GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization

MANUEL LAGUNA

Graduate School of Business, University of Colorado, Campus Box 419, Boulder, CO 80309
Manuel.Laguna@Colorado.Edu

RAFAEL MARTÍ

Departamento de Estadística e Investigación Operativa
Facultad de Matemáticas, Universidad de Valencia
Dr. Moliner 50, 46100 Burjassot (Valencia) Spain
Rafael.Marti@uv.es

Submitted: May 21, 1997
First revision: February 24, 1998
Second revision: June 10, 1998
Final version: August 21, 1998

ABSTRACT — In this paper, we develop a greedy randomized adaptive search procedure (GRASP) for the problem of minimizing straight-line crossings in a 2-layer graph. The procedure is fast and is particularly appealing when dealing with low-density graphs. When a modest increase in computational time is allowed, the procedure may be coupled with a path relinking strategy to search for improved outcomes. Although the principles of path relinking have appeared in the tabu search literature, this search strategy has not been fully implemented and tested. We perform extensive computational experiments with more than 3,000 graph instances to first study the effect of changes in critical search parameters and then to compare the efficiency of alternative solution procedures. Our results indicate that graph density is a major influential factor on the performance of a solution procedure.

The problem of minimizing straight-line crossings in layered graphs has been the subject of study for at least 17 years, beginning with the Relative Degree Algorithm introduced by Carpano[2]. The problem consists of aligning the two shores V_1 and V_2 of a bipartite graph $G = (V_1, V_2, E)$ on two parallel straight lines (layers) such that the number of crossing between the edges in E is minimized when the edges are drawn as straight lines connecting the end-nodes (Jünger and Mutzel, 1997). The problem is also known as the *bipartite drawing problem* (or BDP). In the BDP the problem consists of finding an optimal ordering for the vertices in both layers, which differ from the *layer permutation problem* (LPP) that seeks the optimal ordering of one layer only. Table 1 summarizes some of the relevant work in the area to the present. The research listed in Table 1 combines procedures specifically designed for both 2-layer and multi-layer graphs. In some instances, however, LPP procedures have been extended to the BDP case, in a similar way that 2-layer graph methods have been adapted to the multi-layer case.

Table 1 Summary of relevant literature.

Reference	*Procedure*	*Comments*
Carpano[2]	Relative degree algorithm	
Sugiyama, et al.[21]	Barycenter	Similar to Carpano's.
Eades and Kelly[4]	Greedy insertion Splitting Averaging Greedy switching	Same as barycenter.
Eades and Wormald[6]	Median	
Sugiyama[20] Rowe, et al.[19]	Barycenter / median	Variations and extensions to the barycenter / median approach.
Makinen[15]	Average median Semi median	Barycenter / median hybrids.
Gansner, et al.[10] Eades, et al.[5]	Barycenter / median	Variations and extensions to the barycenter / median approach to multi-layered graphs.
Catarci[3]	Assignment	
Jünger and Mutzel[12]	Branch and cut	Includes a comparative study of heuristic approaches.
Valls, et al.[22]	Branch and bound	Small instances.
Valls, et al.[23] Martí[16]	Tabu thresholding Tabu search	High quality results with long computational times.
Martí and Laguna[17]	Several existing methods	Empirical analysis of 16 heuristics.
Laguna, et al.[14]	Tabu search	Multi-layer graphs.

The main application of these problems is found in automated drawing systems, where drawing speed is a critical factor. This is why for many years researchers were interested only in designing simple heuristic procedures and sacrificed solution quality for speed. The motivation for our current development is to provide high quality solutions to the 2-layer straight line cross minimization problem within a computational time that approaches simple heuristics. We will show that simple heuristics are very fast but result in inferior solutions, while high-quality solutions have been found with meta-heuristics (such as tabu search) that demand an impractical amount of computer time. Hence, our goal is to develop a procedure that can compete in speed with the simple heuristics and in quality with the complex meta-heuristics. In some of our experiments, we target low-density graphs because we believe these graph types can be found in practical applications but typically have not been

used for testing proposed procedures. Also, as shown in the comparative study by Jünger and Mutzel[12], the performance of existing heuristics seems to quickly deteriorate as the graphs become sparser. In particular, we adapt a greedy randomized search procedure (GRASP) in this context. In addition, we explore the adaptation of a search strategy called *path relinking* within the GRASP framework. The path relinking strategy has been suggested in connection with tabu search, however, to the best of our knowledge, it has not been implemented and thoroughly tested. Before describing our basic procedure and its variants, we provide background on both GRASP and path relinking in the following two sections.

