experiments, we extended the set of mutation operators with some new ones resulting in the relaxed violation of aesthetic rules:

- *MiddleMoveMutation* — a block is moved into the middle of a group of blocks with which the moved block is in relation, see Fig. 3,
- *LinkMutation* — two directly linked blocks are set in the distance of one cell (pulling of the blocks closer),
- *Fit1DMutation* — making a chosen coordinate equal for the two randomly selected blocks,
- *Move1DMutation* — exchange of a chosen coordinate between the two randomly selected blocks.

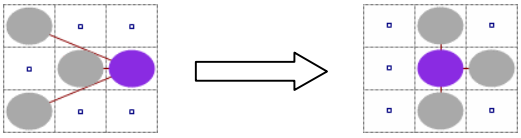


Fig. 4. Example of the application of *MiddleMoveMutation*.

3.2 SUPPORTING HEURISTICS

An improvement in the quality of solutions in comparison to [[3]] was achieved by application of some heuristics. We assumed that the user can have his own preferences and style concerning the layout. They must be considered while looking for the solution.

Firstly, an existing initial layout of elements is used in GA based optimisation. In the initial population, one of the specimens encodes directly the initial layout, in some others this layout is only slightly modified by mutation operators. In that way, the user can deliver to GA a kind of guidelines concerning possible arrangement of some groups of elements, e.g. requesting especially tricky movement of elements. The results of GA can only be improved by the initial layout, especially its speed depending on the number of populations. Any chaotic layout would be changed during subsequent populations.

Secondly, the user can exclude a subset of elements (a part of the diagram) from the work of GA, e.g. the user sets properly the positions of some elements. This information is encoded in specimens, and GA tries to optimise positions of the rest of the elements.

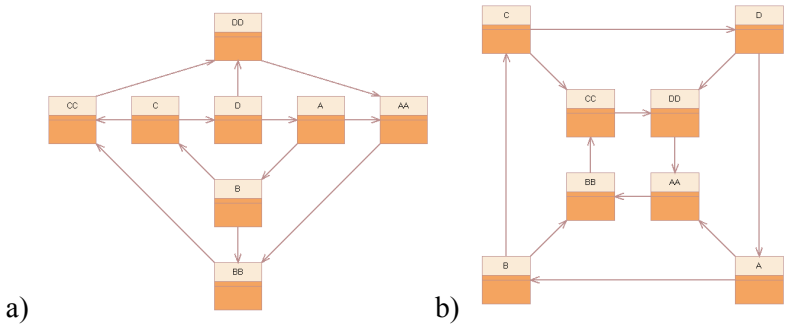
The *threshold heuristic* breaks the process of evolution if during the last k populations (k=900 was used in experiments) the best specimen has not been significantly improved. Such situations happens when the process got stuck in some local maximum and further evolution would not bring better result during many populations. It is better to present the current solution to the user, than to keep him waiting too long. The user can modify the diagram and eventually to continue the optimisation. Finally, a time limit (=5s.) on the time span of evolution was introduced. Also the *threshold heuristic* causes that in the case of quick achievement of good results the process takes less time.

As the scheme of encoding do not take into account links between a block and a link, UML classes of association are encoded internally in GA as linked to one of the participants of the association.

4. VERIFYING TEST OF USABILITY

Examples of layouts created by the algorithm are presented in Fig. 4. The real quality of the proposed solution has been tested on real users in usability tests. A group of several professional UML designers was asked to prepare some Use Case Diagrams and Class Diagrams for a simple project of an Internet communicator. Next the algorithm proposed a layout for each of the diagrams. The users loudly commented the achieved quality and their satisfaction. They generally accepted the algorithm as a tool which was saving their time during introduction of changes to the model. On average, automatically prepared layout needed only a few corrections.

In the second test, a list of 10 drawings were prepared including diagrams prepared manually and automatically. A different group of users were asked to identify the diagrams prepared automatically and to introduce improvements to them, if any are possible. On average, only 45% of guesses were correct.



a) b)

Fig. 4. Examples of layouts created by the algorithm.

The usability tests allowed for making several observations. A big disadvantage of the algorithm is that it cannot guess which element is the key element to the diagram and that it should be exposed. But the algorithm is very helpful in suggesting some solution, which is next improved by the user. The algorithm tends to semantically group the elements (according to relations) while minimising the number of crossing and the size of the diagram. The possibility of blocking allows the user to incrementally improve a diagram. The set of blocked elements is enlarged gradually. The specimen matrix causes alignment of elements along the grid, this has a positive visual effect. Links among associated elements are becoming shorter and groups more dense. The algorithm distinguishes very quickly some connected subgroups in the diagram. With all heuristics applied, the algorithm delivers a list of possible solutions (with the best as the first) in 4s on PC.

5. CONCLUSIONS

The proposed heuristics and mutation operators introduced a significant improvement to the initial algorithm similar to [[3]]. However, regardless the positive opinions of the user, the issue of the automatic layout of UML diagrams is still open. The proposed solution has been optimised mainly for Class Diagrams and Use Case Diagrams, probably for many other classes of diagrams, the results would be probably less satisfactory. Anyway, our work shows, that a quality perceived by the users can be achieved only by taking into account some aspects of semantics of the diagrams.

REFERENCES

[1] Booch G., Jacobson I., Rumbaugh J. (1998) The Unified Modeling Language User Guide, Addison-Wesley.

[2] Whitley D. (1991) A Genetic Algorithms Tutorial. In ed. Davis L. *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, pp. 1-101.

[3] Eloranta T., Makinen E. (2001) *TimGA — A Genetic Algorithm For Drawing Undirected Graphs*. Divulgaciones Matematicas Vol. 9 No. 2, pp. 155-171.

[4] Gansner E. R., Koutsofios E., North S.C., Vo K.-P., (1993) *A Technique for Drawing Directed Graphs*, IEEE Transactions on Software Engineering 19(3), 214—230

[5] Holland J., (1992) *Genetic Algorithms*, Scientific American 267, (1). 4450.

[6] North S.C., Woodhull G. (2001) On-line Hierarchical Graph Drawing. In Mutzel P., Jünger M. and Leipert S., Eds. Proceedings Graph Drawing, pages pp. 232-246, Springer.

[7] North S.C., Koutsofios E. (1994) Applications of Graph Visualization. In *Graphics Interface '94*, pages 235-245.

[8] Panagiotis L., Rajlich V., Korel B. (1991) Layout Heuristics for Graphical Representations of Programs. In *Proceedings of IEEE Conference on Systems, Man and Cybernetics*, pp. 1127-131.

[9] A. Papakostas, I. G. Tollis (1997) Incremental Orthogonal Graph Drawing In Three Dimensions. In Di Battista G., Ed., *Proceedings of the 5th International Symposioum on Graph Drawing*, Springer-Verlag.

[10] Pasek R., (2002) Techniki ewolucyjne w projektowaniu układu ścieżek na płytkach drukowanych. In Kwaśnicka H., Ed., *Zeszyt „Sztuczna Inteligencja Nr 1. Algorytmy Ewolucyjne – Przykłady zastosowań”*, Wyd. PWr.

[11] Sugiyama K., Tagawa S., Toda M. (1981) *Methods for Visual Understanding of Hierarchical System Structures*, IEEE Transactions on Systems, Man and Cybernetics SMC-11(2).