NP-Hardness Results for Tension-Free Layout

C. F. X. de Mendonça N. *P. Eades*
C. L. Lucchesi[*†] *J. Meidanis*[*†]

Relatório Técnico DCC–95-15

Agosto de 1995

NP-Hardness Results for Tension-Free Layout

C. F. X. de Mendonça N.[*†] P. Eades[‡§]
C. L. Lucchesi[*†] J. Meidanis[*†]

Abstract

A *tension-free layout* of a weighted graph G is an embedding of G in the plane such that the Euclidean distance between adjacent nodes is equal to the edge weight. Very few weighted graphs admit such a layout. However, any graph can be made into a tension-free graph by repeated application of an operation called *vertex splitting*, or by removing edges. In this paper we show that computing the minimum number of such operations that yield a tension-free graph is NP-hard.

1 Introduction

A *tension-free layout* of a weighted graph G is an embedding of G in the plane such that the Euclidean distance between adjacent nodes is equal to the edge weight.

Tension-free layouts of graphs play an important role in several visualization problems. For example, in the problem of visualization of email traffic [8, 3] we are required to draw a graph of email connections on the screen, with the Euclidean distance between two nodes being proportional to the amount of email traffic between the nodes. Many other applications of weighted embeddings are given in the literature [4, 7, 6, 9].

Of course, for most weighted graphs, a tension-free layout is impossible. In general, the constraints imposed on the position of a node by all the neighbors of this node are too many to be met simultaneously.

One way to overcome this problem is by considering *vertex splitting* operations. Intuitively, a vertex v may be "split" by making two copies v_1 and v_2 and attaching each edge incident with v to either v_1 or v_2, but not to both. The operation is illustrated in Figure 1. The problem becomes easier because v_1 and v_2 have less constraints to satisfy than the original v.

In this paper we show that finding the minimum number of vertex splitting operations to give a tension-free layout is NP-hard. Eades and Mendonça [2] give a heuristic approach based on the Spring System discussed by Kamada [7]. A slight modification of our proof gives the same NP-hardness result when edge removal rather than vertex splitting operations are considered.

[*]Supported partially by a grant from LAC-FAPESP and Proj. Integrado CNPq
[†]Depto. de Ciência da Comp., UNICAMP SP Brasil
[‡]Supported partially by The Dept. of Comp. Sci. Univ. of Newcastle
[§]Dept. of Computer Science, Univ. of Newcastle NSW Australia

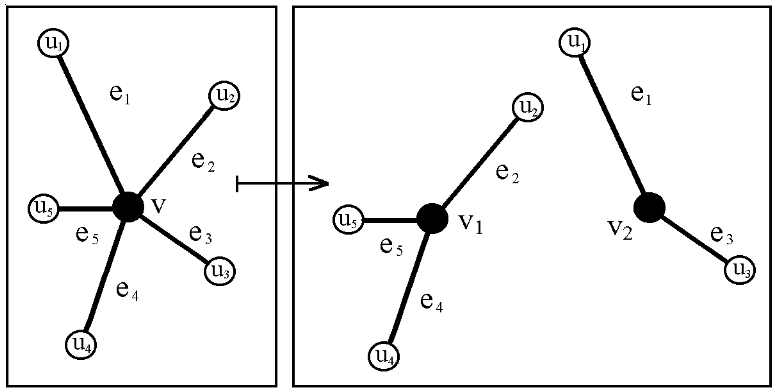

Figure 1: The splitting operation

The rest of this paper is organized as follows. In Section 2 we define the basic terms used throughout. Section 3 presents our main results. Finally, Section 4 contains our conclusions and plans for future work.

2 Terminology and Basic Results

2.1 Graphs

In this section we review basic facts on graphs and fix terminology. Detailed definitions can be found in any standard text, e.g., the book by Bondy and Murty [1].

A *graph* G consists of a set V_G of *vertices* and a set E_G of *edges*, where each edge of G is an unordered pair of distinct vertices of G. The reader may note that this definition is the same as the definition of *simple graph* for some authors. A graph G is said to be *finite* if V_G is finite. When the graph is understood from the context, we write only V and E instead of V_G and E_G.

Let $e = \{u, v\}$ be an edge of G. For simplicity we denote such edge not only by e but also by uv or vu. The edge e is said to be *incident* with u and v; vertices u and v are called the *endpoints* of e; vertex u is said to be *adjacent* to v and vice-versa; and u and v are *neighbors*. For a given vertex x we define the *adjacent vertex set of* x, denoted by $Adj(x)$, as the set of all adjacent vertices of x in G. For a given vertex x we define the *neighborhood* of x, denoted by N_x, which consists of the set of all edges incident with x. Let $d(x)$ denote the size of the neighborhood of x.

A graph $H = (U, F)$ is a *subgraph* of a graph $G = (V, E)$ if U is a subset of V and F is a subset of E.

