Incremental bipartite drawing problem

Rafael Martí*, Vicente Estruch

Departamento de Estadística e Investigación Operativa, Facultad de Matemáticas, Universidad de Valencia, Dr. Moliner 50, 46100 Burjassot, Valencia, Spain

Received 1 June 1999; received in revised form 1 March 2000

Abstract

Layout strategies that strive to preserve perspective from earlier drawings are called incremental. In this paper we study the incremental arc crossing minimization problem for bipartite graphs. We develop a greedy randomized adaptive search procedure (GRASP) for this problem. We have also developed a branch-and-bound algorithm in order to compute the relative gap to the optimal solution of the GRASP approach. Computational experiments are performed with 450 graph instances to first study the effect of changes in grasp search parameters and then to test the efficiency of the proposed procedure.

Scope and purpose

Many information systems require graphs to be drawn so that these systems are easy to interpret and understand. Graphs are commonly used as a basic modeling tool in areas such as project management, production scheduling, line balancing, business process reengineering, and software visualization. Graph drawing addresses the problem of constructing geometric representations of graphs. Although the perception of how good a graph is in conveying information is fairly subjective, the goal of limiting the number of arc crossings is a well-admitted criterion for a good drawing. Incremental graph drawing constructions are motivated by the need to support the interactive updates performed by the user. In this situation, it is helpful to preserve a “mental picture” of the layout of a graph over successive drawings. It would not be very intuitive or effective for a user to have a drawing tool in which after a slight modification of the current graph, the resulting drawing appears very different from the previous one. Therefore, generating incrementally stable layouts is important in a variety of settings. Since “real-world” graphs tend to be large, an automated procedure to deal with the arc crossing minimization problem in the context of incremental strategies is desirable. In this article, we develop a procedure to minimize arc crossings that is fast and capable of dealing with large graphs, restricting our attention to bipartite graphs. © 2001 Elsevier Science Ltd. All rights reserved.

Keywords: Combinatorial optimization; Heuristic search; Graph drawing

*Corresponding author. Tel.: +34-96-386-4362; fax: +34-96-386-4735.
E-mail address: rafael.marti@uv.es (R. Martí).
1. Introduction

Researchers in the graph-drawing field have proposed several drawing conventions and aesthetic criteria that attempt to capture the meaning of a “good” map of a graph. Although readability may depend on the context and the map’s user, most authors agree that crossing reduction is a fundamental aesthetic criterion in graph drawing. We refer the reader to the recent book by Di Battista et al. [1] for a survey on graph drawing.

The problem of minimizing the number of arc crossings in a 2-layer graph where the edges are drawn as straight lines (bipartite drawing problem or BDP) is known to be NP-complete [2]. This version of the problem as well as the general multi-layer case have been the subject of study for at least 17 years. Several heuristic algorithms have been proposed through the years, beginning with the relative degree algorithm introduced by Carpano [3]. These heuristics are based on simple ordering rules, reflecting the goal of researchers and practitioners of obtaining solutions of reasonable quality fast [4,5]. Recent developments, however, have shown that meta-heuristic approaches can be successfully applied to the BDP [6,7]. An exact procedure, due to Valls et al. [8], has also been developed for BDP. Martí and Laguna [9] present a computational comparison of algorithms for BDP.

In a great number of practical situations, it is helpful to preserve a “mental picture” of the layout of a graph over successive drawings. It can be distracting to make a slight modification, perform the graph-drawing algorithm, and have the resulting drawing appear very different from the previous one. Layout strategies that strive to preserve perspective from earlier drawings are called incremental. As it is stated in Di Battista et al. [1]: Incremental strategies are motivated by the need to support the interactive updates performed by the user. In this paper we study the incremental drawing problem in the context of bipartite graphs. Specifically, we consider the problem of minimizing the number of arc crossings when new vertices and arcs have been added to the graph.

In order to preserve the drawing of the original graph, we keep the relative position among the old vertices as it appears in the previous drawing. To the best of our knowledge, this problem has not been treated in the literature so far, although it deals with a key point in graph drawing systems.

Fig. 1 shows three graphs. The first one represents the original graph. The second is obtained by adding four vertices and four arcs to the first one. In the third one the number of arc crossings is minimized preserving the relative position of the old vertices (those in the original graph).

