# Disconnected Graph Layout and the Polyomino Packing Approach

Karlis Freivalds[1], Ugur Dogrusoz[2], and Paulis Kikusts[1]*

[1] Institute of Mathematics and Computer Science, Univ. of Latvia, Riga, Latvia
{karlisf,paulis}@mii.lu.lv
[2] Computer Engineering Department, Bilkent Univ., Ankara, Turkey
ugur@cs.bilkent.edu.tr

**Abstract.** We review existing algorithms and present a new approach for layout of disconnected graphs. The new approach is based on polyomino representation of components as opposed to rectangles. The parameters of our algorithm and their influence on the drawings produced as well as a variation of the algorithm for multiple pages are discussed. We also analyze our algorithm both theoretically and experimentally and compare it with the existing ones. The new approach produces much more compact and uniform drawings than previous methods.

## 1 Introduction

Graphs model the complex information of a system of discrete objects and their relationship. *Graph layout* is the automatic positioning of the nodes and edges of a graph in order to produce an aesthetically pleasing drawing that is easy to comprehend [5,8].

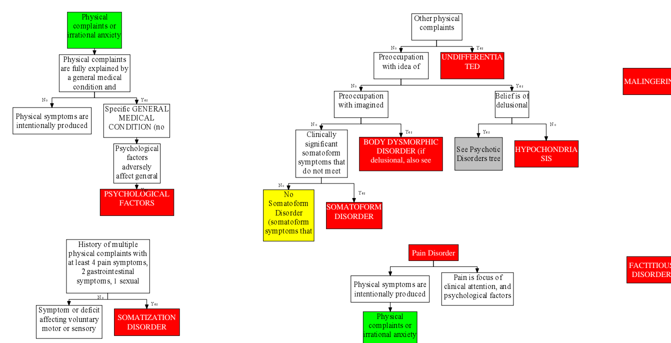


**Fig. 1.** An example of a disconnected graph.

* Research supported in part by NIST, Advanced Technology Program grant number 70NANB5H1162 and Tom Sawyer Software, Oakland, CA, USA.

P. Mutzel, M. Jünger, and S. Leipert (Eds.): GD 2001, LNCS 2265, pp. 378–391, 2002.
© Springer-Verlag Berlin Heidelberg 2002

Many graph layout and editing systems have been developed in the past [5, 7,11]. One essential aspect that has not been addressed sufficiently, is the layout of disconnected graphs; that is, the placement of the components (possibly consisting of a single isolated node) of a disconnected graph. Disconnected graphs occur rather frequently in real life applications either during the construction of a graph interactively or because of the nature of the application (Figure 1).

Most graph layout algorithms assume a graph to be connected and try to minimize the area needed for the resulting drawing. No matter how effective such an algorithm is, the space wasted overall could be arbitrarily large if the relative locations of disconnected objects of a graph are chosen by a naive, inefficient method.

Another key parameter here is the aspect ratio of the region (e.g., a window) within which the graph is to be displayed (Figure 2). When displaying a graph, the larger the wasted space is, the less visible objects will be, making the visualization process more difficult. Thus, a disconnected graph layout algorithm must strive for a packing of disconnected objects which respects the aspect ratio of the region in which it is to be displayed.

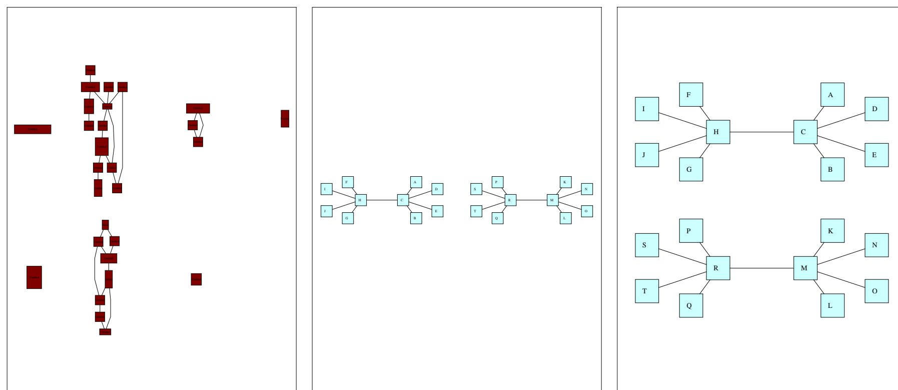


**Fig. 2.** How a naive disconnected graph layout algorithm can make inefficient use of the area **(left)**, and why the aspect ratio of the region in which the graph is to be drawn should be taken into account during disconnected graph layout **(middle and right)**.