A *path* between vertices u and v in a graph G is a sequence $(u = v_1, v_1v_2, v_2, ..., v_k = v)$ of alternating vertices and edges, with no repeated vertices, that is, for any pair of vertices v_i and v_j in the path we have $v_i \neq v_j$ if $i \neq j$. Each edge in the path is incident with the vertices preceding and following it in the path. Since the edges are well defined by their end points, we may denote a path simply as a sequence of vertices $(u = v_1, v_2, ..., v_k = v)$ where each vertex v_i is adjacent to v_{i-1} and v_{i+1} (except u and v of course). The *length* of a path is defined as the number of edges in the path. The *distance* between two vertices u

and v is defined as the length of the shortest path between u and v. To avoid any conflict between this graph-theoretic definition of distance and the definition of geometrical distance between two points, we adopt the convention that the distance between two points in the plane is called *Euclidean distance* (square root of the sum of the squares of the differences of x coordinates and y coordinates).

2.2 Graph Drawings

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

A *weight assignment* for a graph $G = (V, E)$ is a function $w : E \to R^+$. A weighted graph $G = (V, E, w)$ consists of a graph $G = (V, E)$ and a weight assignment w for G. The *weight* of an edge $e \in E$ is the non-negative real value $w(e)$.

The *tension* in an edge in a drawing of a weighted graph is defined as the difference between the edge weight and the Euclidean distance between the two endpoint vertices. A drawing of a weighted graph G is said to be *tension-free* if the tension is 0 for all edges of G. When a weighted graph $G = (V, E, w)$ admits a tension-free drawing we say that w is a *valid* weight assignment.

A *splitting* operation on a vertex v is a partition of the neighborhood of v into $k \geq 2$ non empty subsets of edges, followed by replacement of v by k new vertices, one for each subset in the partition. The new vertices will have these sets as their neighborhoods. Hence, the number of edges remains the same under a splitting operation, but the number of nodes grows by $k - 1$. Also, if the original graph is weighted, the edge weights remain the same. If $k = 2$ we have a *binary* splitting operation. Since in our work we consider only binary splitting operations, we omit the word binary throughout this paper.

Proposition 2.1 *A graph $G = (V, E)$ can be transformed into:*

- *a planar graph, or*
- *a forest, or*
- *a bipartite graph*

by a sequence of splitting operations.

Proof: We perform splitting operations on all vertices with degree bigger than 1 until we get a graph where all vertices have degree 1. This graph belongs to all the above classes. □

Corollary 2.2 *A weighted graph $G = (V, E, w)$ can be transformed in a graph with a valid weight assignment by a sequence of splitting operations.* □

3 Complexity of the SPLIT-TENSION-FREE GRAPH problem

In this Section we show that computing the minimum set of splitting operations to validate a weight assignment is NP-Hard.

SPLIT-TENSION-FREE GRAPH
Instance: *Graph G, positive integer number $K \leq |E|$, weight assignment w for G.*
Question: *Is there a sequence of K or less splitting operations that yields a graph $G' = (V', E)$ for which w is a valid weight assignment?*

Theorem 3.1 *The SPLIT-TENSION-FREE GRAPH problem is NP-hard.*

To prove this theorem we must make some definitions and state two lemmas.

A *circuit* is a connected graph where all vertices have degree 2. The number of vertices of the circuit is called *size*. A circuit of size 3 is called *triangle*.

The *perimeter* of a circuit in a weighted circuit is the sum of the weights of its edges.

Lemma 3.2 *If a weight assignment for a circuit $C = (V, E)$ of size $n > 2$ is a valid assignment then no edge weight exceeds half of the perimeter.*

Proof: Let C be a circuit of G, e one of its edges and L a valid weight assignment with perimeter p. Let D be a tension-free layout of C. By definition of tension-free layout, the Euclidian distance between the endpoints of e is precisely $w(e)$. Furthermore, that Euclidian distance does not exceed the sum of the weights of the edges in C distinct from e. Therefore,

$$p = \sum_{a \in E} w(a) = w(e) + \sum_{\substack{a \in E \\ a \neq e}} w(a) \geq 2w(e).$$

□

Lemma 3.3 *Two different tension-free drawings for the same valid weight assignment of a triangle are isometric.*

Proof: Let T be a triangle with a valid weight assignment w, and let v, u, and x be the nodes of T. Given two tension-free drawings of T, we can always translate one of them so that the images of v coincide in a point v'. After doing that, we can now apply a rotation around v' and make the images of u coincide in a point u'. Note that such a rotation leaves v' fixed. At this stage either the images of w coincide or they are symmetric with respect to the line $v'u'$. But a reflection with respect to a line is also an isometry, so in all cases one drawing can be obtained from the other by composing with a plane isometry. □

Proof: (of Theorem 3.1) We reduce 3SAT to the SPLIT-TENSION-FREE GRAPH problem.

3-SATISFIABILITY (3SAT)
Instance: *Set U of variables, collection C of clauses over U such that each clause $c \in C$ has $|c| = 3$.*
Question: *Is there a satisfying truth assignment for C?*
Reference: Garey and Johnson [5] problem [LO2] page 259.

Let U be a set of variables and C be a collection of clauses over U such that each clause $c \in C$ has $|c| = 3$. We shall construct a weighted graph $G = (V, E, w)$ such that a satisfying truth assignment exists for C if and only if there is a sequence of $K = |U|$ splitting operations in G that yields a weighted graph $G' = (V', E, w)$ for which w is a valid weight assignment.