1. GRASP

The GRASP methodology was developed in the late 1980s, and the acronym was coined by Tom Feo (Feo and Resende[8]). It was first used to solve computationally difficult set covering problems (Feo and Resende[7]). 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. Recently, GRASP has also been applied to a special type of edge crossing minimization problem (Resende and Ribeiro[18]).

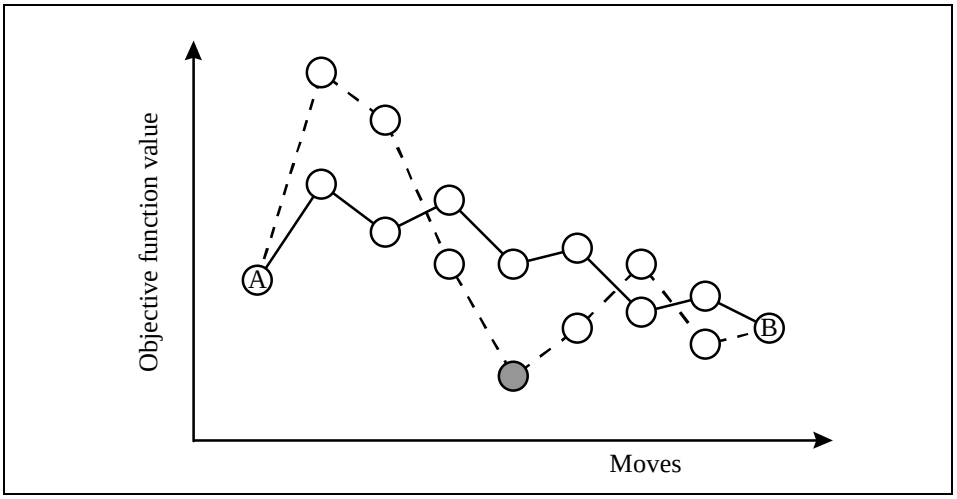
2. Path Relinking

Tabu search is by now a well-established meta-heuristic for optimization (Glover and Laguna[11]). One of the main goals in a tabu search design is to create a balance between search intensification and search diversification. Path relinking has been suggested as an approach to integrate intensification and diversification strategies (Glover and Laguna[11]). This approach generates new solutions by exploring trajectories that connect high-quality solutions — by starting from one of these solutions, called an *initiating solution*, and generating a path in the neighborhood space that leads toward the other solutions, called *guiding solutions*. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions.

The approach may be viewed as an extreme (highly focused) instance of a strategy that seeks to incorporate attributes of high quality solutions, by creating inducements to favor these attributes in the moves selected. However, instead of using an inducement that merely encourages the inclusion of such attributes, the path relinking approach subordinates all other considerations to the goal of choosing moves that introduce the attributes of the guiding solutions, in order to create a "good attribute composition" in the current solution. The composition at each step is determined by choosing the best move, using customary choice criteria, from the restricted set of moves that incorporate a maximum number (or a maximum weighted value) of the attributes of the guiding solutions.

The approach is called path relinking because in tabu search any two solutions are linked by a series of moves executed during the search. Figure 1 shows two hypothetical paths (i.e., a sequence of moves) that link solution A to solution B. The solid line indicates the original path followed by the "normal" operation of the procedure to move from A to B, while the dashed line depicts the relinking path. The paths are different because the move selection during the normal operation is generally "greedy" with respect to the objective function evaluation. For example, it is customary to adopt a move selection strategy that chooses the neighborhood move that minimizes (or maximizes) the objective function value in the local sense. During path relinking, however, the main goal is to incorporate attributes of the guiding solution (or solutions) while at the same time recording the objective function values.

Fig. 1 Path relinking illustration.



