Vertex Splitting and Tension-Free Layout

P. Eades[1]* and C. F. X. de Mendonça N.[2]**

[1] Dept. of Computer Science
Univ. of Newcastle NSW Australia
[2] Depto. de Ciência da Comp.
UNICAMP SP Brasil

Abstract. In this paper we discuss the "vertex splitting" operation. We introduce a kind of "spring algorithm" which splits vertices to obtain better drawings. We relate some experience with the technique.

1 Introduction

Suppose that $G = (V, E)$ is a graph and $w : E \rightarrow R^+$ assigns a weight to each edge of G. Graph drawing functions may be required to draw G so that the Euclidean distance between two vertices is the same as the weight of the edge between them. For example a weighted graph model is proposed in [1, 9] to represent email relationships: if two users u_1 and u_2 exchange email frequently, then the weight on the edge (u_1, u_2) is small; otherwise the weight is large. To visualize these relationships, we draw the graph so that the Euclidean distance between two vertices represents the closeness of the relationship between two users. Further applications of weighted graph drawings are given [3, 4, 6].

The problem of drawing a weighted graph so that the Euclidean distance between vertices conforms to prespecified edge weights is NP-hard (see [7]). The heuristic approach of [1] uses a kind of "spring algorithm" [2] as a heuristic.

This paper takes a different approach to the problem. Suppose that vertex a is required to be close to vertex b, and b is required to be close to c, but a is required to be far from c. The triangle inequality makes this situation impossible to represent. The operation presented in this paper resolves this conflict by creating two copies of the vertex b, so that a may appear close to one copy of b and c may appear close to the other copy of b.

Intuitively, a vertex v may be "split" by making two copies v_1 and v_2 and attaching the edges incident with v to either v_1 or v_2. The operation is illustrated in Figure 1.

This simple operation is commonly used in manual layout: it has been observed in diagrams used in theorem proving systems, prerequisite diagrams for course structures, diagrammatic representations of metro systems and computer networks, and in circuit schematics. In each of these cases, a small number of vertices are split to change the graph a little to make it amenable to layout.

* Supported partially by a grant from LAC-FAPESP and by The Dept. of Comp. Sci. Univ. of Newcastle

** Supported partially by a grant from LAC-FAPESP and Proj. Integrado CNPq

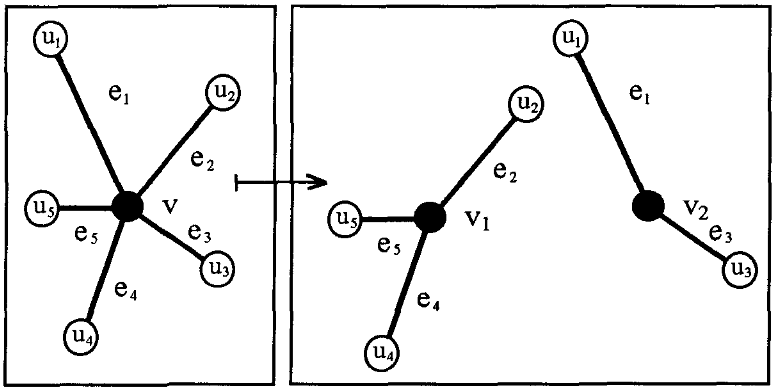

Fig. 1. The splitting operation

2 Preliminaries

A *graph* consists of a finite set V of vertices and a finite set E of edges, where each edge is an unordered pair of vertices of G. A vertex u is said to be *adjacent* to a vertex v if uv is an edge of G; in this case, the edge uv is said to be *incident* with u and v. The *neighbor* (or incident) set N_u of a vertex u is the set of all edges incident with u.

A *splitting* operation replaces v by two vertices v_1 and v_2, and partitions N_v into two nonempty disjoint sets N_{v_1} and N_{v_2}.

A *straight line drawing* of a graph $G = (V, E)$ is a function $D : V \to R^2$ that associates a position $D(v)$ to each vertex v of V. Since all drawings in this paper are straight line drawings we omit the term "straight line".