In mathematical terms, let \( G = (V, E) \) be a graph where \( V = V_1 \cup V_2 \). Since the vertices are constrained to lie on two vertical lines (layers) and each arc is drawn as a straight line between two vertices, a drawing is completely specified by the permutations of both layers. Let \( D = (G, \sigma_1, \sigma_2) \) be a drawing of a graph \( G \) where \( \sigma_1 \) is a permutation of \( V_1 \) and \( \sigma_2 \) is a permutation of \( V_2 \) (i.e., \( \sigma_1(v) \) or \( \sigma_2(v) \) is the position of \( v \) in its corresponding layer).

Suppose that \( \sigma_1 \) is a fixed ordering of \( V_1 \) and \( u \) and \( v \) are two vertices in \( V_2 \). Then \( K(u, v) \) is defined as the number of crossings that arcs incident with \( u \) make with arcs incident with \( v \) when \( \sigma_2(u) < \sigma_2(v) \). Note that given \( u, v \in V_2 \) the value \( K(u, v) \) depends only on the positions \( \sigma_2(u) \) and \( \sigma_2(v) \) and the ordering \( \sigma_1 \). Similarly, we define \( K(u, v) \) for a fixed ordering \( \sigma_2 \) of \( V_2 \) and vertices \( u, v \) in \( V_1 \). The number of crossings \( K(D) \) of a drawing \( D = (G, \sigma_1, \sigma_2) \) can be calculated as follows:

\[
K(D) = \sum_{u, v \in V_1} K(u, v) = \sum_{u, v \in V_2} K(u, v).
\]
The graph $IG = (IV, IE)$, where $IV = IV_1 \cup IV_2$, is an incremented graph of $G = (V, E)$ where $V = V_1 \cup V_2$, if $V_1 \subseteq IV_1$, $V_2 \subseteq IV_2$ and $E \subseteq IE$. A drawing $ID = (IG, \pi_1, \pi_2)$ is an incremental drawing of $D = (G, \sigma_1, \sigma_2)$ if

$$\pi_i(v) < \pi_i(w) \quad \forall v, w \in V/\sigma_i(v) < \sigma_i(w), \quad i = 1, 2.$$  

Given an incremented graph $IG$ of a graph $G$ and a drawing $D$ of $G$, the incremental bipartite drawing problem (IBDP) consist of finding an incremental drawing $ID$ of $D$ such that the number of arc crossings is minimized. Obviously, a trivial lower bound on the number of crossings for every $ID$ of $D$, is the number of arc crossings of $D$.

In the example given in Fig. 1 the sets are: $V_1 = \{1, 2, 3\}$, $V_2 = \{4, 5, 6\}$, $IV_1 = \{1, 2, 3, 7, 8\}$ and $IV_2 = \{4, 5, 6, 9, 10\}$. Fig. 1 also shows that the incremental drawing preserves the relative position of the vertices in the original graph; for example, $\pi_1(3) = 2 < \pi_1(1) = 3$ since $\sigma_1(3) = 1 < \sigma_1(1) = 2$. The number of arc crossings in the solution given in the figure (third graph), could be reduced by ordering $IV_1$ according to $(8, 1, 3, 7, 2)$; however, this will not produce a feasible solution for the IBDP.

In this paper we undertake to explore methods for solving the IBDP. In particular, we develop an exact and a heuristic algorithm. The exact procedure is based on branch-and-bound techniques while the heuristic is a greedy randomized search procedure (GRASP). IBDP is an NP-hard problem and the branch-and-bound procedure requires exponential time, thus only relatively small instances can be optimally solved. We have adapted the exact algorithm given in Valls et al. [8] for the BDP to our problem. The computational testing in Section 4 with 450 graphs shows the performance of both approaches.

2. Branch-and-bound approach

Given an incremented graph $IG = (IV, IE)$ of a graph $G = (V, E)$ and a drawing $D = (G, \sigma_1, \sigma_2)$, the search tree provides a generation and partition of the set of solutions for IBDP (incremental drawings of $D$). Consider that the vertices in the original graph $G$ are $V_1 = \{v_1, v_2, \ldots, v_n\}$ and
\( V_2 = \{v_{n+1}, v_{n+2}, \ldots, v_{n+m}\} \), numbered according to \( \sigma_1 \) and \( \sigma_2 \), respectively (\( \sigma_1(v_i) = i \), \( \sigma_2(v_{n+j}) = j \)). Let \( iv_1, \ldots, iv_{s+t} \) be the vertices added to \( V \) to obtain \( IV \) as

\[
IV_1 = V_1 \cup \{iv_1, iv_2, \ldots, iv_s\} \quad \text{and} \quad IV_2 = V_2 \cup \{iv_{s+1}, iv_{s+2}, \ldots, iv_{s+t}\}.
\]