As shown in Figure 1, the purpose of performing the relinking moves is to find improved solutions that were not in the neighborhood of the solutions visited by the original path. The dark circle in Figure 1 represents a better solution than both A or B, considering a minimization problem.

3. GRASP Implementation

Before describing our implementation, we introduce some useful notation (Wilson[24]). Given a graph $G = (V,E)$, where $V = V_1 \cup V_2$, let $n_1 = |V_1|$, $n_2 = |V_2|$, $m = |E|$, and let $N(v) = \{w \in V \mid e = \{v, w\} \in E\}$ denote the set of neighbors of $v \in V$ (Jünger and Mutzel[12]). A solution is completely specified by a permutation π_1 of V_1 and a permutation π_2 of V_2, where $\pi_1(v)$ or $\pi_2(v)$ is the position of v in its corresponding layer. The degree of a vertex v with respect to the set of vertices V is $\rho(v,V)$.

As mentioned earlier, GRASP consists of a construction phase and an improvement phase. 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 improvement 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:

Construction Phase

This phase starts by creating a list of unassigned vertices ($U = U_1 \cup U_2$), which at the beginning consists of all the vertices in the graph (i.e., initially $U = V$). Also, the current position of each vertex is assigned a value of zero (i.e., $\pi_1(v) = 0 \ \forall \ v \in V_1$

and $\pi_2(v) = 0 \ \forall \ v \in V_2$). The first vertex v is randomly selected from all those vertices in U with maximum degree. That is, the first vertex is chosen from

$$U = \{ \ v \in V \mid \rho(v,V) = \rho_{max}\},$$

where $\rho_{max} = \max\{\rho(v,V) \ \forall \ v \in V\}$.

Once v is selected, U is updated by deleting v from its corresponding set (e.g., $U_1 = U_1 - \{v\}$ if $v \in V_1$ or $U_2 = U_2 - \{v\}$ if $v \in V_2$). 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 δ of the maximum degree of all the vertices in U. Vertex degree, in this case, is calculated with respect to the subgraph given by the partial solution obtained from previous vertex selections. That is, $U = \{ \ v \in U \mid \rho(v,V\text{-}U) \geq \delta\rho_{max}\}$, where $\rho_{max} = \max\{\rho(v,V\text{-}U) \ \forall \ v \in U\}$. Note that this is different from the first vertex selection, where maximum degree ρ_{max} is calculated with respect to all the vertices in the graph.

A selected vertex v is placed as prescribed by the barycenter calculation. 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:

$$bc(v) = \frac{\sum_{w \in K} \pi_2(w)}{|\hat{E}|}$$

Where $K = \{w \in V_2 \mid \pi_2(w) > 0$ and $\{v,w\} \in E\}$. (A similar calculation is carried out for a vertex $v \in V_2$.) Note that $bc(v)$ may be a fractional value and therefore rounding must be used to determine the exact position of v. We try the assignments $\pi_1(w) = \lceil bc(v) \rceil$ and $\pi_1(w) = \lfloor bc(v) \rfloor$, and select the best. If a previously chosen vertex already occupies the barycenter position of the most recently selected vertex (i.e., $\pi_1(w) = \lceil bc(v) \rceil$ or $\lfloor bc(v) \rfloor$ for $w \in V_1$), then vertex v is placed either one position "before" (i.e., $\lfloor bc(v) \rfloor$-1) or one position "after" (i.e., $\lceil bc(v) \rceil$+1) the barycenter (whichever produces the least number of crosses with respect to the partial construction). Once vertex v has been positioned in the partial solution, the vertex is deleted from the set U and the vertex degree calculations $\rho(v,V\text{-}U)$ are updated accordingly. The construction phase terminates after n_1+n_2 steps, when all vertices have been selected (i.e., when $U = \varnothing$).