A *weight* $w(e)$ is a non-negative real value associated with an edge e of a graph. A *weighted graph* $G = (V, E, w)$ consists of a graph $G = (V, E)$ and a weight $w(e)$ for each $e \in E$.

Suppose that $G = (V, E, w)$ is a weighted graph. The *tension* in an edge $e = vu$ from a vertex v incident with e in a drawing D of G is a vector $\mathbf{t}_{ve}$ whose magnitude is the difference between the edge weight and the Euclidean distance between the two endpoint vertices, and whose direction is from $D(v)$ to $D(u)$. That is,

$$\mathbf{t}_{ve} = \frac{D(v) - D(u)}{d(v,u)}(d(v,u) - w(vu))$$

where $d(v, u)$ denotes the Euclidean distance between $D(u)$ and $D(v)$. Note that $\mathbf{t}_{ue} = -\mathbf{t}_{ve}$. A vertex v of G is *at equilibrium* if the sum of the tensions in the edges from v is zero, that is, if

$$\sum_{e \in N(v)} \mathbf{t}_{ve} = 0.$$

The drawing D is *at equilibrium* if every vertex is at equilibrium.

A drawing of a weighted graph G is said to be *tension-free* if $\mathbf{t}_{ve} = 0$ for all edges e of G. When a weighted graph admits a tension-free drawing we say that the weighted graph G is *tension-free*[3]. Every graph can be transformed into a set of nonadjacent edges by a sequence of vertex splitting operations, and in particular we can obtain a tension-free graph in this way.

Proposition 2.1 *A weighted graph $G = (V, E, w)$ can be transformed in a tension free weighted graph a sequence of vertex splitting operations.*

The problem of creating a tension free drawing of a graph is NP-hard. The proof uses a complex transformation from 3SAT; details may be found in [7]. In practice very few weighted graphs are tension-free. In this paper we consider splitting vertices to preprocess a graph to make its tension-free layout possible. Our approach is based on the spring system of Kamada and Kawai [4, 5].

3 The TensionSplit algorithm

First we review Kamada's spring algorithm [5]. The graph structure is simulated by a set of particles (for the vertices) in a plane where these particles are connected by springs (for edges and/or paths). An optimization method is applied to find a state of locally minimum energy of this system; this minimum energy state corresponds to the final drawing. The locally minimum energy state is found by repeating two steps: a *local minimization* step, and a *rearrange* step. The first step moves a vertex to a position where it is at equilibrium. In the second step some overlappings and crossings may be avoided by swapping vertices in pairs. In both steps the main goal is to reduce potential energy. The two steps are applied one after the other until the rearrange step does not change the layout.

Kamada's approach is be modified in two ways to form procedure TensionSplit in Figure 2. Our algorithm performs the local minimization step for *all* the vertices; and a modified rearrange step is used to perform splitting operations. Intuitively, the algorithm works in a very similar way to the Kamada algorithm, but when a vertex splits when it becomes "critical", that is, it has too much tension on it.

We now give details of the steps in procedure TensionSplit.

The first step partitions the edge set E into parts $E_1, E_2, \ldots, E_r$, where each E_i is the edge set of a biconnected component.

Step 2(a) is a implemented the the same way as the Kamada algorithm.

For step 2(b) we need to define some terms. For each vector $\mathbf{t}_{v,e}$ we define a *split line* which consists of a straight line through v perpendicular to the direction of the vector. Figure 3 displays a split line for the vector $\mathbf{t}_{v,e}$ (in this example, all edge weights are zero). This split line divides the plane into two semi-planes. The point where the vertex v is located divides the split line into two semi-lines.