Before describing the graph G, let us introduce certain elements that appear very frequently in G. One of these elements is what we call an *overlapped vertex*. This is not really a vertex but rather a path $v = (v_1, v_2, ..., v_{K+1})$ of $K + 1$ vertices where all the edges have weight 0. So, although an overlapped vertex is actually composed of several vertices, in any tension-free drawing they must be drawn in the same coordinates. When depicting such a drawing, an overlapped vertex will be denoted by a round vertex. We say that vertex v_i is the vertex at *layer* i of v.

To distinguish them from overlapped vertices, ordinary vertices in G will be called *split vertices*. This name is meaningful because these vertices are good candidates to split if we don't have a tension-free layout but want one. Overlapped vertices are not good candidates to split because even if we split K of them there will still be one left to make the assignment invalid. Remember that we are allowed at most K splitting operations. In a drawing, split vertices are denoted by a square with rounded corners.

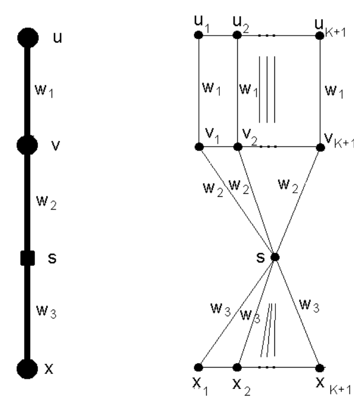

Figure 2: A drawing of overlapped graph

An edge connecting two split vertices is just a regular edge in G. In contrast, edges connecting at least one overlapped vertex are called *overlapped edges*. An overlapped edge connecting two overlapped vertices u and v means that each vertex u_i is adjacent to a vertex v_i in the same layer. An overlapped edge connecting an overlapped vertex u to a split vertex s means that the vertex s is adjacent to all vertices u_1, u_2, ...,u_{K+1}. When a weight is assigned to an overlapped edge e, the same assignment is given to all $K + 1$ edges of e. In a drawing, an overlapped edge is denoted by a thick line.

If a graph is composed of overlapped vertices, split-vertices and overlapped edges it is called *overlapped graph*. These concepts are illustrated in Figure 2.

We are now ready to describe the graph G constructed from an instance of 3SAT. The graph G has three parts:

pillar a constant graph for "holding" the remainder of G,

flippers one for each variable in U, and

swings one for each clause in C.

The *pillar* consists of three overlapped triangles $\{r,t,x\}$, $\{v,\overline{v},x\}$ and $\{v,\overline{v},t\}$ with the vertex x in common between the first and second triangles and the edge $v\overline{v}$ between the second and the third triangles. More precisely, the pillar consists of five overlapped vertices, v, x, $\overline{v}$, r, and t and the following overlapped edges with their weights:

$$\begin{aligned} w(v\overline{v}) &= 18, \\ w(vx) &= 15, \\ w(x\overline{v}) &= 15, \\ w(vt) &= 9, \\ w(\overline{v}t) &= 9, \\ w(xr) &= 113, \\ w(tr) &= 101, \\ w(xt) &= 12. \end{aligned}$$

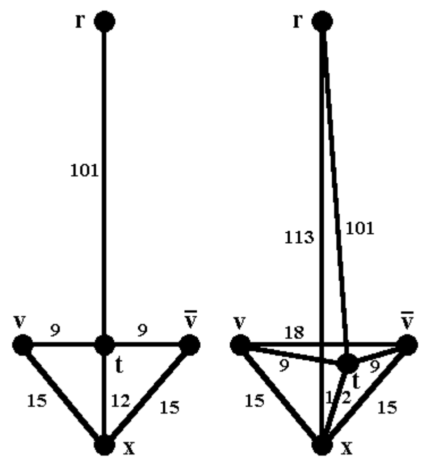

Figure 3: The pillar

Figure 3 displays a tension-free drawing of the pillar. Vertex t must overlap the edges $v\overline{v}$ and rx to make the drawing tension-free (the triangles $\overline{v}tv$ and xtr must have area 0). It is easy to see that the weight assignment for the pillar is valid (a tension free layout may be $(-9,12),(0,0),(9,12),(0,12),(0,113)$ for v, x, $\overline{v}$, t, and r respectively).