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 $v \in V_1$ is selected, three moves are considered: (1) to insert the vertex one position before the barycenter (i.e., $\pi_1(v) = \lfloor bc(v) \rfloor - 1$), (2) to insert the vertex at the barycenter position (i.e., $\pi_1(v) = \lceil bc(v) \rceil$ or $\lfloor bc(v) \rfloor$), and (3) to insert the vertex one position after the barycenter (i.e., $\pi_1(v) = \lceil bc(v) \rceil + 1$). The vertex v is placed in the position that produces the maximum reduction in the number of crossings. (In our notation we assume that when the position of a vertex changes, the position of the other vertices are updated to preserve the correct relative ordering. In the actual implementation, however, we handle the position changes with an appropriate pointer structure.) If no reduction is possible, then the vertex is not moved. The vertex is removed from the set 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 n_1+n_2 trials. (Note that within the same improvement step, some vertices may be moved while others may stay in their original positions.) 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 $\pi_1(v) = \pi_1(w)+1$. This process is performed on each layer, according to the vertex order in the current solution, i.e., $\pi_1(v) = 1, \ldots n_1\text{-}1$ and $\pi_2(v) = 1, \ldots n_2\text{-}1$.

After a number of GRASP iterations, it is possible to estimate the percent improvement achieved by the application of the improvement phase and use this information to increase the efficiency of the search. Define the percent improvement in the GRASP iteration i as:

$$P(i) = \frac{S(i) - S^*(i)}{S(i)}$$

where $S(i)$ is the number of crossings in the solution at iteration i after the construction phase and before the improvement phase, while $S^*(i)$ is the number of crosses in the solution obtained after the improvement phase is performed in the same GRASP iteration. After n GRASP iterations, the following mean and standard deviation estimates of P can be calculated:

$$\hat{\mu}_P = \frac{\sum_{i=1}^{n} P(i)}{n}$$

$$\hat{\sigma}_P = \sqrt{\frac{\sum_{i=1}^{n} \left(P(i) - \hat{\mu}_P \right)^2}{n-1}}$$

Then, at a given iteration i, these estimates can be used to determine whether is "likely" that the improvement phase will be able to improve the current construction enough to produce a better solution than the current best (S_{best}). In particular, we calculate the minimum percentage improvement x that is necessary for a construction $S(i)$ to be better than S_{best}, as follows:

$$x > \frac{S(i) - S_{best}}{S(i)}$$

Therefore if $\frac{x - \hat{\mu}_P}{\hat{\sigma}_P} > \alpha$, then the construction corresponding to $S(i)$ is discarded and the improvement phase is not performed. The value of α is a search parameter, which represents a threshold on the number of standard deviations away from the estimated mean percent improvement that the procedure is allowed to tolerate at a given GRASP iteration. In Section 5, we perform a set of preliminary experiments to test the effect of different α values on solution quality and speed.

4. Path Relinking Implementation

In our implementation of path relinking, the procedure stores a small set of high quality (elite) solutions to be used for guiding purposes. Fluerent and Glover[9] have proposed the use of elite solutions within the GRASP framework. Specifically, after each GRASP iteration, the resulting solution is compared to the best three solutions found during the search. If the new solution is better than any one in the elite set, the set is updated. Instead of using attributes of all the elite solutions for guiding purposes, one of the elite solutions is randomly selected to serve as a guiding solution during the relinking process. The relinking in this context consists of finding a path between a solution found after an improvement phase and the chosen elite solution. Therefore, the relinking concept has a different interpretation within GRASP, since the solutions found from one GRASP iteration to the next are not linked by a sequence of moves (as in the case of tabu search). The relinking process implemented in our search may be summarized as follows:

> The set of elite solutions is constructed during the first three GRASP iterations. Starting with the fourth GRASP iteration, every solution after the improvement phase (called the *initiating solution*) is subject to a relinking process by performing moves that transform the initiating solution into the guiding solution (i.e., the elite solution selected at random). The transformation is relatively simple, at each step, a vertex v is chosen from the initiating solution (using the same probabilistic rule as in the improvement phase of GRASP) and is placed in the position occupied by this vertex in the guiding solution. So, if $\pi_1^g(v)$ is the position of vertex v in the guiding solution, then the assignment $\pi_1^i(v) = \pi_1^g(v)$ is made. (For the purpose of this discussion, we once again assume that an updating of the positions of vertices in V_1 of the intiating solution occurs.) After this is done, an expanded neighborhood from the current solution defined by $\pi_1^i(v)$ and $\pi_2^i(v)$ is examined. The expanded neighborhood consists of a sequence of position exchanges of vertices that are one position away from each other, which are performed until no more improvement (with respect to crossing minimization) can be found. This is the same exchange mechanism used at the end of the improvement phase of GRASP. Once the expanded neighborhood has been explored, the relinking continues from the solution defined by $\pi_1^i(v)$ and $\pi_2^i(v)$ before the exchanges were made. The relinking finishes when the initiating solution matches the guiding solution, which will occur after $n_1 + n_2$ relinking steps.