Node 0 in the tree represents all the possible solutions for IBDP, i.e., the entire set of incremental drawings \( ID = (IG, \pi_1, \pi_2) \). For each of the \( n+1 \) possible positions in which element \( iv_1 \) can be inserted in \((v_1, v_2, \ldots, v_n)\) a node is generated

\[
(iv_1, v_1, v_2, \ldots, v_n), (v_1, iv_1, v_2, \ldots, v_n), (v_1, v_2, iv_1, \ldots, v_n) \ldots (v_1, v_2, \ldots, v_n, iv_1).
\]

Node 0 branches into these \( n+1 \) nodes in depth 1 of the tree. Each one represents all the solutions in which the position of vertex \( iv_1 \) is fixed relative to the position of vertices in \( V_1 \). Similarly, each of these nodes branches in \( n+2 \) nodes where the position of \( iv_2 \) is fixed relative to the position of vertices in \( V_1 \cup \{iv_1\} \). Then, each node at depth \( k \) in the tree \((k < s)\) branches into \( n+k+1 \) nodes, by fixing vertex \( iv_{k+1} \) in all possible positions with respect to previous fixed vertices. Nodes at depth \( s \) have the \( n+s \) vertices of \( IV_1 \) fixed, then they represent the set of solutions in which the ordering \( \pi_1 \) of \( IV_1 \) is fixed.

For each node at depth \( s \) we consider all the permutations \( \pi_2 \) of \( IV_2 \) satisfying \( \sigma_2(\pi_2(v_{n+i})) < \sigma_2(\pi_2(v_{n+j})) \), \( 1 \leq i < j \leq m \). Each permutation is constructed fixing a vertex in each depth of the tree, in a similar way to \( \pi_1 \). Each node at depth \( s + t \) represents a permutation \( \pi_1 \) of \( IV_1 \) satisfying \( \pi_1 \) and a permutation \( \pi_2 \) of \( IV_2 \) satisfying \( \sigma_2 \), thus it represents a solution of the IBDP.

The number of crossings of an initial solution gives an upper bound of the problem. The tree is explored in a depth first search starting with node 0. A lower bound is computed at each node, if it is greater than the upper bound we fathom the node, otherwise we branch the node and explore its first son. Fig. 2 shows a partial representation of the search tree for the example given in Fig. 1.

Suppose at node \( N \) of depth \( r \) in the search tree, you have positioned the vertices \( V_1 \) and \( V_2 \) using permutations \( \pi_1 \) on \( V_1 \) and \( \pi_2 \) on \( V_2 \). (Note that if \( r < s \) then \( V_1 \subseteq V_1^N \subseteq IV_1 \), \( V_2^N = V_2 \), and if \( s < r \leq s + t \) then \( V_1^N = IV_1 \), \( V_2 \subseteq V_2^N \subseteq IV_2 \).) A lower bound on the number of crossings for every solution in \( N \) can be computed as \( LB = K_A + K_B + K_C \), where \( K_A \) is the number of crossings of arcs incident with vertices in \( V_1^N \), \( K_B \) is a lower bound on the number of crossings produced between arcs incident with a vertex in \( V_2^N \) and arcs incident with a vertex in \( IV_2 - V_2^N \), and \( K_C \) is a lower bound on the number of crossings produced between arcs incident with vertices in \( IV_2 - V_2^N \):

\[
K_A = \sum_{u,v \in V_1} K^N(u,v),
\]

\[
K_B = \sum_{u \in V_1} \sum_{v \in IV_2 - V_2^N} \min(K^N(u,v), K^N(v,u)),
\]

\[
K_C = \frac{1}{2} \sum_{u,v \in IV_2 - V_2^N} \min(K^N(u,v), K^N(v,u)),
\]

where \( K^N(u,v) \) is defined as the number of crossings that arcs from a vertex in \( V_1^N \) to \( u \) make with arcs from a vertex in \( V_1^N \) to \( v \) when \( \pi_2(u) < \pi_2(v) \).
Let $ID = (IG, \pi_1, \pi_2)$ be an optimal solution of the problem, if $u, v$ are consecutive vertices in $IV_2(\pi_2(v) = \pi_2(u) + 1)$, then $K(u, v) < K(v, u)$. From this trivial property, the following fathom test may be used for nodes at depth $s + k$, since they cannot contain an optimal solution for the problem. If $K(u, v) > K(v, u)$ and $u$ precedes $v$ in the partial ordering of the node, then fathom the node.