For each variable $u \in U$ we build a gadget called *flipper* which consists of four vertices, namely, a pair of splitting vertices s and $\overline{s}$ adjacent to u and $\overline{u}$, respectively. The flipper is connected to the pillar by two overlapped edges: sv, and $\overline{s}\overline{v}$. More precisely, we have two overlapped vertices u and $\overline{u}$, two split vertices s and $\overline{s}$, and the following overlapped edges:

$$\begin{aligned} w(u\overline{u}) &= 15, \\ w(su) &= 0.5, \\ w(\overline{s}\overline{u}) &= 0.5, \\ w(sv) &= 0.5, \\ w(\overline{s}\overline{v}) &= 0.5. \end{aligned}$$

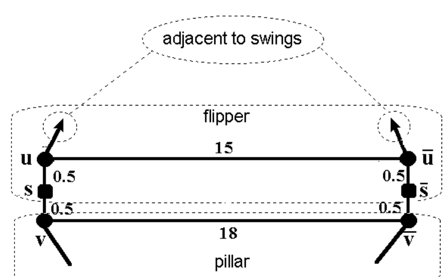

Figure 4: The flipper

Figure 4 displays a drawing of the flipper associated with a literal and the weight assignment of each edge. The flippers associated with the different variables overlap each other, however they are not bound to each other, except by the pillar, and therefore they are independent. It is easy to see that the weight assignment shown in Figure 4 in the combination flipper attached to the pillar is not valid, by Lemma 3.2. Note that the circuit $\{v, \overline{v}, \overline{s}, \overline{u}, u, s\}$ has perimeter 35 and one of the edges $v\overline{v}$ has weight 18. However, one splitting operation in s or $\overline{s}$ makes the weight assignment valid.

For each clause $c \in C$, $c = l_1 \vee l_2 \vee l_3$ with literals l_1, l_2, l_3 in U, we build a gadget called *swing*. Without loss of generality let us suppose that the literals l_1, l_2 and l_3 are respectively u_1, $\overline{u}_2$ and $\overline{u}_3$. The corresponding swing consists of two overlapped triangles $\{l_1, l_2, y\}$ and $\{y, l_3, r\}$. Note that the vertex r is the vertex on the "top" of the pillar. The weight assignment for this swing is as follows:

$$
\begin{aligned}
w(l_1 l_2) &= 16, \\
w(l_1 y) &= 8, \\
w(l_2 y) &= 8, \\
w(r y) &= 4, \\
w(r l_3) &= 4, \\
w(y l_3) &= 8.
\end{aligned}
$$

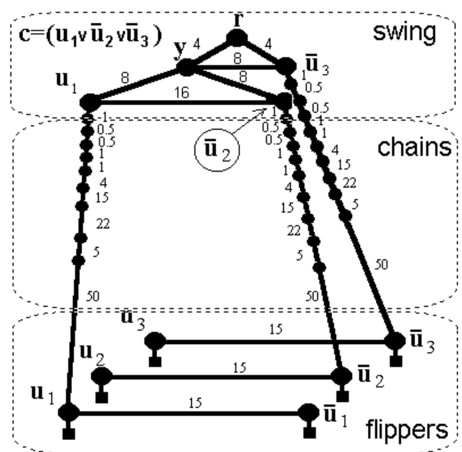

Figure 5: The swing

Figure 5 displays one of the clauses. The swings overlap each other, but they are bound to each other only by the common vertex r. To have a tension free layout the two triangles $\{l_1, l_2, y\}$ and $\{y, l_3, r\}$ must have area 0. Finally, each swing will be connected to three literals (which correspond to the literals that appear in the clause) by an overlapped path containing ten overlapped edges and nine overlapped vertices ($c_0 = l_1|l_2|l_3$,) c_1, c_2, ..., c_9, ($c_{10} = u_i|\overline{u}_i$). We call this path *chain*. The weight assignment for the chains is as follows:

$$
\begin{aligned}
w(c_0 c_1) &= 1, \\
w(c_1 c_2) &= 0.5, \\
w(c_2 c_3) &= 0.5, \\
w(c_3 c_4) &= 1, \\
w(c_4 c_5) &= 1, \\
w(c_5 c_6) &= 4, \\
w(c_6 c_7) &= 15, \\
w(c_7 c_8) &= 22, \\
w(c_8 c_9) &= 5, \\
w(c_9 c_{10}) &= 50.
\end{aligned}
$$

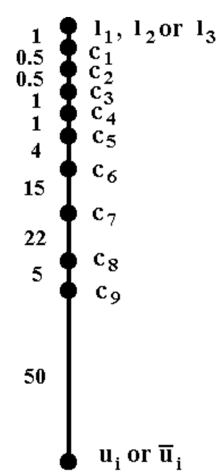

Figure 6: The chains

Figure 6 displays a chain and its weight assignment.