[3] Our choice of terminology here is slightly nonstandard but it is motivated by the algorithm which follows.

```
procedure TensionSplit;
1. divide the graph into biconnected components;
2. repeat until there is no critical vertex:
   (a) run the local minimization on all vertices until the drawing is at
       equilibrium;
   (b) if a critical vertex exists then
       i. choose a critical vertex $v$ for which the the tension $T_v$ on $v$ is
          greatest;
       ii. split $v$ into $v_1$ and $v_2$;
       iii. partition $N_v$ into $N_{v_1} = N_v \cap R_{v,e}$ and $N_{v_2} = N_v \cap L_{v,e}$;
       iv. run the local minimization step on vertices $v_1$ and $v_2$;
   (c) run the rearrange step for all vertices of G;
```

Fig. 2. The Tension Split algorithm

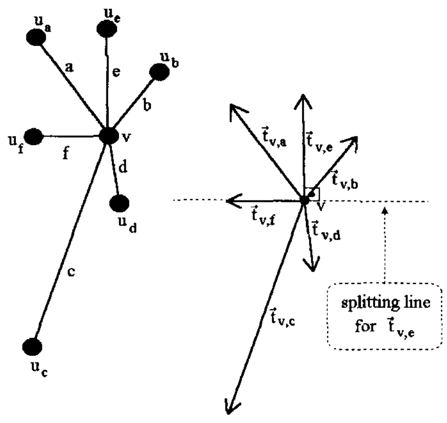

Fig. 3. A sample of a split line

Let R denote the region of the plane defined by one semi-plane and one semi-line, and let L denote the complement of R. Figure 4 displays the two regions. We define a partition of N_v into edge sets $R_{v,e}$ and $L_{v,e}$ as follows. An edge $e = vw$ is in $R_{v,e}$ if w lies in R; and e is in $L_{v,e}$ if w lies in L. Figure 5 displays the partitions $R_{v,e}$ and $L_{v,e}$ for the graph in Figure 3.

A split line is *valid* if there is at least one biconnected component E_i for which both $E_i \cap R_{v,e}$ and $E_i \cap L_{v,e}$ are nonempty. Figure 6 displays: (a) a valid and an invalid split line for $\mathbf{t}_{v,f}$, and (b) an invalid split line for $\mathbf{t}_{v,e}$.

For each valid split line $\mathbf{t}_{v,e}$ we define the *tension for this split line* as

$$\mathbf{T}_{v,e} = \sum_{f \in R_{v,e}} \mathbf{t}_{v,f}.$$

Since the graph drawing is at equilibrium after step 2(a), we could replace $R_{v,e}$ by $L_{v,e}$ in the definition of $\mathbf{T}_{v,e}$ above. The *tension T_v in vertex v* is the maximum value of $|\mathbf{T}_{v,e}|$ over all valid split lines. Our algorithm uses a constant τ corresponding to the value of the minimum permissible tension. This constant can be explicitly changed by the user. A vertex is *critical* if $T_v > \tau$.