In this paper, we review existing two-dimensional packing algorithms for the layout of disconnected graphs for a specified aspect ratio based on *strip-packing*, *tiling*, and *alternate-bisection* methodologies [6], and introduce a new algorithm that represents disconnected objects with polyominoes as opposed to rectangles. We also discuss the experimental results obtained and compare our algorithm with the previous ones. As expected, the new approach, which uses a more

accurate representation for the objects, produces much more compact results. The drawings are also more aesthetically pleasing as the new approach places the objects more uniformly.

## 2 Definitions and Basics

Throughout the paper, the terms "graph object", or simply "object", are used interchangeably to denote a component or an isolated node of the graph to be laid out.

The tightest rectangle bounding the drawing of a graph object or the entire graph is said to be its *bounding rectangle*. The size of a graph's drawing is identified with its bounding rectangle's. The *aspect ratio* of a rectangle $R = (W, H)$ is equal to $W/H$.

Most layout algorithms represent graph objects with either points or rectangles in the plane. A *polyomino* is a geometric figure formed by joining unit squares at the edges. In our new approach we use polyominoes to represent graph objects (Figure 3).

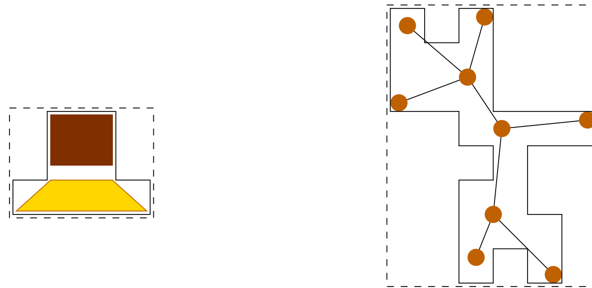

**Fig. 3.** Polyominoes facilitate more accurate geometric representation for graph objects. Two different representations of an isolated node and a graph component: rectangle (dashed) and polyomino (solid).

The task for a disconnected layout algorithm is to position a set of objects represented by rectangles or polyominoes with ordered dimensions (i.e., no rotations allowed) such that no pair of objects overlap and the area of the bounding rectangle of the drawing is minimized, respecting the aspect ratio of the region in which the graph is to be displayed. There has been extensive research done on two-dimensional packing of rectangles [3,1,4]. In the graph layout version of the problem, the user also specifies a *desired aspect ratio* for the resulting drawing so that the scaling that needs to be done before displaying the graph is minimal (Figure 2).

For an arbitrary list of $n$ objects $L_n$, or simply $L$, let $\mathrm{A}^A(L)$ denote the area actually used by a particular algorithm $A$ when applied to $L$. The *wasted space* is the unoccupied area of the packing: $\mathrm{WS}^A(L) = \mathrm{A}^A(L) - \sum_{i=1}^{n} \mathrm{A}_i$, where

$A_i$ is the area of object $L_i$. Similarly, the *fullness* of a packing expresses, in percentage, how effectively the area is used by the packing algorithm: $\mathrm{F}^A(L) = 100 \cdot (\sum_{i=1}^{n} A_i)/\mathrm{A}^A(L)$. The *adjusted fullness* of a packing $\mathrm{AF}^A(L)$ $(\leq \mathrm{F}^A(L))$, expresses the fullness of a packing, in percentage, with respect to the desired aspect ratio. To be precise, it considers the additional area wasted when the final drawing is displayed in a region of desired aspect ratio DAR: $\mathrm{AF}^A(L) = \mathrm{F}^A(L) \cdot \frac{\mathrm{AR}^A(L)}{\mathrm{DAR}}$ where $\mathrm{AR}^A(L)$ is the aspect ratio of the packing produced by algorithm $A$ when applied to objects $L$ and we assume $\mathrm{AR}^A \leq \mathrm{DAR}$. Figure 4 illustrates this with an example, in which $\mathrm{F}^A(L) = \frac{A}{A+B}$, whereas $\mathrm{AF}^A(L) = \frac{A}{A+B+C}$.