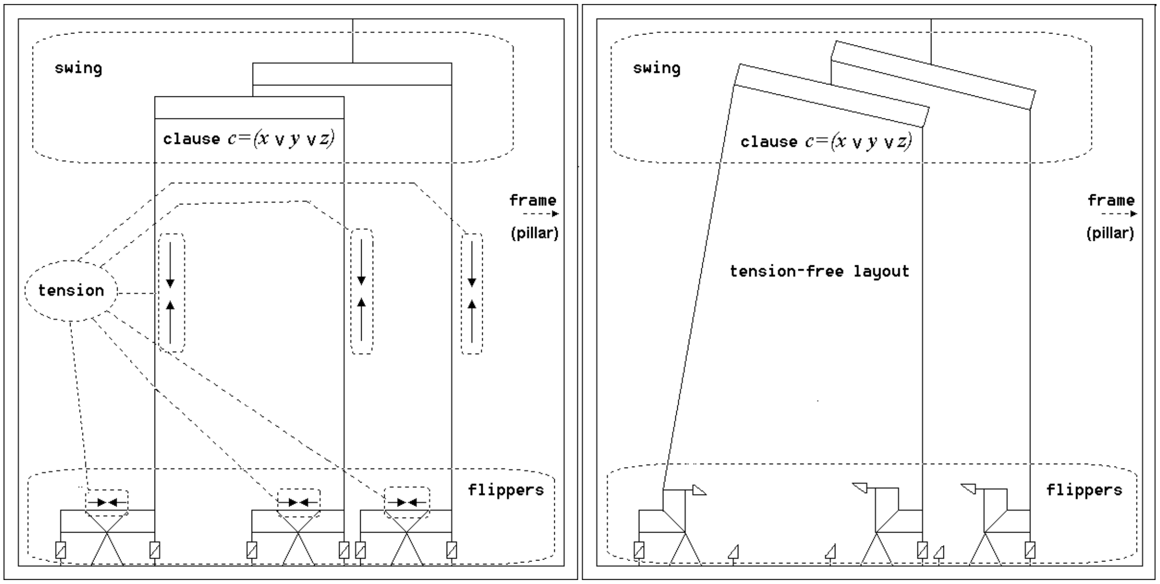

Figure 7: An instance for a single clause

The combination of the gadgets pillar, flippers and swings forming the weighted graph G does not have a tension free layout. However, if one of the flippers connected to each swing is "broken" the new graph admits a tension free layout. Figure 7 displays a layout in which the tension appears in the flippers and chains and a tension-free layout after a splitting operation.

This collection of flippers, swings and chains in a pillar forms an instance of the SPLIT-TENSION-FREE GRAPH problem. It is constructed in polynomial time $(a_1 + a_2K + a_3K|C| + a_4K^2)$. We claim that there is a satisfying truth assignment for C if and only if there is a sequence of $K = |U|$ splitting operations on the weighted graph $G = (V, E, w)$ that yields a weighted graph $G' = (V', E, w)$ (with the same assignment w of G) such that w is valid.

Suppose we have a satisfying truth assignment for C. If a variable u_i is true, then we split the split-vertex s_i in the flipper corresponding to u_i; if u_i is false, then we split $\overline{s}_i$. These split operations give a graph G'. Thus each flipper has been "broken" at either s or $\overline{s}$. Since each clause c has at least one true literal, the swing-flipper combination is "broken" at the split-vertex of the flipper on the side connected by a chain to the swing correspondent to the clause c. This is represented schematically in Figure 8 for a clause $c = (x \vee y \vee z)$

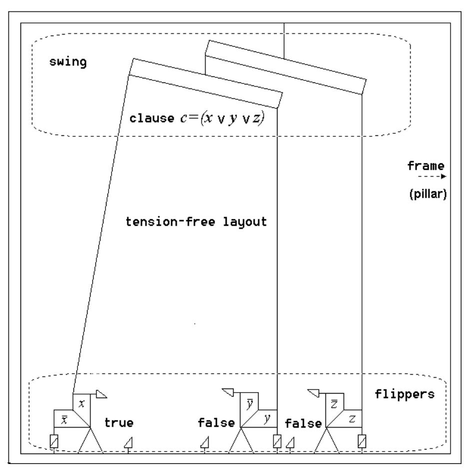

Figure 8: A tension free layout of a combination swing-chain-flippers

with literal x true.

The flipper which is “broken” on the side connected to the swing allows the swing to “tip-over” and “releases” the tension for the clause.

