Joint NCC & IIS Bull., Comp. Science, 18 (2002), 15–26

Some modifications of Sugiyama approach*

D. Baburin

We present a graph drawing algorithm that was developed for real-life call graphs, data flows and class hierarchies. The algorithm is an extension of the hierarchical layout method of Sugiyama [12]. The main difference is that we achieved an orthogonal layout with the maximum number of edge bends equal to 2. For that purpose we have designed a special algorithm for horizontal coordinate assignment and solved the task of minimizing the number of orthogonal edge crossings between any pair of adjacent hierarchy levels. We have also tried out a new heuristic approach for creation of an initial position of nodes before starting a crossing reduction step of the algorithm. Finally, we provide some experimental results and references to applications that already use our algorithm.

1. Introduction

This work has appeared in the development of the RescueWare system by Relativity Technologies, Inc [8]. The system is designed for reengineering of a real world COBOL, PL/1 and Natural systems. Different parts of the system at all levels of abstraction needed a visual support to ease the user perception of reengineered systems and their parts. To this end, a special visual component BED (Boxes and Edges Diagram) has been designed.

The main part of this work describes a graph drawing algorithm that has been developed for automatic drawing of graphs in BED component. The algorithm is suited to draw large, directed, dense graphs with multiedges, labels and nodes of nonzero size. The layout emphasizes a uniform edge orientation and allows the grouping of nodes into the levels of a hierarchy. In reengineering such graphs are call graphs, data flows, class hierarchies, etc.

Hierarchical layout method of Sugiyama [12] has been modified and extended to fit our needs. The main difference was that we achieved an orthogonal layout with the maximum number of edge bends equal to 2. For that purpose we have designed a special algorithm for horizontal coordinate assignment and solved a task of minimizing the number of orthogonal edge crossings between any pair of adjusted levels of the hierarchy. Also a new heuristic was tried to create an initial position of nodes before starting a crossing reduction step of the algorithm.

*Supported by the Russian Foundation for Basic Research under Grant No 01-01-794 and the Ministry on Education of Russia.

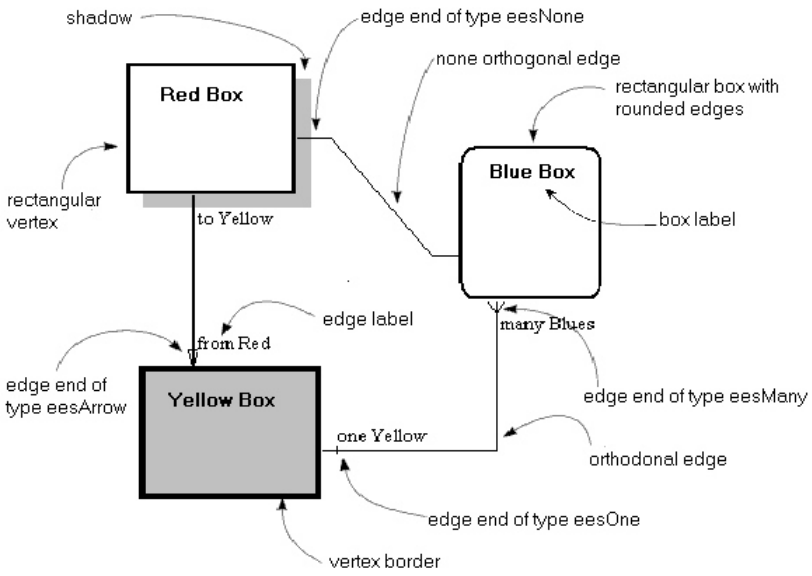

Figure 1. Graphical potentialities of the BED component

In the next section we would give some functional descriptions of the visual and programming interface of the BED component. Section 3 deals with the graph drawing algorithm that we use to place graphs and diagrams. The last section sketches some experimental results and references to applications that use our algorithm.

2. BED

A practical aspect of program visualization does not allow us to be limited just by visualization of a graphical structure. Some extended information should also be displayed. Studies like [3] and [4] show what kind of extended information is desired for visualization of entity relationship diagrams and different hierarchies. Work with these graphical structures requires boxes of different sizes, label placement mechanism, and several different types of edge ends. Also we need to take into account that diagrams could have several edges between two vertices and edges that connect a vertex with itself. We have considered all these requirements while designing our component. In Figure 1 you can see the variety of graphical potentialities implemented in the BED component.