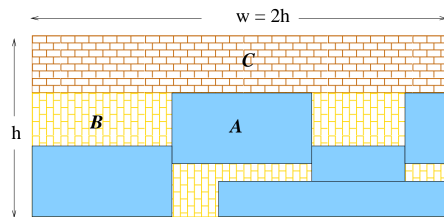


**Fig. 4.** Total area in which rectangles are packed is divided into three disjoint regions A (rectangles), B (wasted area), and C (additional area wasted when displayed in a region of aspect ratio 2).

**Strip-Packing:** One can find substantial literature on the design and analysis of algorithms for two-dimensional packing [1,3,4], the most popular version being *strip-packing*. In strip-packing, given a list of $n \geq 1$ rectangles $L_n = (R_1, \ldots, R_n)$, each having ordered dimensions $(W_i, H_i)$, they are to be packed into a semi-infinite strip of unit width without any overlaps in order to minimize the height of the packing. This problem has applications in many areas including stock-cutting, two-dimensional storage problems, and resource-constrained scheduling in computer systems [2].

The most popular approach to strip-packing is the level algorithms. *First-Fit Decreasing Height* (FFDH) is a level algorithm, in which, at any point in the packing sequence, the next rectangle to be packed is placed left-justified on the first level on which it will fit. If none of the current levels will accommodate this rectangle, a new level is started. *Best-Fit Decreasing Height* (BFDH) is similar to FFDH except that the rectangles are packed, whenever possible, on current levels where they fit best.

For an arbitrary list of $n$ rectangles $L_n$, all assumed to have width no greater than 1, $\mathrm{OPT}(L)$ denotes the minimum possible bin height within which rectangles in $L$ can be packed.

**Ordered One-Dimensional Packing:** For a set of rectangles, *one-dimensional packing* or simply 1D packing along x-axis (y-axis) corresponds to the process of ordering these rectangles with respect to their x-coordinates (y-coordinates)

without any overlaps to *minimize* the total width (height) of the bounding rectangle. If the current relative positions of rectangles are to be preserved in the packing, we call it *ordered 1D packing*. Ordered 1D packing of $n$ objects can be performed in $O(n \log n)$ time [12].

## 3 Related Work

In this section, we review the existing algorithms for disconnected graph layout, which represent disconnected objects with rectangles. Detailed information on these algorithms may be found in [6].

### 3.1 Strip-Packing Method

This method directly applies a known strip-packing algorithm such as BFDH. The width of the strip (equivalently, the factor by which the rectangle dimensions are to be scaled) is calculated based on the desired aspect ratio, using the theoretical performance of the strip-packing algorithm. With BFDH, assuming object dimensions to be independent uniform random samples from the interval $[0, 1]$ and $\mathrm{OPT}(L) \approx n/4$, the expected value of adjusted fullness, $E[\mathrm{AF}^{BFDH}(L_n)]$, is shown to be 58.8 [6]. However, these calculations are based on the worst-case performance bounds and are rarely met in practice, making it not a particularly good "guess" for the bin width.

The algorithm is of $O(n \log n)$ time complexity.

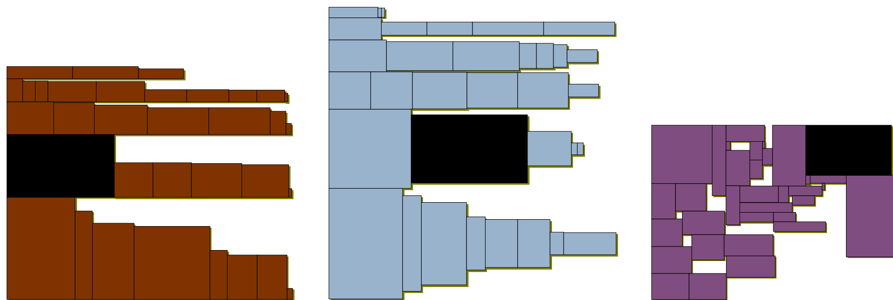

**Fig. 5.** The same graph laid out with strip-packing, tiling, and alternate-bisection methods, respectively for desired aspect ratio 1.0.