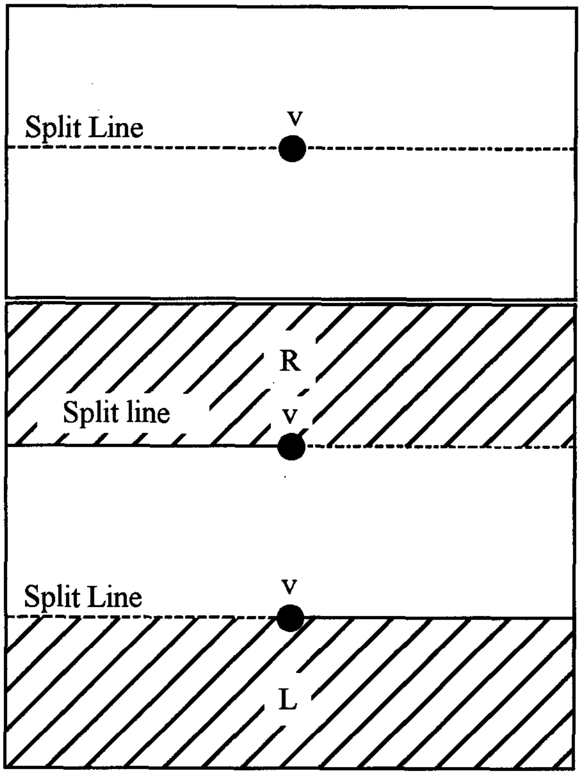

Fig. 4. The two regions define by a split line

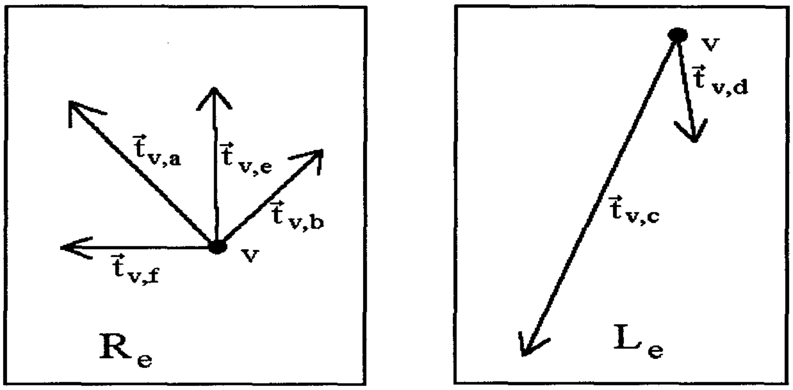

Fig. 5. The partition of the vectors into two sets

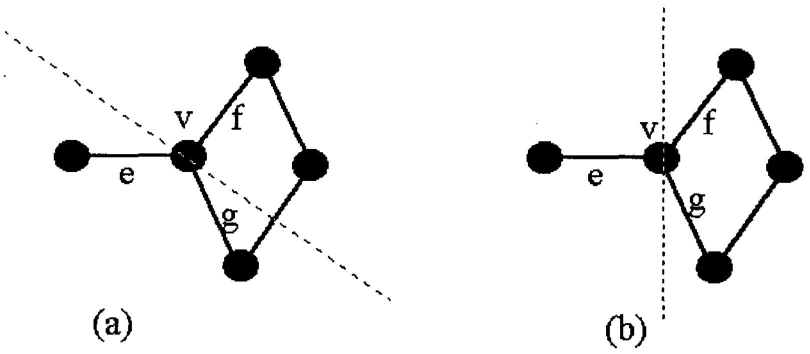

Fig. 6. A valid and an invalid split-lines

4 Samples

Some typical samples of splitting operations performed by the TensionSplit algorithm are described below.

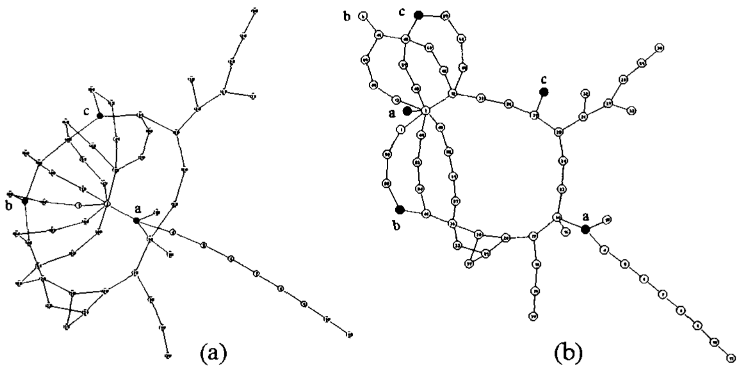

Fig. 7. example 1

Figures 7 and 8, are graphs for which spring algorithms perform poorly. Figure 7(a) displays the layout without splittings. Figure 7(b) displays the layout with three splittings (indicated by the letters *a*, *b* and *c*). Figure 8(a) displays the layout without splittings. Figure 8(b) and (c) display an intermediate stage layouts of the Tension Split algorithm with one splitting (indicated by the letter *a*) and two critical vertices (indicated by *b* and *c*). Figure 8(d) displays the final drawing with very little tension.