The BED component has one constraint on the form of edges between vertices. Edges are drawn as polygonal lines with labels and marks on them,

they could be orthogonal or nonorthogonal. But in any case the form of an edge is defined by two fixed points. The geometry of polygonal lines could be uniquely reconstructed by positions of these two points according to some internal rules and the edge type. Figure 2 shows what kind of edges could be drawn in our component. Fixed points are displayed by bold dots.

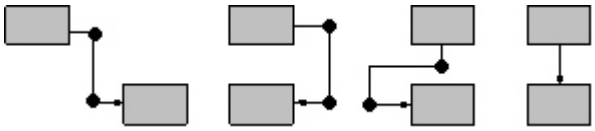

Figure 2. Edges in the BED component

We concentrate our work on the program analysis and transformation rather than on visualization of arbitrary graphs and diagrams. So the main requirement to the component was its extensibility and easiness of embedding it into other environments and applications. In Windows, the most adequate way to satisfy such a requirement is to use a mechanism of ActiveX controls.

In several cases, a printable document should be provided as a result of work with the component. To do this, we implemented procedures of image storing into graphical formats EMF (Enhanced Metafile) and BMP (Windows Bitmap).

There are two typical scenarios of forming the structure of our diagram. In the first case the graph structure is filled through the programming interface and then it could be manually edited by the user. For the second case we designed a special XML format to store information that is required for reconstruction of the internal state of the diagram. This gives a user the possibility to create additional methods of diagram processing by using such XML files as interface to the component. One of these external methods is the graph drawing algorithm discussed in the next sections.

3. Graph drawing algorithm

In a number of works [9–11], Sander presents his methods and heuristics on how to combine hierarchical approach with the methods of orthogonal graph drawing and Manhattan conventions. In our work we develop and extend these ideas. The main difference is that we have achieved an orthogonal layout with the maximum number of edge bends equal to two as compared to four bends in Sander's works. We have also solved the task of minimizing the number of orthogonal edge crossings between any pair of adjacent levels

of hierarchy and tried a new heuristic to find an initial position of all nodes before starting the crossing reduction step of the algorithm.

Hierarchical approach is suited for drawing graphs in any direction: from top to bottom and from left to right. We have chosen the left to right direction according to the following argument. All vertices and edges in graphs that do appear while reengineering contain some semantic information that is depicted as the box and edge labels in the diagram. If layers in a graph are directed from left to right, then most of the edges have horizontal direction and this allows us to draw edge labels on them. It was also noticed that most of the graphs in our processes have relatively small depths (the length of the longest path). Being combined with a large width of boxes, this fact makes horizontal direction more appropriate for creating compact and balanced diagrams. So, in our work the coordinates X and Y were swapped comparing to the usual description of this approach.

Now we will remind the basis of the hierarchical approach as described in [4]. This approach consists of the following steps:

- The *layer assignment* step constructs a layered digraph where the vertices of a graph G are assigned to the layers $L_1, \ , L_h$, such that if (u, v) is an edge with $u \in L_i$ and $v \in L_j$, then $i < j$. In the final drawing, each vertex in the layer L_i will have the x coordinate equal to i. Also during this step, the dummy vertices are inserted along the edges that span more than two layers so that each edge after that connects the adjusted layers.
- The *crossing reduction* step produces the order for the vertices on each layer. The order of the vertices on the layers determines the topology of the final drawing and is chosen in such a way that the number of crossings is kept as small as possible.
- The *y-coordinate assignment* assigns the final y coordinates to the vertices while preserving the ordering computed in the crossing reduction step. The edges are also positioned during this step.

In Figure 3, we present a part of a diagram constructed by our algorithm to illustrate the further discussion.

3.1. Heuristic method for the crossing reduction step

The crossing reduction step considers the problem of drawing a layered digraph with a small number of edge crossings. The problem of minimizing the number of edge crossings in a layered digraph is NP-complete, even if there are only two layers. So a variety of heuristics has been used to reduce the crossings. Surprisingly, but we did not find any studies that refer to the