3. GRASP approach

The GRASP methodology was developed in the late 1980s, and the acronym was coined by Tom Feo [10]. It was first used to solve computationally difficult set covering problems [11]. Each GRASP iteration consists of constructing a trial solution and then applying an exchange procedure to find a local optimum (i.e., the final solution for that iteration). The construction phase is iterative, greedy, and adaptive. It is iterative because the initial solution is built considering one element at a time. It is greedy because the addition of each element is guided by a greedy function. It is adaptive because the element chosen at any iteration in a construction is a function of those previously chosen. (That is, the method is adaptive in the sense of updating relevant information from one construction step to the next.) The improvement phase typically consists of a local search procedure.

Performing multiple GRASP iterations may be interpreted as a means of strategically sampling the solution space. Based on empirical observations, it has been found that the sampling distribution generally has a mean value that is inferior to the one obtained by a deterministic construction, but the best over all trials dominates the deterministic solution with a high probability. The intuitive justification of this phenomenon is based on the ordering statistics of sampling. GRASP implementations are generally robust in the sense that it is difficult to find or devise pathological instances for which the method will perform arbitrarily bad. The robustness of this method has been well documented in applications to production, flight scheduling, equipment and tool selection, location, and maximum independent sets.
The most important element in the construction phase is that the selection in each step must be guided by a greedy function that adapts according to the selections in previous steps. The improving phase performs a sequence of moves towards a local optimum solution, which becomes the output of a complete GRASP iteration. The details of the two GRASP phases follow.

3.1. Construction phase

This phase starts by creating a list $U$ of unassigned vertices, which at the beginning consists of all the vertices in the graph (i.e., initially $U = IV$). Also, the current position of each vertex is assigned a value of zero (i.e., $\pi_1(v) = 0 \ \forall v \in IV_1$ and $\pi_2(v) = 0 \ \forall v \in IV_2$). The first vertex $v$ is randomly selected from all those vertices in $U$ with maximum degree.

Once $v$ has been positioned in the partial solution, $U$ is updated by deleting $v$ ($U = U - \{v\}$). In subsequent construction steps, the next vertex $v$ is randomly selected from a set $U'$ that consists of vertices with a degree of no less than $z$ times the maximum degree of all the vertices in $U$, for some predetermined multiple $z$, $0 \leq z \leq 1$. Vertex degree, in this case, is calculated with respect to the subgraph given by the partial solution obtained from previous vertex selections. That is, if $\rho(v, V)$ denotes the degree of a vertex $v$ with respect to the set of vertices $V$, then

$$U' = \{v \in U \mid \rho(v, IV - U) \geq z \rho_{\text{max}}\} \quad \text{where} \quad \rho_{\text{max}} = \max \{\rho(v, IV - U) \ \forall v \in U\}.$$  

A selected vertex $v$ is placed in its layer in the position prescribed by the barycenter calculation, except for the first vertex, which is placed in an arbitrary position. The barycenter of a vertex $v \in V_1, bc(v)$, is the arithmetic mean of the current positions of the vertices $w \in V_2$ adjacent to $v$. In mathematical terms, the barycenter is

$$bc(v) = \frac{\sum_{w \in K} \pi_2(w)}{|K|} \quad \text{where} \quad K = \{w \in V_2 \mid \pi_2(w) > 0 \ \text{and} \ \{v, w\} \in E\}.$$  

If vertex $v$ belongs to the original graph (i.e., $v \in V_1$), then $v$ is placed in the closest feasible position to $bc(v)$ with respect to the original vertices previously assigned ($\pi_1(v) < \pi_1(w) \ \forall w \in V_1/\sigma_1(v) < \sigma_1(w)$). Otherwise, we try the assignments $\pi_1(v) = \lceil bc(v) \rceil$ and $\pi_1(v) = \lfloor bc(v) \rfloor$ since $bc(v)$ is a fractional value, and select the best. If a previously assigned vertex already occupies one of both positions, then we try either one position “before” or one position “after” and select the best (that one that produces the least number of crosses with respect to the partial solution). Similar calculations are carried out for a selected vertex $v \in V_2$. It should be noted that when a position is already occupied, then a renumbering of some of the $\pi$ values may be necessary to fit $v$ in “before” or “after” the occupying vertex.

Once vertex $v$ has been positioned in the partial solution, it is deleted from the set $U$ and the vertex degree calculations $\rho(v, V - U)$ are updated accordingly. The construction phase terminates after $|IV|$ steps, when all vertices have been selected and positioned.