The diagram in Figure 9(a) is a schema from a commercial database design. It is drawn by a standard spring technique. Some node overlaps and crossings

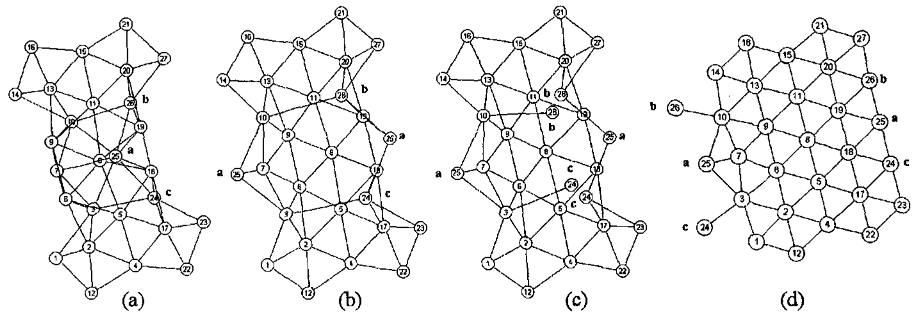

Fig. 8. example 2

appear. Algorithm TensionSplit gives better diagram in Figure 9(b) with only one vertex split (indicated by a surrounding ellipse). This sample shows a very good application of TensionSplit algorithm.