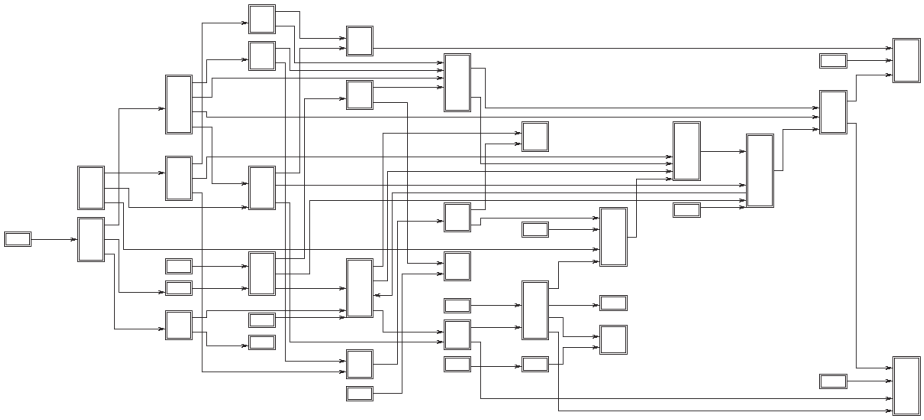

Figure 3. Sample layout produced by our algorithm. The graph has 40 vertices and 60 edges

problem of finding a good initial order of vertices before applying different iteration-based heuristics. So, we present a heuristic method that could be used to form an initial ordering with a relatively small number of edge crossings. Then this ordering could be improved by any of the known crossing reduction heuristics, such as barycenter or median.

The main idea is a refinement of the tree-based initial ordering of nodes in the layers presented in [6]. As an input, our heuristic takes the spanning tree constructed during the layer assignment step or right after it. Then it changes the order in which subtrees appear in this tree to minimize the crossings that are formed by non-spanning edges. This heuristic is recursive and is applied to each vertex to determine the order of its subtrees in a spanning tree. The subtrees are ordered greedily according to the number of edges connecting them to the left top and down neighbors.

Let V be a current vertex observed by our heuristic and T_i, where $i \in \{1..p\}$, be its subtrees. The algorithm orders these subtrees by finding the most appropriate subtree to place at each step. At each step the algorithm places the best subtree with all subtrees that could be found recursively inside of it. Let us assume that k subtrees are already placed and the algorithm looks for the next subtree from the T_i set, where $i \in \{k+1..p\}$. For this we calculate a balance for each unplaced subtree. The balance is the number $B_{1j} - B_{2j}$, where B_{1j} is the number of edges that connect this subtree with a part of the graph that is already placed, and B_{2j} is the number of edges that connect vertices of this subtree with the rest of the graph. A subtree with the maximal balance is chosen and is being placed next. After that the algorithm recalculates balances of the residual subtrees.

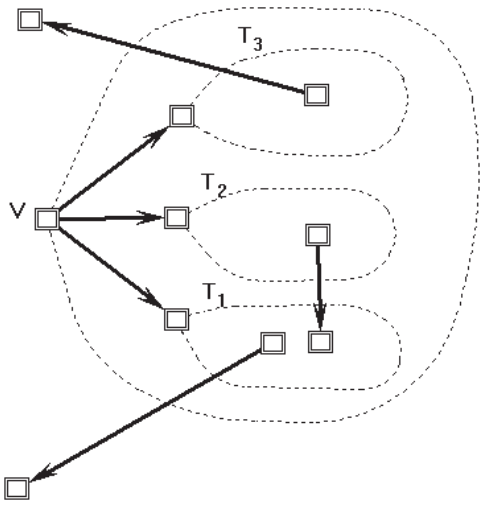

Figure 4. Placing of subtrees

The basis of this approach is that the higher balances correspond to cases when there exist a lot of edges to a part of a graph that was already placed. And if we place the subtrees bottom-up, then these edges would push this subtree to be placed before the others.

The algorithm is kept being quadratic by introducing a special binary ordering on the vertices that allows us to recalculate the subtree balances quickly.

Figure 4 illustrates placing of subtrees T_1, T_2, T_3 for a vertex V.

Step 1. None of subtrees are placed yet. Balance for T_1: $B_{11} - B_{21} = 1 - 1 = 0$, for T_2: $B_{12} - B_{22} = 0 - 1 = -1$, for T_3: $B_{13} - B_{23} = 0 - 1 = -1$. T_1 is chosen.

Step 2. T_1 is already placed, choosing between T_2 and T_3. Balance for T_2: $B_{12} - B_{22} = 1 - 0 = 1$, for T_3: $B_{13} - B_{23} = 0 - 1 = -1$. T_2 is chosen. So, the calculated order is T_1, T_2, T_3.