3.2. Improvement phase

An improving step begins with making $U = V$. Each step of the improvement phase consists of selecting each vertex to be considered for a move. The probabilistic selection rule is such that vertices with higher degree $\rho(v, V)$ are more likely to be selected first at each step of this process. In
particular, the probability \( \Pr(v) \) that a vertex \( v \) is selected is given by
\[
\Pr(v) = \frac{\rho(v, V)}{\sum_{w \in V} \rho(w, V)}.
\]
When a vertex \( v \in IV_1 \) is selected, three moves are considered: (1) to insert the vertex one position before the barycenter (i.e., \( \pi_1(v) = \lceil bc(v) \rceil - 1 \)), (2) to insert the vertex at the barycenter position (i.e., \( \pi_1(v) = \lfloor bc(v) \rfloor \) or \( \lfloor bc(v) \rfloor + 1 \)), and (3) to insert the vertex one position after the barycenter (i.e., \( \pi_1(v) = \lceil bc(v) \rceil + 1 \)). If vertex \( v \in V_1 \) and we move it to another position, only feasible positions according to \( \sigma_1 \) can be considered. If \( u \in V_1 \) and precedes \( v \) in \( \sigma_1 \) (\( \sigma_1(u) < \sigma_1(v) \)) then \( u \) must precede \( v \) in \( \pi_1 \). Similarly, for a vertex \( u \in V_1 \) in a posterior position of \( v \). Then, the three considered values for \( \pi_1(v) \) are mapped to the closest feasible positions.

The vertex \( v \) is placed in the position that produces the maximum reduction in the number of crossings. We assume that when the position of a vertex changes, the position of the other vertices are updated to preserve the correct relative ordering. If no reduction is possible, then the vertex is not moved. The vertex is removed from the set \( \tilde{U} \) after being considered, so \( U = U - \{v\} \). An improvement step terminates when all vertices have been considered for insertion, i.e., when \( U = \emptyset \).

Hence, an improvement step consists of \( |IV| \) trials. More steps are performed as long as at least one vertex is moved (i.e., as long as the current solution keeps improving).

When a step fails to improve the current solution, and before abandoning the improvement phase, an attempt is made to exchange the positions of vertices \( v \) and \( w \) in the same layer in order to find an improved solution. We restrict the search to exchanges of vertices that are one position away from each other. In other words, we exchange the positions of \( v \) and \( w \) as long as \( \pi_1(v) = \pi_1(w) + 1 \). If \( u, v \in V_1 \) (\( u, v \in V_2 \)) we do not try the switching since it will violate \( \sigma_1(\sigma_2) \). This process is performed on each layer, according to the vertex order in the current solution, i.e., \( \pi_1(v) = 1, \ldots, n_1 - 1 \) and \( \pi_2(v) = 1, \ldots, n_2 - 1 \).

After a number of GRASP iterations, it is possible to estimate the percent improvement achieved by the application of the improving phase and use this information to increase the efficiency of the search. Then, at a given iteration, these estimates can be used to determine whether it is “likely” that the improving phase will be able to improve the current construction enough as to produce a better solution than the current best. In particular, we calculate the minimum percent improvement \( x \) that is necessary for a construction to be better than the current best. Therefore, if this percent \( x \) is beyond \( \beta \) standard deviations away from the estimated mean percent improvement, then the construction is discarded and the improving phase is not performed.

4. Computational experiments

Graphs are generated with the random_bigraph code of the Stanford GraphBase by Knuth [12], then they are incremented adding vertices and arcs up to pre-established numbers. These numbers are calculated as a percentage of the quantities in the original graph
\[
|IV_1| = \delta|V_1|, \quad |IV_2| = \delta|V_2| \quad \text{and} \quad |IE| = \delta|E| \quad \text{where} \quad \delta \geq 1.
\]
Arcs in \( IE - E \) are generated as follows. For each vertex in \( IV_1 - V_1 \), an arc to a randomly chosen vertex in \( IV_2 \) is included. Similarly, for each vertex in \( IV_2 - V_2 \), an arc to a randomly chosen
vertex in $IV_1$ is included. This guarantees that each new vertex has a degree of at least one. Additional arcs are added by randomly choosing two vertices, one in each layer, up to the desired number $|IE|$. Since the most applied heuristic for BDP is the barycenter method, we have used it to obtain the drawing $D = (G, \sigma_1, \sigma_2)$ of the original graph. Moreover, independent studies by Jünger and Mutzel [13] and Martí and Laguna [9] have concluded that the barycenter algorithm outperforms rival approaches based on relatively simple vertex positioning rules.