### 3.2 Tiling: Strip-Packing with Variable Width Strip

The tiling method eliminates the need to "guess" the right size strip by maintaining a bin whose width *dynamically* changes (i.e., increases). The algorithm starts by creating an initial level and placing the first rectangle in this level. It

proceeds by determining whether the next rectangle in line should be added to one of the existing levels (the one which is the least utilized at the moment) or to a newly created level. The rectangle is tiled on one of the existing levels if there is enough room. Otherwise, a decision is made on whether the current strip width should be enlarged or a new level should be formed to keep the aspect ratio closer to the desired one.

In general, the tiling algorithm does not assume any particular ordering of the objects. However, experiments show that when graph objects are sorted in nonincreasing height, most compact drawings are obtained. Notice that when objects are processed in order of nonincreasing height, the algorithm turns into a variation of a strip-packing algorithm, BFDH to be more specific, where the strip width is dynamically increased as necessary to better fulfill the aspect ratio constraint.

The algorithm is of $O(n \log n)$ time complexity.

### 3.3 Alternate-Bisection Method

This divide-and-conquer method works by bisecting the disconnected objects of a graph alternately as follows. The objects are bipartitioned using a metric such as total area and objects in each partition are recursively laid out. The recursion continues until a partition consists of a small, constant number of objects (e.g., one) whose optimal layout becomes easy if not trivial. At the end of each recursive step, when placing the two embedded partitions relatively, the orientation is alternated. For instance, the last step would place the two already positioned partitions side by side (horizontally) if the four partitions in the previous step were placed one on top of the other (vertically) pairwise.

The theoretical analysis prove that the total area wasted by the algorithm for $n$ objects, $W(n)$, is roughly $O(n^{1.41})$ [6], which is quite inefficient. However, when simple alternating ordered 1D packings are applied in each recursive step (e.g., objects in upper (lower) left partition are packed downwards (upwards), towards the horizontal separating axis in Figure 6), the experimental results show that much more compact results are obtained [6]. The overall time complexity of the algorithm is $O(n \log^2 n)$.

For independent, uniformly distributed random object dimensions, this algorithm will not favor one orientation over the other and yield "square-like" drawings. The desired aspect ratio can be respected by this algorithm by initially recursively partitioning the set of objects into two, one partition to be laid out with aspect ratio 1.0 and the other with $\text{DAR}(L) - 1.0$ $(= \frac{w-h}{h})$, assuming $\text{DAR}(L) = \frac{w}{h} > 1.0$ (Figure 6). Alternatively, the object dimensions can be scaled with respect to the desired aspect ratio as a preprocessing step, after which, the desired aspect ratio may be assumed to be 1.0.

### 3.4 Comparison of the Methods

In [6], experiments with graphs laid out with random aspect ratio and with random object dimensions are presented. Notice here that the graph objects

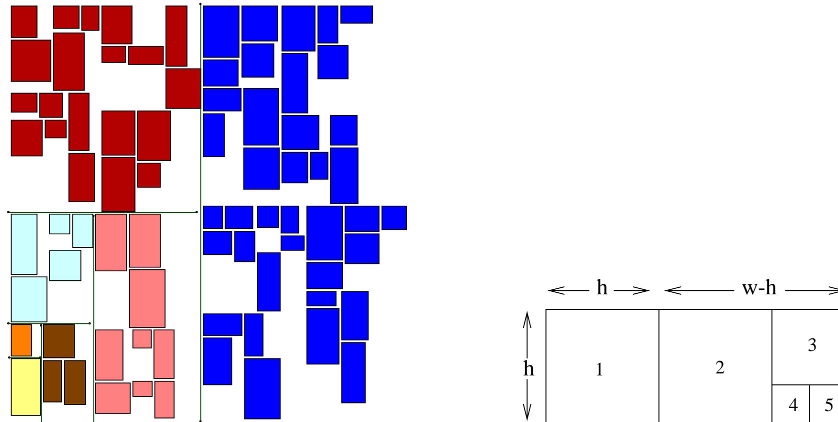


**Fig. 6.** An example of the alternate-bisection method; on one branch of the recursion, alternately partitioned objects are shown with separating lines and different colors **(left)**. An example of how the alternate-bisection method can be adapted to an arbitrary aspect ratio **(right)**.

are represented with rectangles and the area wasted by such representation is ignored. In the context of graph layout, it is argued that the object dimensions are not completely of uniform distribution since the two types of disconnected objects, isolated nodes and larger components, in most cases will be of highly varying dimensions. Experiments conducted with two groups of objects with dimensions uniformly distributed within each group but with different means are presented in Figure 7.