3.2. The vertical coordinate assignment

Originally the step of the vertical coordinate assignment in a hierarchical layout deals only with vertices and leaves the task of edge positioning to further steps of the algorithm. In our work we divide the edge-positioning task into two separate steps. The first step is performed along with the vertices vertical coordinate assignment and each edge gets vertical coordinates of its

horizontal segment at this step. And at the final edge-positioning step, the vertical edge segments get their horizontal coordinates.

In our algorithm we introduce a notion of an edge width. Each edge has an assigned width of some minimum value *Wmin*. The width could appear to be greater than *Wmin*, if the edge has some label. The other reason for having a greater width could be multi edges that are combined into one edge at the preprocessor step of the algorithm. Later, when a drawing is made, we draw all edges from such multi edge package in a space that was assigned to this edge. This conversion allows us to draw parallel multi edges.

The concept of the edge width is also used in assigning vertical coordinates to the vertices. Originally, at this step coordinates are assigned according to some order generated at the previous step (while crossing reduction). Then vertical coordinates are adjusted by some iteration process, like the pendulum method. In our drawing algorithm we cannot use such technique because it cannot assure that long edges consisting of dummy vertices are drawn as straight lines. In [9], the method that draws segments of dummy vertices in one horizontal line is presented. But this algorithm cannot be applied together with our BED component, because it generates edges that have at most four bends and only two bends are allowed by our component.

In our approach we are not assigning exact vertical coordinates to vertices — instead of it we assign some vertical ranges. These ranges can be bounded by some coordinates or can dynamically depend on the neighbor ranges. After the range assignment for all layers, we are fixing the precise coordinates by straightening the maximum amount of edges.

In the range assignment, we guarantee that all dummy vertices of one chain would get the same vertical range. For this we traverse the hierarchy from the layer n to layer *1* and for each nondummy vertex we form a sorted list of edge ends that are entering this vertex and has a dummy vertex at the other end. Such edges get a range of possible vertical coordinates that are bound by the vertical coordinate of a box that is currently being placed. The dummy vertices get ranges according to the ranges already assigned in the previous layer.

Sometimes in this process the vertex coordinates interlace with coordinates of dummies that got their predefined vertical range. In such cases the interfering ranges are squeezed. And if such interlacing could not be eliminated by this operation and the vertex could not be placed according to the order defined at the previous step of the algorithm, then such an order is adjusted and this vertex jumps over dummy vertices that took the required space by their ranges.

In Figure 5, we show the example of range assignment for edges. While placing the vertex $V3$, all incoming edges were sorted and for edges from

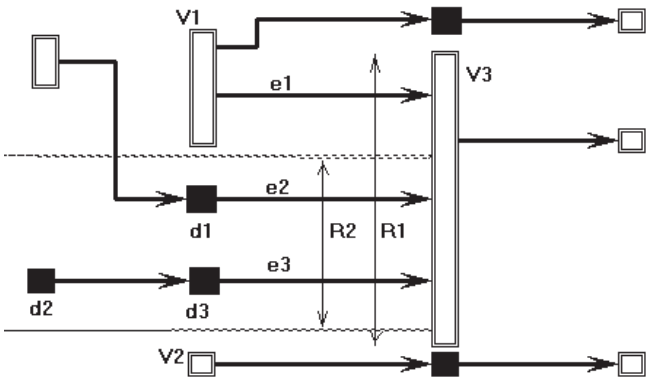

Figure 5. Range assignment for edges

dummy vertices ($e2$ and $e3$) the vertical range $R1$ was assigned. Then, when the algorithm places the next level with vertices $v1$, $d1$, $d3$ and $v2$, it shrinks the range $R1$ to a smaller range $R2$. Now the range bounds are defined by vertical coordinates of vertices $V1$ and $V2$.

3.3. Positioning of edges

The problem of edge positioning is very similar to the well-known issue from VLSI channel routing [13]. The task is to route orthogonal edges between two levels of hierarchy when edge endpoints are fixed.

When we draw edges between two adjacent levels of hierarchy, we divide the horizontal space between layers V_i and V_{i+1} into K_i vertical levels that would contain vertical segments of edges.

Let us say that the edges (U_1, D_1) and (U_2, D_2), where U_i and D_i stand for the vertical coordinates of their up and down ends accordingly, are *potentially crossing* if there exists $i \in \{1, 2\}$ such that $U_i \in (U_{3-i}, D_{3-i})$. This potential crossing could be eliminated if there exists $i \in 1, 2$ such that $U_i \in (U_{3-i}, D_{3-i})$ and $D_i < D_{3-i}$ and if these edges follow the same vertical direction. The relationship of potential crossings sets a partial order in the set of edges.