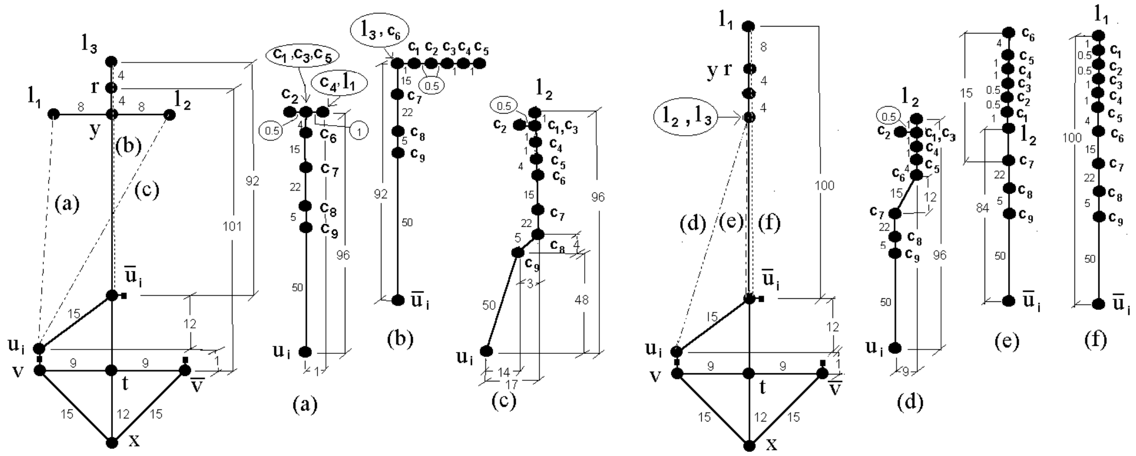

Figure 9: The two possible releases of the tension by a splitting operation

The weight assignment for G' is valid. The pillar and swings (in isolation) are always valid, each flipper is “broken” and thus valid. For each swing, the tension in at least one chain is released by the breaking of one of it’s literals counterpart split-vertices, and thus the swing may tip over to release the tension on the other two chains, as in Figure 9. Furthermore, the chains connecting the flippers and the swings may assume one of the six configuration shown in Figure 9 (a), (b), (c), (d), (e) or (f). Note that, if two or three literals are true in a single clause we choose the second configuration shown in Figure 9. Therefore, the graph G can be embedded in the plane such that each vertex coordinates are a multiple of 0.5.

Figure 10 displays the case for the clause $c = (u_1 \vee overlineu_2 \vee \overline{u}_3)$ where u_1 and u_3 are false and u_2 is true. The layout of the three chain connecting l_1 to u_1, l_2 to $\overline{u}_2$ and l_3 to $\overline{u}_3$ are shown in Figure 10 (a), (b) and (c), respectively.

Conversely, let K be a set of splitting operations in the weighted graph $G = (V, E, w)$ which yields a graph $G' = (V', E, w)$ such as w a valid weight assignment. Since each flipper must be “broken” to give a valid weight assignment, and there are $K = |U|$ flippers, each flipper must be broken by splitting exactly one vertex. Such a split can only occurs at one of the split-vertices s or $\overline{s}$. If s is split, we assign the

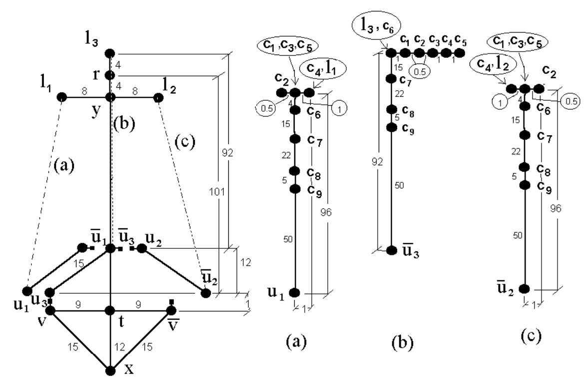

Figure 10: a tension-free layout for $c = (u_1 \vee \overline{u}_2 \vee \overline{u}_3)$

corresponding variable to be true; if $\overline{s}$ is split, we assign the corresponding variable false. This is a satisfying truth assignment for C. Since there are already K splitting operation in the flippers, no further vertices were split. To release the tension of a swing-chain combination, at least one of the flippers connected to the swing must be "broken" on the same side as the chain connecting to the swing. Thus a literal in each clause is true.

□

The previous proofs have implications in subgraph embeddings as follows.

TENSION-FREE WEIGHTED SUBGRAPH
Instance: *Graph G, positive integer number $K < |E|$, weight assignment w.*
Question: *Is there a subset $E' \subseteq E$ with $|E - E'| \leq K$ such that the weight assignment applied to the edges remaining in $G' = (V, E')$ is a valid assignment?*

Corollary 3.4 *The TENSION-FREE WEIGHTED SUBGRAPH is a NP-hard problem.*

Proof: Reduce 3SAT to TENSION-FREE WEIGHTED SUBGRAPH. Given an instance of 3SAT, build a graph in the same way as in the proof of Theorem 3.1 with a slightly different flipper. The new flipper has the vertices u_i and $\overline{u}_i$ as single vertices which are adjacent by single edges to the vertices s_i and $\overline{s}_i$ (see Figure 11). □

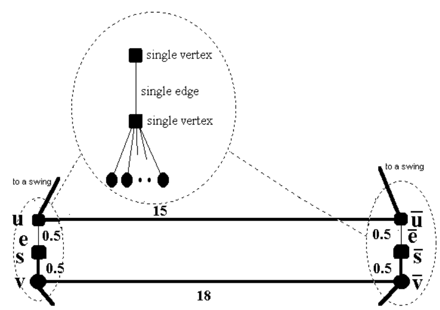

Figure 11: the new flipper

4 Conclusions

We have shown that computing the minimum number of certain operations that transform a given weighted graph into one that can be embedded in the plane in a tension-free way is NP-hard. The operations considered are vertex splitting and edge removal.

These results suggest that exact algorithms for the problems are probably exponential, and that it makes sense to look for alternative algorithms. For the vertex splitting operation, heuristic methods have been proposed [2]. It would be interesting to come up with similar methods for the case of edge removal.

We are currently studying a version of the problem where nodes are placed in points of the form $(i\epsilon, j\epsilon)$, with i and j integer and ϵ a fixed real number. In this case, we seek layouts with tension at most ϵ in every edge.

References

[1] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications.* American Elsevier Publishing Co., Inc., 1976.

[2] P. Eades and C. F. X. N. de Mendonça. Vertex Splitting and Tension-Free layout. to appear in Information Processing Letters, 1996.