The procedures described in the previous sections were implemented in C, and all experiments were performed on a Pentium 166 MHz personal computer. Before testing the effectiveness of our procedures, we perform three preliminary experiments to explore the effect of changes in the two search parameters $\alpha$ and $\beta$ of the GRASP approach. We also explore the effect of allowing the procedure to run longer, by increasing the number of iterations (STOP) that the procedure is allowed to run without improving the best solution found.

For these preliminary experiments 90 graphs have been generated with $|V_1| = |V_2| = |E|/2$ and $\delta = 1.4$. For each value of $|E| = 10$, 50 and 90, 30 instances have been considered. The experiments can be described as follows:

- A termination criterion of 100 GRASP iterations without improvement is established (i.e., STOP = 100). We consider the value $\alpha = 2/3$ and we test the effect of changing $\beta$ by assigning values 1–4. Results of this experiment are reported in Table 1.
- We set $\alpha = 2/3$, $\beta = 3$ and try a termination criterion STOP = 50, 100 and 150 GRASP iterations without improvement. Results of this experiment are reported in Table 2.
- We set $\beta = 3$, STOP = 100 and try $\alpha = 1$, 1/3 and 2/3. Results of this experiment are reported in Table 3.

These tables show that the results are not very sensitive to the variations of the parameters. Anyway, Table 1 shows that as $\beta$ increases the average solution quality also increases. This is to be expected, since skipping the improvement phase may result in a missing opportunity to improve the best solution found. Table 2 also shows an improving trend in terms of solution quality as the procedure is allowed to run longer. Table 3 shows some mixed results. On the one hand, the solution quality increases when $\alpha$ is changed from 1/3 to 2/3. On the other, the solution quality slightly decreases when $\alpha$ is changed from 2/3 to 1. Given the results of the preliminary experimentation we use STOP = 100, $\alpha = 2/3$ and $\beta = 3$ for the GRASP procedure.

In the first experiment, we employ medium size instances in order to compute the optimal solution with the branch and bound procedure. These instances have 10 vertices in each layer of the original graph and a number of edges ranging from 10 to 90, then they are incremented with $\delta = 1.2$ and 1.6. Table 4 shows the results with 90 instances with $\delta = 1.2$ (10 instances for each edge density) and Table 5 considers 90 instances with $\delta = 1.6$. The drawing of the original graph $D = (G, \sigma_1, \sigma_2)$ is obtained with the barycenter method, then a trivial lower bound on the number of crossings of the incremental drawing is given by the number of crossings $K(D)$ of the original drawing. Tables 4 and 5 report, respectively:

- The average number of crossings of the GRASP procedure (Grasp).
- The average number of crossings of the optimal solutions (branch and bound).
- The average number of crossings of the initial solution, where new vertices are placed just “after” the original ones (initial solution).
Table 1

<table>
<thead>
<tr>
<th>( \beta )</th>
<th>Average no. of crossings</th>
<th>Average CPU seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>6.4</td>
<td>194.9</td>
</tr>
<tr>
<td>2</td>
<td>6.4</td>
<td>195.2</td>
</tr>
<tr>
<td>3</td>
<td>6.4</td>
<td>194.7</td>
</tr>
<tr>
<td>4</td>
<td>6.4</td>
<td>194.7</td>
</tr>
</tbody>
</table>

Table 2

<table>
<thead>
<tr>
<th>STOP</th>
<th>Average no. of crossings</th>
<th>Average CPU seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>50</td>
<td>6.4</td>
<td>194.9</td>
</tr>
<tr>
<td>100</td>
<td>6.4</td>
<td>194.7</td>
</tr>
<tr>
<td>150</td>
<td>6.4</td>
<td>194.7</td>
</tr>
</tbody>
</table>

Table 3

<table>
<thead>
<tr>
<th>( x )</th>
<th>Average no. of crossings</th>
<th>Average CPU seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>1/3</td>
<td>6.4</td>
<td>195.1</td>
</tr>
<tr>
<td>2/3</td>
<td>6.4</td>
<td>194.7</td>
</tr>
<tr>
<td>1</td>
<td>6.4</td>
<td>194.9</td>
</tr>
</tbody>
</table>

The average deviation of the GRASP procedure from the optimal solutions (Opt. deviation).
- The value of the trivial lower bound (lower bound).
- The average deviation of the GRASP procedure from the lower bound (LB deviation).
- The number of optimal solutions found by the grasp procedure (num. of optima).
- The average CPU seconds of the GRASP procedure (run time).