It is clear that any pair of edges that are potentially crossing should be drawn in different vertical segments. For eliminated crossings we need to use the levels in a proper order to actually eliminate them. Elimination of all such crossings would minimize the number of edge crossings for this pair of adjusted levels of the hierarchy. The number of vertical levels should also be minimized.

In VLSI terminology, this channel routing problem is completely characterized by the vertical constraint graph and the horizontal constraint graph. The first graph constraints the order of vertical segments for edges that has the same vertical coordinates of their ends. And the second graph restricts two edges to be placed in one vertical segment if they do have a potential crossing.

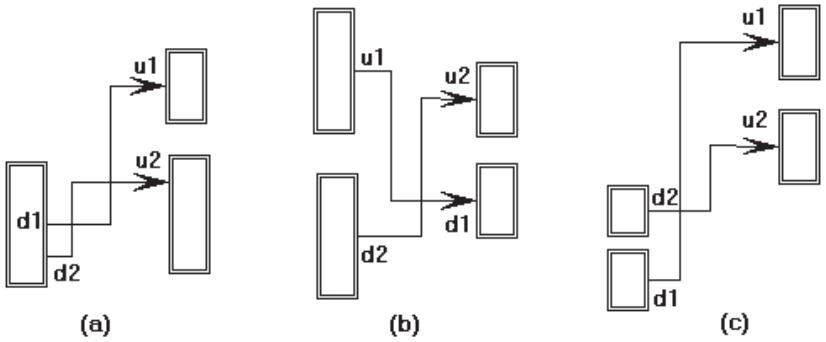

Figure 6. Edge crossings

Figure 6 illustrates three cases of edge crossings. All these cases correspond to the relation of potential crossing because $U_2 \in (U_1, D_1)$. But only in case (a) such potential crossing could be eliminated: $U_2 \in (U_1, D_1)$ and $D_2 < D_1$ and both edges point upward. To do this, the vertical segment of the edge $[d2, u2]$ should be drawn to the right of the vertical segment of the edge $[d1, u1]$.

In our approach we solve the routing problem by constructing a supplementary graph G'. We create a vertex of G' for each edge in the original graph and create an edge for each pair of edges that are potentially crossing. An edge in a graph G' is directed if the potential crossing presented by it can be eliminated. The edge direction then shows what edge in the original graph should be drawn first (the leftmost edge) to eliminate such crossing.

So, G' is a horizontal constraint graph that is directed so to represent the information of the vertical constraint graph. We do not have potential doglegs in our case, so the vertical constraint graph becomes acyclic and could be depicted as a partial order on the vertices of the horizontal constraint graph.

Now our edge routing problem could be formulated in terms of a coloring problem for a supplementary graph G'. The vertices of G' need to be colored in a minimum number of colors so that any adjusted vertices would have different colors. Also, all directed edges should point from vertices with smaller color indices to vertices with larger color indices. If such a coloring

scheme is found then we can draw edges in the original graph by assigning vertical levels to colors in the increasing order.

It is known that the coloring problem is NP-hard, but some good linear heuristics exist for it [5]. Our challenge was to adapt these heuristics for directed graphs.

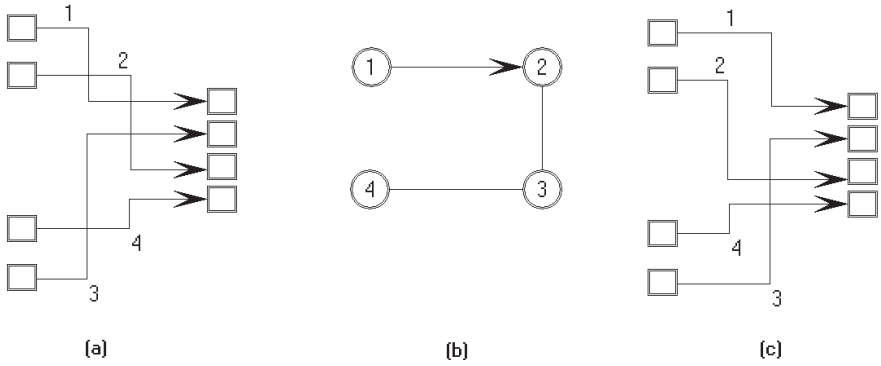

Figure 7. Supplementary graph construction

In Figure 7 we show how the task of edge positioning for the simple graph is solved. 7(a) represents an initial ordering, with three potential crossings: edges 1 and 2, 2 and 3, 3 and 4. Only one crossing could be eliminated — between edges 1 and 2. These facts are depicted in the supplementary graph G' 7(b). To color G', only two colors are needed: edges 1 and 3 get the same color and edges 2 and 4 get the other one. Figure 7(c) shows the final drawing with only two crossings and only two layers for vertical segments.