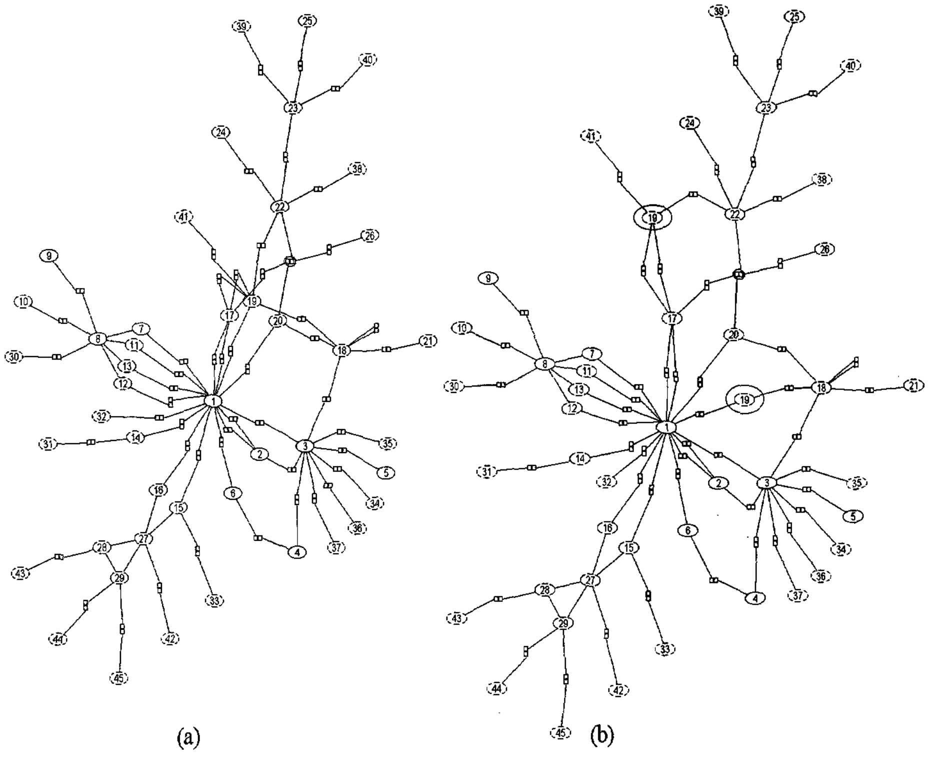

Fig. 9. example 3

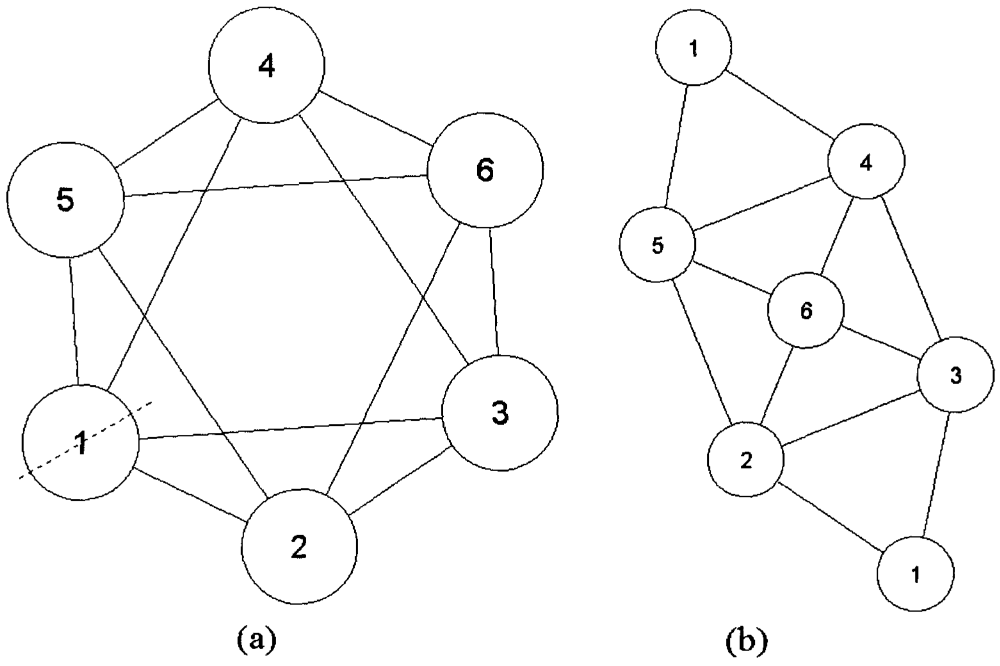

Fig. 10. example 4

Figures 10, 11 and 12 are general graphs. For each figure, a relatively small value of critical tension was used. The splitting operations are displayed. For all layouts the algorithm gives considerable symmetry. the resulting drawings are attractive: the symmetry is not bad and the edge length tends to 1.

5 Conclusion

The TensionSplit algorithm produces a layout which is near tension- free and symmetric. However, it has some problems which are inherited from Spring System [2]: it is relatively slow, and does not resolve edge crossings. For a more comprehensive approach to vertex splitting which takes edge crossings into account, see [8].

References

1. P. Eades, W. Lai, and X. Mendonça. A Visualizer for E-mail Trafic. In *4th Int. Conf. Proc. Pacific Graphics'94 / CADDM'94*, pages 64–67, 1994.
2. P. D. Eades. A Heuristic for Graph Drawing. *Congr. Numer.*, 42:149–160, 1984.
3. T. Kamada. *Visualizing Abstract Objects and Relations.* World Scientific, 1989.