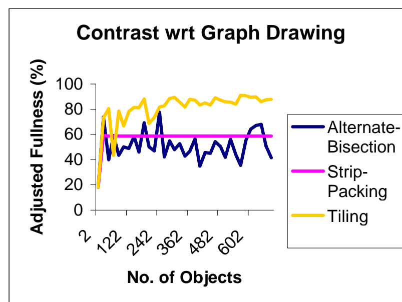


**Fig. 7.** Comparison of the performance of the three methods using a distribution model of dimensions that is more suitable in the context of graph layout.

In terms of execution time, both split-packing and tiling methods are superb since they are of $O(n \log n)$. The alternate-bisection method, on the other hand, gets a little slow as the number of objects are over a thousand. Considering that

most interactive graph drawing applications will not consist of more than, say one hundred, disconnected objects, this method is also of practical value.

In terms of the quality of the packings produced, the experiments show that the tiling method clearly produces the most compact drawings. However, the results obtained from the alternate-bisection method tend to be more aesthetically pleasing since the objects are generally distributed more uniformly (Figure 5).

## 4 Polyomino Packing Approach

In this approach each graph object is represented by a polyomino. We define a polyomino as a finite set of $k \geq 1$ cells of the infinite planar square grid $G$ that are fully or partially covered by the drawing of the object. If the case that an object is placed completely inside another one is not desirable, the definition can be modified and the uncovered grid cells that are completely bounded by the covered ones can be included as well.

Given a set of polyominoes $P_i, 1 \leq i \leq n$, packing them into a minimum area is clearly NP-hard, even when the polyominoes are restricted to rectangles [9]. Our heuristic algorithm for polyomino packing is a greedy one: it places the objects one by one, finding the optimal place for the new object, one at a time, with respect to the already placed ones. The optimal place for a polyomino is simply calculated as the grid cell $G_{xy}$ located at $(x, y)$ where the function $\max(|x|, |y|)$ is minimized over all grid cells. The cost function defines the order in which the cells are examined and this order is the same for all polyominoes.

A grid data structure is used to represent free grid cells, which are later marked as occupied as polyominoes are placed. In order to find the best place the algorithm PackPolyominoes looks sequentially through all cells in the increasing order of the cost function defined above. If an available spot (a set of unoccupied grid cells where the polyomino fits) is found, it is placed there and the corresponding grid cells are marked as occupied. When testing for intersections, we simply go through all polyomino cells and test whether each can be placed in a free grid cell.

Our experiments show that the quality of the packing depends very much on the order in which the polyominoes are processed. The best results are obtained when they are ordered and processed from the largest to the smallest, which conforms to the ordering in heuristic approaches for the bin packing [2] and strip-packing problems [3]. The size of each polyomino can be defined in several ways including its number of cells (i.e., area) and the perimeter of its bounding rectangle. Experiments show that both give similar results, so we choose the perimeter of the bounding rectangle for calculating the object sizes for the ease of implementation.

Here is a pseudo code of our algorithm:

**algorithm** PackPolyominoes($P_i, 1 \leq i \leq n$)
(1) sort $P_i, 1 \leq i \leq n$ in the order of nonincreasing size
(2) initialize the grid $G$ using the sizes of $P_i, 1 \leq i \leq n$
(3) **foreach** polyomino $P_i$ **do**

(4) calculate $(x, y)$ such that the cost function is minimized
(5) **while** cannot place $P_i$ in $G$ centered at $(x, y)$ **do**
(6) calculate next $(x, y)$ using the cost function
(7) **end while**
(8) mark the cells in $G$ covered by $P_i$ as occupied
(9) **end foreach**

Figure 8 illustrates an example drawing produced using our algorithm for the component placement of a forest. Also see Figure 9 for the drawing produced by our algorithm for the graph in Figure 1.