[3] 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.

[4] P. D. Eades. A heuristic for graph drawing. *Congr. Numer.*, 42:149–160, 1984.

[5] M. R. Garey and Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* CA:Freeman, San Francisco, 1979.

[6] T. Kamada. *Visualizing Abstract Objects and Relations.* World Scientific, 1989.

[7] 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.

[8] Wei Lai. *Icon and Dion Applications.* PhD thesis, University of Newcastle, 1993.

[9] X. Lin. *Analysis of Algorithms for Drawing Graphs.* PhD thesis, University of Queensland, Department of Computer Science, University of Queensland, 1992.

Relatórios Técnicos – 1992

92-01 **Applications of Finite Automata Representing Large Vocabularies,** *C. L. Lucchesi, T. Kowaltowski*

92-02 **Point Set Pattern Matching in d-Dimensions,** *P. J. de Rezende, D. T. Lee*

92-03 **On the Irrelevance of Edge Orientations on the Acyclic Directed Two Disjoint Paths Problem,** *C. L. Lucchesi, M. C. M. T. Giglio*

92-04 **A Note on Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams,** *W. Jacometti*

92-05 **An (l, u)-Transversal Theorem for Bipartite Graphs,** *C. L. Lucchesi, D. H. Younger*

92-06 **Implementing Integrity Control in Active Databases,** *C. B. Medeiros, M. J. Andrade*

92-07 **New Experimental Results For Bipartite Matching,** *J. C. Setubal*

92-08 **Maintaining Integrity Constraints across Versions in a Database,** *C. B. Medeiros, G. Jomier, W. Cellary*

92-09 **On Clique-Complete Graphs,** *C. L. Lucchesi, C. P. Mello, J. L. Szwarcfiter*

92-10 **Examples of Informal but Rigorous Correctness Proofs for Tree Traversing Algorithms,** *T. Kowaltowski*

92-11 **Debugging Aids for Statechart-Based Systems,** *V. G. S. Elias, H. Liesenberg*

92-12 **Browsing and Querying in Object-Oriented Databases,** *J. L. de Oliveira, R. de O. Anido*

Relatórios Técnicos – 1993

93-01 **Transforming Statecharts into Reactive Systems,** *Antonio G. Figueiredo Filho, Hans K. E. Liesenberg*

93-02 **The Hierarchical Ring Protocol: An Efficient Scheme for Reading Replicated Data,** *Nabor das C. Mendonça, Ricardo de O. Anido*

93-03 **Matching Algorithms for Bipartite Graphs,** *Herbert A. Baier Saip, Cláudio L. Lucchesi*

93-04 **A lexBFS Algorithm for Proper Interval Graph Recognition,** *Celina M. H. de Figueiredo, João Meidanis, Célia P. de Mello*

93-05 **Sistema Gerenciador de Processamento Cooperativo,** *Ivonne. M. Carrazana, Nelson. C. Machado, Célio. C. Guimarães*

93-06 **Implementação de um Banco de Dados Relacional Dotado de uma Interface Cooperativa,** *Nascif A. Abousalh Neto, Ariadne M. B. R. Carvalho*

93-07 **Estadogramas no Desenvolvimento de Interfaces,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

93-08 **Introspection and Projection in Reasoning about Other Agents,** *Jacques Wainer*

93-09 **Codificação de Seqüências de Imagens com Quantização Vetorial,** *Carlos Antonio Reinaldo Costa, Paulo Lício de Geus*

93-10 **Minimização do Consumo de Energia em um Sistema para Aquisição de Dados Controlado por Microcomputador,** *Paulo Cesar Centoducatte, Nelson Castro Machado*

93-11 **An Implementation Structure for RM-OSI/ISO Transaction Processing Application Contexts,** *Flávio Morais de Assis Silva, Edmundo Roberto Mauro Madeira*

93-12 **Boole's conditions of possible experience and reasoning under uncertainty,** *Pierre Hansen, Brigitte Jaumard, Marcus Poggi de Aragão*

93-13 **Modelling Geographic Information Systems using an Object Oriented Framework,** *Fatima Pires, Claudia Bauzer Medeiros, Ardemiris Barros Silva*

93-14 **Managing Time in Object-Oriented Databases,** *Lincoln M. Oliveira, Claudia Bauzer Medeiros*

93-15 **Using Extended Hierarchical Quorum Consensus to Control Replicated Data: from Traditional Voting to Logical Structures,** *Nabor das Chagas Mendonça, Ricardo de Oliveira Anido*

93-16 $\mathcal{LL}$ – **An Object Oriented Library Language Reference Manual,** *Tomasz Kowaltowski, Evandro Bacarin*

93-17 **Metodologias para Conversão de Esquemas em Sistemas de Bancos de Dados Heterogêneos,** *Ronaldo Lopes de Oliveira, Geovane Cayres Magalhães*

93-18 **Rule Application in GIS – a Case Study,** *Claudia Bauzer Medeiros, Geovane Cayres Magalhães*

93-19 **Modelamento, Simulação e Síntese com VHDL,** *Carlos Geraldo Krüger e Mário Lúcio Côrtes*