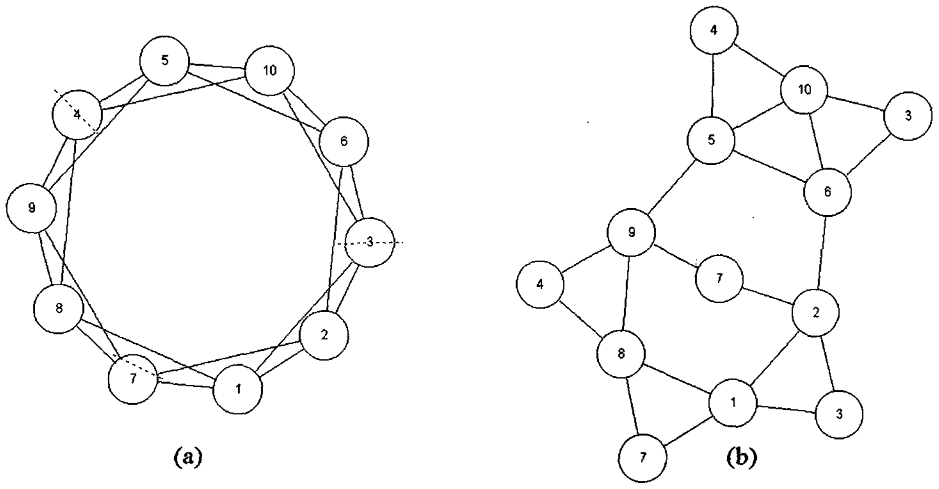

Fig. 11. example 5

4. T. Kamada and S. Kawai. Automatic Display of Network Structures for Human Understanding. Technical Report 88-007, Department of Information Science Faculty of Science, University of Tokyo, Tokyo, 1988.
5. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.
6. X. Lin. *Analysis of Algorithms for Drawing Graphs.* PhD thesis, University of Queensland, Department of Computer Science, University of Queensland, 1992.
7. C. F. X. Mendonça. A Layout System for Information System Diagrams. Technical Report 94-01, Department of Computer Science, University of Newcastle, Australia, April 1994.
8. C. F. X. Mendonça and T. A. Halpin. Automatic Display of NIAM Conceptual Schemas Diagrams. Technical Report 209, Department of Computer Science, University of Queensland, Australia, July 1991.
9. Wei Lai. *Building Iteractive Diagram Applications.* PhD thesis, University of Newcastle, 1993.

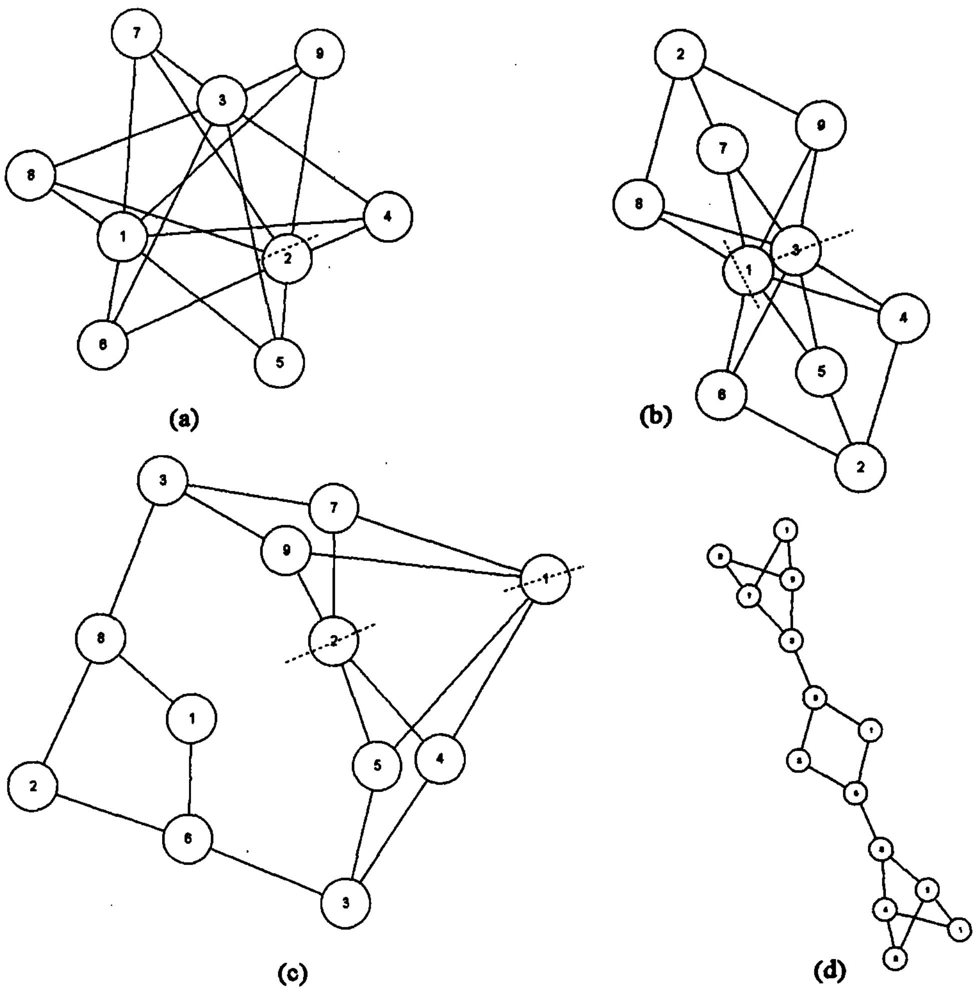

Fig. 12. example 6