4. Conclusion

The first version of our algorithm was implemented in 2000 by means of C++ and with the extensive use of STL. Then it was embedded into the RescueWare system [8] and used there to place the routine call graphs, data flows, and different kinds of the system diagrams. Our experience in using it in this system is described in [2].

Experiments have shown that the algorithm can work with "real-world" graphs. On PII-300 computer, it had placed any graph of less than 100 vertices in less than one second. Each thousand of vertices required from 10 to 20 seconds of additional run time. Actually graphs larger than one hundred vertices were out of our interest due to other visualization aspects — screen area, scale, labels, navigation, etc. Different methods were tried to

ease the graph understanding at the level of a graphical component rather than at the level of a placement algorithm [1].

In 2001 the graph drawing algorithm was provided with an interface that enabled its usage in the Higres system [7].

An open question is how does the overall number of bends compare with methods where the number of bends on a single edge can be arbitrary. We also need to gather more accurately run-times and data on how our initial subtree ordering improves the work of different heuristic methods for crossing reduction step in Sugiyama approach.

Acknowledgements. We would like to acknowledge the work of M. Bulyonkov and N. Filatkina as the authors of the BED component.

References

[1] Baburin D. E., Bulyonkov M. A., Emelianov P. G., Filatkina N. N. Visualizing facilities in program reengineering // Programming and Computer Software. — Interperiodica Publishing, 2001. — Vol. 27, N 2. — P. 69–77.

[2] Baburin D.E. Using graph based representations while reengineering. // Proc. of Intern. Conf. on Software Maintenance and Reengineering, CSMR2002. — Budapesht, 2002. — P. 203–206.

[3] Batini C., Talamo M., Tamassia R. Computer aided layout of entity-relationship diagrams // J. of Systems and Software. — 1984. — Vol. 4. — P. 163–173.

[4] Di Battista G., Eades P., Tamassia T., Tollis I.G. Algorithms for drawing graphs: annotated bibliography // Comput. Geom. Theory Appl. — 1994. — P. 235–282.

[5] Bulyonkova A.L, Aproximate algorithm of coloring for large graphs // Problems of Theoretical and System Programming. — Novosibirsk State University. — 1982. — P. 81–86 (in Russian).

[6] Gasner E.R., Koutsofios E., North S.C., Vo K.P. A technique for drawing directed graphs // IEEE Transactions on Software Engineering. — 1993. — Vol. 19, N 3. — P. 214–230.

[7] Lisitsyn I. A., Kasyanov V.N. Higres — visualization system for clustered graphs and graph algorithms // Proc. of the Symposium on Graphdrawing, GD'99. — Lect. Notes Comput. Sci. — 1999. — Vol. 1731. — P. 82–89.

[8] Relativity Technologies RescueWare system http://www.relativity.com

[9] Sander G. A fast heuristic for hierarchical mangattan layout // Proc. of the Symposium on Graph Drawing, GD'95. — Lect. Notes Comput. Sci. — 1995. — Vol. 1027. — P. 447–458.

[10] Sander G. Graph layout through the VCG tool // Proc. of the Symposium on Graphdrawing, GD'94. — Lect. Notes Comput. Sci. — 1995. — Vol. 894. — P. 194–205.

[11] Sander G. Graph layout for applications in compiler construction. 1996. — (Tech. Rep./ Universitas des Saarlandes; N A/01/96).

[12] Sugiyama K., Tagawa S., Toda M. Methods for visual undertanding of hierarchical systems // IEEE Trans. Syst., Man and Cybern. — 1981. — Vol. 11, N 2. — P. 109–125.

[13] Yoshimura T., Kuh E. Efficient algorithms for channel routing // IEEE Trans. on Computer-Aided Design of Integrated Circuit and Syst. — 1982. — Vol. CAD-1. — P. 25–35.