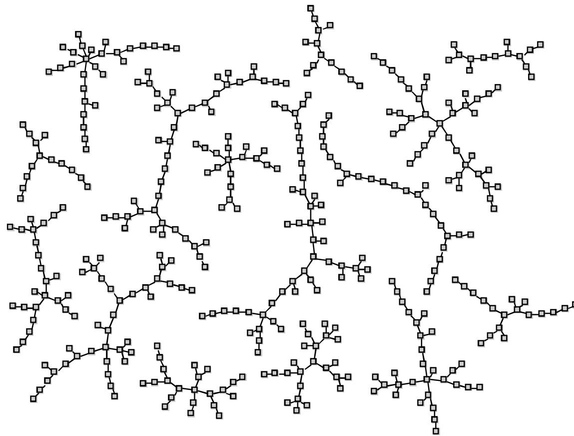

**Fig. 8.** An example packing produced by our algorithm.

### 4.1 Parameters

The grid step $l$ is obviously the most significant parameter of this approach. We would like to guarantee that the average polyomino size $s$ is not exceeding some constant $c$:

$$s = \frac{1}{n}\sum_{i=1}^{n}\lceil\frac{W_i}{l}\rceil\lceil\frac{H_i}{l}\rceil \leq c$$

$$\Rightarrow \quad \frac{1}{n}\sum_{i=1}^{n}(\frac{W_i}{l}+1)(\frac{H_i}{l}+1) \leq c$$

$$\Rightarrow \quad \sum_{i=1}^{n} W_i H_i + l\sum_{i=1}^{n}(W_i + H_i) + l^2 \leq cnl^2$$

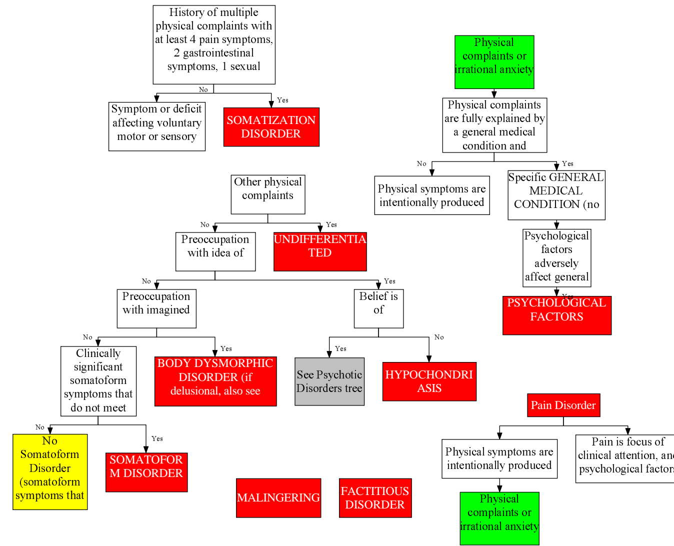


**Fig. 9.** The disconnected graph in Figure 1 laid out with the new algorithm (displayed with the same width to illustrate better usage of the area of aspect ratio 1.0).

Consequently, the grid step $l$ can be calculated from the following quadratic equation:

$$(cn - 1)\, l^2 - \sum_{i=1}^{n} (W_i + H_i)\, l - \sum_{i=1}^{n} W_i H_i = 0$$

With the average polyomino size $s \leq c$, the total area of all polyominoes does not exceed $n \cdot s$. Practical experiments show that the algorithm produces drawings of almost constant fullness (Figure 12), so the total packing area is also $O(n \cdot s)$.

To find a suitable place, each polyomino is tested for each cell. The test whether a polyomino fits in the specified place can be performed in $O(s)$ time in the worst case. Thus the complexity of the algorithm is $O(n^2 \cdot s^2)$. Since $c$ and consequently $s$ are constants this yields an $O(n^2)$ time overall, based on experimental results.

The value of the constant $c$ must be selected carefully since its influence on the running time can be as much as $O(c^2)$. Figure 10 shows how the approximation quality paremeter $c$ influences the adjusted fullness and the running time. For measurements, as a typical example a random forest of 300 trees of random order between 2 and 100 were generated (Figure 8 shows a smaller example of such a forest). The trees were laid out with a spring embedder algorithm similar to [10]. For small values of $c$, the adjusted fullness increases rapidly and converges towards 60%. The observed running time increases linearly with $c$. The difference from the theoretical bound of $O(c^2)$ can be explained by the observation that an occupied place is detected on average in constant time since almost all tested

places are occupied. The choice $c = 100$ seems to be a good compromise between the quality and the speed.