These tables show that the GRASP method provides high-quality solutions, since it is able to match 75 optimal solutions with \( \delta = 1.2 \) and 39 with \( \delta = 1.6 \). Moreover, it presents an average percent deviation from optima of 0.1% when \( \delta = 1.2 \) achieved on an average of 0.2 s, and 1.4% in 0.6 s on average when \( \delta = 1.6 \). Optimal solutions in Table 4 were found with the branch and bound procedure in few minutes. It turns out in Table 5 that with increasing density, the computation time of the branch-and-bound algorithm increases rapidly. Some instances with \(|E| > 50\) take more than 1 h of computation. The Grasp method is able to reduce the number of crossings in a 12% on average from the initial solution. We have compared the heuristic solutions with the lower bound as a reference point for the second experiment.

It should be noted that the performance of the heuristic deteriorates, as the graph becomes sparser. Table 5 shows an average percent deviation from optima of 5.9% for \(|E| = 20\) while this deviation is lower than 0.5% for \(|E| > 50\). An exception is given in \(|E| = 10\), but note that there is only one arc for each vertex, so all the solutions produce the same number of crossings (including the optimal solution). These results confirm what has been proven in other crossing minimization problems: Sparse graphs are more difficult to optimize than dense graphs.
Table 4  
|V_1| = |V_2| = 10, δ = 1.2

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower bound</td>
<td>1.3</td>
<td>18.4</td>
<td>68.8</td>
<td>164.4</td>
<td>291.9</td>
<td>461.7</td>
<td>701.5</td>
<td>1002.5</td>
<td>1427.2</td>
<td>459.7</td>
</tr>
<tr>
<td>Initial solution</td>
<td>9.1</td>
<td>40.9</td>
<td>130.2</td>
<td>295.1</td>
<td>514.7</td>
<td>813.7</td>
<td>1158.9</td>
<td>1630.1</td>
<td>2181.2</td>
<td>752.7</td>
</tr>
<tr>
<td>Grasp</td>
<td>1.5</td>
<td>24.0</td>
<td>99.0</td>
<td>239.1</td>
<td>429.6</td>
<td>672.4</td>
<td>1020.8</td>
<td>1430.9</td>
<td>2024.1</td>
<td>660.2</td>
</tr>
<tr>
<td>Optimum</td>
<td>1.5</td>
<td>23.9</td>
<td>99.0</td>
<td>238.1</td>
<td>429.1</td>
<td>671.3</td>
<td>1020.1</td>
<td>1430.9</td>
<td>2023.9</td>
<td>659.8</td>
</tr>
<tr>
<td>LB deviation (%)</td>
<td>7.5</td>
<td>33.6</td>
<td>44.6</td>
<td>45.7</td>
<td>47.3</td>
<td>45.8</td>
<td>45.6</td>
<td>42.8</td>
<td>41.8</td>
<td>39.4</td>
</tr>
<tr>
<td>Opt deviation (%)</td>
<td>0.0</td>
<td>0.3</td>
<td>0.0</td>
<td>0.4</td>
<td>0.1</td>
<td>0.2</td>
<td>0.1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.1</td>
</tr>
<tr>
<td>No. of optima</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>10</td>
<td>9</td>
<td>75</td>
</tr>
<tr>
<td>Run time</td>
<td>0.07</td>
<td>0.11</td>
<td>0.12</td>
<td>0.18</td>
<td>0.21</td>
<td>0.22</td>
<td>0.24</td>
<td>0.27</td>
<td>0.36</td>
<td>0.20</td>
</tr>
</tbody>
</table>