93-20 **Reflections on Using Statecharts to Capture Human-Computer Interface Behaviour,** *Fábio Nogueira de Lucena e Hans Liesenberg*

93-21 **Applications of Finite Automata in Debugging Natural Language Vocabularies,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

93-22 **Minimization of Binary Automata,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

93-23 **Rethinking the** DNA **Fragment Assembly Problem,** *João Meidanis*

93-24 EGOLib — **Uma Biblioteca Orientada a Objetos Gráficos,** *Eduardo Aguiar Patrocínio, Pedro Jussieu de Rezende*

93-25 **Compreensão de Algoritmos através de Ambientes Dedicados a Animação,** *Rackel Valadares Amorim, Pedro Jussieu de Rezende*

93-26 GeoLab: **An Environment for Development of Algorithms in Computational Geometry,** *Pedro Jussieu de Rezende, Welson R. Jacometti*

93-27 **A Unified Characterization of Chordal, Interval, Indifference and Other Classes of Graphs,** *João Meidanis*

93-28 **Programming Dialogue Control of User Interfaces Using Statecharts,** *Fábio Nogueira de Lucena e Hans Liesenberg*

93-29 EGOLib – **Manual de Referência,** *Eduardo Aguiar Patrocínio e Pedro Jussieu de Rezende*

Relatórios Técnicos – 1994

94-01 **A Statechart Engine to Support Implementations of Complex Behaviour,** *Fábio Nogueira de Lucena, Hans K. E. Liesenberg*

94-02 **Incorporação do Tempo em um SGBD Orientado a Objetos,** *Ângelo Roncalli Alencar Brayner, Claudia Bauzer Medeiros*

94-03 **O Algoritmo KMP através de Autômatos,** *Marcus Vinícius A. Andrade e Cláudio L. Lucchesi*

94-04 **On Edge-Colouring Indifference Graphs,** *Celina M. H. de Figueiredo, João Meidanis, Célia Picinin de Mello*

94-05 **Using Versions in GIS,** *Claudia Bauzer Medeiros and Geneviève Jomier*

94-06 **Times Assíncronos: Uma Nova Técnica para o** Flow Shop Problem**,** *Hélvio Pereira Peixoto e Pedro Sérgio de Souza*

94-07 **Interfaces Homem-Computador: Uma Primeira Introdução,** *Fábio Nogueira de Lucena e Hans K. E. Liesenberg*

94-08 **Reasoning about another agent through empathy,** *Jacques Wainer*

94-09 **A Prolog morphological analyser for Portuguese,** *Jacques Wainer, Alexandre Farcic*

94-10 **Introdução aos Estadogramas,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

94-11 **Matching Covered Graphs and Subdivisions of K_4 and $\overline{C_6}$,** *Marcelo H. de Carvalho and Cláudio L. Lucchesi*

94-12 **Uma Metodologia de Especificação de Times Assíncronos,** *Hélvio Pereira Peixoto, Pedro Sérgio de Souza*

Relatórios Técnicos – 1995

95-01 **Paradigmas de algoritmos na solução de problemas de busca multidimensional,** *Pedro J. de Rezende, Renato Fileto*

95-02 **Adaptive enumeration of implicit surfaces with affine arithmetic,** *Luiz Henrique de Figueiredo, Jorge Stolfi*

95-03 **W3 no Ensino de Graduação?,** *Hans Liesenberg*

95-04 **A greedy method for edge-colouring odd maximum degree doubly chordal graphs,** *Celina M. H. de Figueiredo, João Meidanis, Célia Picinin de Mello*

95-05 **Protocols for Maintaining Consistency of Replicated Data,** *Ricardo Anido, N. C. Mendonça*

95-06 **Guaranteeing Full Fault Coverage for UIO-Based Methods,** *Ricardo Anido and Ana Cavalli*

95-07 **Xchart-Based Complex Dialogue Development,** *Fábio Nogueira de Lucena, Hans K.E. Liesenberg*

95-08 **A Direct Manipulation User Interface for Querying Geographic Databases,** *Juliano Lopes de Oliveira, Claudia Bauzer Medeiros*

95-09 **Bases for the Matching Lattice of Matching Covered Graphs,** *Cláudio L. Lucchesi, Marcelo H. Carvalho*

95-10 **A Highly Reconfigurable Neighborhood Image Processor based on Functional Programming,** *Neucimar J. Leite, Marcelo A. de Barros*

95-11 **Processador de Vizinhança para Filtragem Morfológica,** *Ilka Marinho Barros, Roberto de Alencar Lotufo, Neucimar Jerônimo Leite*

95-12 **Modelos Computacionais para Processamento Digital de Imagens em Arquiteturas Paralelas,** *Neucimar Jerônimo Leite*

95-13 **Modelos de Computação Paralela e Projeto de Algoritmos,** *Ronaldo Parente de Menezes e João Carlos Setubal*

Departamento de Ciência da Computação — IMECC
Caixa Postal 6065
Universidade Estadual de Campinas
13081-970 – Campinas – SP
BRASIL

reltec@dcc.unicamp.br