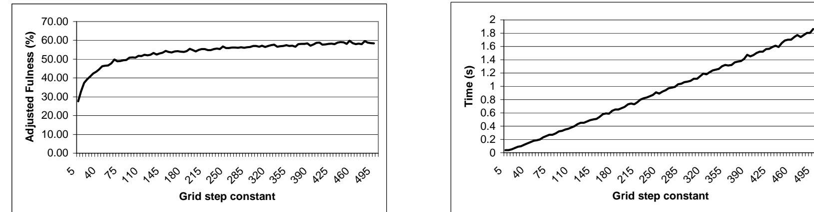


**Fig. 10.** How the grid step influences the adjusted fullness and running time.

The white space desired among the graph objects can be obtained by simply enlarging each object by half the spacing amount on each side.

The algorithm above does not favor one dimension over the other one and yields square-like drawings. In order to satisfy the desired aspect ratio DAR, one can simply take DAR as the unit grid step in $x$ direction and 1 as the unit step in the $y$ direction.

### 4.2 Packing in Multiple Pages

In certain applications, the graph is to be laid on multiple pages, and the task is to minimize the number of pages where the size of a page is defined in advance.

The approach is the same as above except the cost function is modified as $\max(x, y), x \geq 0, y \geq 0$, which defines the ordering starting from the corner of the page. Although such ordering lacks the nice central symmetry that we had in the original case, we have to modify the placement rule in order to better fill the sides of each page. We assume that each object separately fits in an empty page; otherwise an appropriate scaling should be performed. In algorithm PackPolyominoesInMultiPages we start by fitting the first polyomino on the first page. If the current polyomino does not fit in the current page, the next page is tried until it is successfully placed. Similar to the original algorithm, the best results are obtained when the objects are sorted in their decreasing sizes. An optimization can be achieved by considering only those grid cells on the current page for which the bounding rectangle of the current polyomino is completely inside the page.

Here is a pseudo code of the multi page placement algorithm:

**algorithm** PackPolyominoesInMultiPages($P_i, 1 \leq i \leq n$, $pageSize$)
(10) sort $P_i, 1 \leq i \leq n$ in the order of nonincreasing size
(11) initialize the grid $G$ for the first page using $pageSize$
(12) **foreach** polyomino $P_i$ **do**
(13) set $pageNo$ to 1
(14) **while** $P_i$ not placed **do**

(15) calculate $(x, y)$ such that the cost function is minimized
(16) **while** cannot place $P_i$ on page $pageNo$ centered at $(x, y)$
(17) and $(x, y)$ is within page boundaries **do**
(18) calculate next $(x, y)$ using the cost function
(19) **end while**
(20) **if** $P_i$ not placed **then**
(21) set $pageNo$ to the next one
(22) **if** $G$ not extended for page $pageNo$ **then**
(23) extend $G$ for page $pageNo$
(24) **end if**
(25) **end if**
(26) **end while**
(27) mark those cells in $G$ covered by $P_i$ as occupied
(28) **end foreach**

## 5 Comparison with Previous Methods

We have compared our new method with the tiling and alternate-bisection methods discussed earlier. During the experiments, graphs that contained up to a thousand disconnected objects were used. Each object was assumed to be a star polygon with random number of corners in $[3 \ldots 8]$, each with random integer coordinates in $[1 \ldots 100]$, all independent and uniformly distributed. The value for the approximation quality constant $c$ was taken to be 100. The desired aspect ratio was taken to be 1. For the previous methods the tightest rectangles bounding these polygons were used, whereas with our new approach, the smallest polyomino tightly bounding the polygons were used. Figure 11 shows a sample set of drawings produced by these methods for the same set of objects. The performance comparison of the methods is presented in Figure 12.

Clearly the new approach results in much more compact drawings. In terms of the execution time, it is slower but still easily within acceptable bounds given the fact that it is highly rare that a graph contains more than a few hundred disconnected objects.

## 6 Conclusion

In this paper, we reviewed existing algorithms and presented a new approach for layout of disconnected graphs. The new approach uses polyominoes as opposed to rectangles used in previous approaches for representation of isolated nodes and components, and produces much more compact and uniform drawings. The parameters of our algorithm and how they affect the drawings produced as well as a variation of the algorithm for multiple pages were discussed.

**Acknowledgement.** The authors wish to thank Cihad Baskoy for his help with the implementation and experimentation of certain algorithms.