Table 5  
|V_1| = |V_2| = 10, δ = 1.6

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
<th>70</th>
<th>80</th>
<th>90</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower bound</td>
<td>1.3</td>
<td>18.4</td>
<td>68.8</td>
<td>164.4</td>
<td>291.9</td>
<td>461.7</td>
<td>701.5</td>
<td>1002.5</td>
<td>1427.2</td>
<td>459.7</td>
</tr>
<tr>
<td>Initial solution</td>
<td>15.2</td>
<td>102.9</td>
<td>290.0</td>
<td>548.8</td>
<td>908.3</td>
<td>1368.7</td>
<td>1861.3</td>
<td>2564.3</td>
<td>3397.4</td>
<td>1228.5</td>
</tr>
<tr>
<td>Grasp</td>
<td>1.4</td>
<td>29.6</td>
<td>162.3</td>
<td>385.3</td>
<td>736.9</td>
<td>1147.3</td>
<td>1680.3</td>
<td>2309.7</td>
<td>3189.3</td>
<td>1071.3</td>
</tr>
<tr>
<td>Optimum</td>
<td>1.4</td>
<td>28.5</td>
<td>156.4</td>
<td>380.3</td>
<td>732.6</td>
<td>1143.7</td>
<td>1678.1</td>
<td>2303.7</td>
<td>3184.7</td>
<td>1067.7</td>
</tr>
<tr>
<td>LB dev. (%)</td>
<td>3.3</td>
<td>68.6</td>
<td>141.0</td>
<td>135.3</td>
<td>153.6</td>
<td>149.5</td>
<td>139.6</td>
<td>130.5</td>
<td>123.5</td>
<td>116.1</td>
</tr>
<tr>
<td>Opt dev. (%)</td>
<td>0.0</td>
<td>5.9</td>
<td>3.9</td>
<td>1.2</td>
<td>0.6</td>
<td>0.3</td>
<td>0.1</td>
<td>0.3</td>
<td>0.1</td>
<td>1.4</td>
</tr>
<tr>
<td>No. of optima</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>39</td>
</tr>
<tr>
<td>Run time</td>
<td>0.10</td>
<td>0.16</td>
<td>0.30</td>
<td>0.43</td>
<td>0.55</td>
<td>0.70</td>
<td>1.01</td>
<td>1.31</td>
<td>1.12</td>
<td>0.63</td>
</tr>
</tbody>
</table>

In our second experiment, we undertake to study the performance of the GRASP procedure using sparse large graphs. In specific, we generate 180 additional instances with number of vertices in each layer equal to the half of the number of edges. We generate 90 instances with number of edges ranging from 10 to 90 in increments of 10. These instances are incremented with δ = 1.2 in Table 6 and with δ = 1.6 in Table 7. Within 2 h of computation time, the branch-and-bound algorithm can find optimum solutions for instances with up to 20 vertices in each layer. Then, we cannot report optimal solutions for this experiment. Tables 6 and 7 report, respectively, the average number of crossings of the GRASP algorithm, the lower bound, the average deviation from the lower bound, the average number of crossings of the initial solution, and the average CPU seconds.

In order to compare the results in Tables 6 and 7 with those given in previous tables, it should be noted that the second column in both tables (|E| = 20) also appears in Tables 4 and 5, respectively. We cannot assess how close the Lower bound values are from the optimal solutions, but we know these distances in small instances (Tables 4 and 5). We know that when |E| = 20, the percent
deviation from optima is 0.3% and from the lower bound is 33.6% when $\delta = 1.2$ and 5.9% and 68.6%, respectively when $\delta = 1.6$. Moreover, the percent of reduction in the number of crossings achieved by the Grasp algorithm from the initial solution is, for $|E| = 20$, 41% when $\delta = 1.2$ and 71% when $\delta = 1.6$. Tables 6 and 7 present instances with similar densities (or even lower) to those ones presented in column $|E| = 20$ in Tables 4 and 5. Then, we can see that the GRASP approach perform remarkably well in sparse instances since it obtains an average percentage of reduction from the initial solution of 62% when $\delta = 1.2$ and 79% when $\delta = 1.6$, and an average percent deviation from lower bound of 27.7 and 79.5%, respectively. The computational effort associated to the heuristic procedure is still modest (lower than 5 s).

5. Conclusions

We have developed a heuristic procedure based on the GRASP methodology to provide high-quality solutions to the incremental problem of minimizing straight-line crossings in a 2-layer graph. We have also developed an exact algorithm based on the branch-and-bound techniques. As far as we know, this is the first study that deals with this problem.

Overall, experiments with 450 graphs were performed to test both procedures. In medium-size instances, where the optimal solution is obtained with the exact method, the GRASP procedure
presents an average percent deviation from optima of 0.75%, achieved on 0.4 s. We have also tested the heuristic procedure in large sparse instances, where a percentage of reduction in the number of crossings of 70% from the initial solution is obtained on average. This experimentation allows us to conclude that our GRASP implementation performs remarkably well.

References


Rafael Martí is a Titular Professor in the Statistics and Operations Research Department at the University of Valencia, Spain. He received a doctoral degree in Mathematics from the University of Valencia. His areas of research include Graph Drawing, Routing Problems and Metaheuristics for solving Combinatorial Optimization Problems.

Vicente Estruch is a doctoral student in the Statistics and Operations Research Department at the University of Valencia, Spain. He is currently working in graph drawing problems.