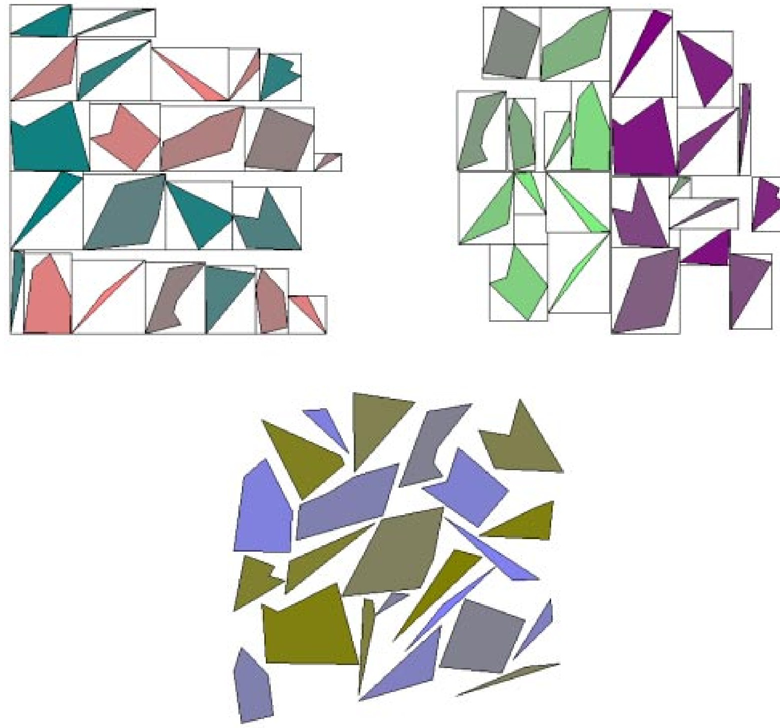

**Fig. 11.** A sample from the random set of objects laid out with all three methods: tiling (left), alternate-bisection (right), and polyomino (middle) packing.

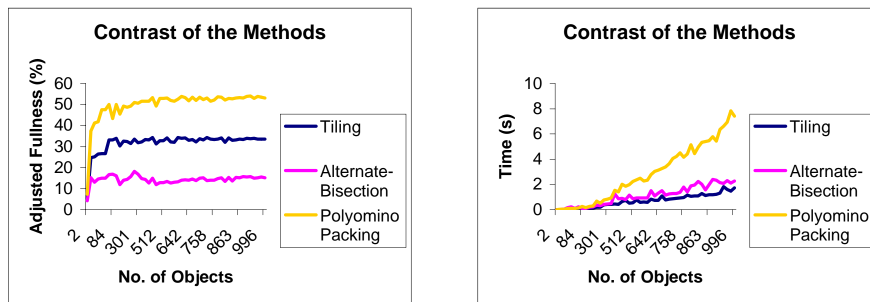


**Fig. 12.** Comparison of the new approach with the previous ones.

## References


1. B. S. Baker, E. G. Coffman, and R. S. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9(4):846–855, November 1980.
2. E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: An updated survey. In G. Ausiello, M. Lucertini, and P. Serafini, editors, *Algorithm Design for Computer System Design*, pages 49–106. Springer-Verlag, New York, 1984.
3. E. G. Coffman, M. R. Garey, D. S. Johnson, and R. E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808–826, November 1990.
4. E. G. Coffman and P. W. Shor. Packings in two dimensions: Asymptotic average-case analysis of algorithms. *Algorithmica*, 9:253–277, 1993.
5. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom. Theory Appl.*, 4:235–282, 1994.
6. U. Dogrusoz. Algorithms for layout of disconnected graphs. *Information Sciences*, to appear.
7. U. Dogrusoz, M. Doorley, Q. Feng, A. Frick, B. Madden, and G. Sander. Toolkits for development of software diagramming applications. *IEEE Computer Graphics and Applications*, to appear.
8. U. Dogrusoz and G. Sander. Graph visualization. *ACM Computing Surveys*, to appear.
9. M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-completeness*. Freeman, San Francisco, 1979.
10. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.
11. P. Kikusts and P. Rucevskis. Layout algorithms of graph-like diagrams of GRADE windows graphic editors. In F.J. Brandenburg, editor, *Graph Drawing (Proc. GD '95)*, volume 1027 of *Lecture Notes in Computer Science*, pages 361–364. Springer-Verlag, 1995.
12. T. Lengauer. *Combinatorial algorithms for integrated circuit layout*. John Wiley & Sons, 1990.