Note that two consecutive solutions after a relinking step differ only in the position of two vertices (after the assignment $\pi_1^i(v) = \pi_1^g(v)$ is made). Therefore, it is not efficient to apply the expanded neighborhood exploration (i.e., the exchange mechanism) at every step of the relinking process. We introduce the parameter β to control the application of the exchange mechanism. In particular, the

exchange mechanism is applied every β steps of the relinking process. We report on the effectiveness of the procedure with different values of β in the computational testing that follows.

5. Computational Experiments

The procedure described in the previous section was implemented in C, and all experiments were performed on a Pentium 166 MHz personal computer. Before testing the effectiveness of our procedure, we perform 4 preliminary experiments to explore the effect of changes in the three search parameters α, β and δ. 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 experiments we use the `random_bigraph` code of the Stanford GraphBase by Knuth[13] to generate 100 graphs with $n_1 = n_2 = 50$ and m = 50, 250, and 500, for a total of 300 instances. The experiments can be described as follows:

1) A termination criterion of 10 GRASP iterations without improvement is established (i.e., STOP = 10). We start with this criterion because it is in line with our objective of developing a procedure that can compete in both speed and solution quality. We consider the value δ = 2/3 and we test the effect of changing α by assigning the values 1, 2, 3 and 4. Results of this experiment are reported in Table 2.
2) We set α = 3, δ = 2/3 and try a termination criterion STOP = 5, 10, 15 and 20 GRASP iterations without improvement. Results of this experiment are reported in Table 3.
3) We set α = 3, STOP = 10 and try δ = 1, 1/3 and 2/3. Results of this experiment are reported in Table 4.
4) We set α = 3, δ = 2/3 and the termination criterion to 10. We then make β a function of the number of edges in the graph. We try the following values: b = 0.05*m*, 0.1*m*, 0.2*m* and 0.3*m*. Results of this experiment are reported in Table 5.

Table 2. Preliminary experiment (1).

	Average no. of crossings			**Average CPU seconds**			**% of Improvement phases skipped**		
α / *m*	**50**	**250**	**500**	**50**	**250**	**500**	**50**	**250**	**500**
1	6.48	6725.8	37752	0.13	0.91	4.72	48%	42%	33%
2	6.46	6717.8	37736	0.15	1.23	6.48	21%	17%	11%
3	6.15	6714.7	37712	0.15	1.48	7.43	5%	4%	2%
4	6.13	6712.9	37710	0.15	1.52	7.41	1%	1%	0%

Table 3. Preliminary experiment (2).

	Average no. of crossings			**Average CPU seconds**		
STOP / *m*	**50**	**250**	**500**	**50**	**250**	**500**
5	7.12	6762.5	37781	0.09	0.73	3.85
10	6.15	6714.7	37712	0.15	1.48	7.43
15	5.87	6691.6	37666	0.22	2.17	10.90
20	5.53	6674.4	37645	0.28	2.94	14.19

Table 4. Preliminary experiment (3).

	Average no. of crossings			**Average CPU seconds**		
δ	**50**	**250**	**500**	**50**	**250**	**500**
1/3	6.34	6698.2	37682	0.17	1.43	7.12
2/3	6.15	6714.7	37712	0.15	1.48	7.43

1	6.15	6717.1	37716	0.17	1.52	7.28

Table 5. Preliminary experiment (4).

	Average no. of crossings			**Average CPU seconds**			**% of PR phases where solution improved**		
β / m	**50**	**250**	**500**	**50**	**250**	**500**	**50**	**250**	**500**
0.05*m*	6.26	6633.2	37528	0.58	10.70	39.93	3%	80%	84%
0.1*m*	6.23	6640.5	37534	0.39	5.97	23.71	2%	60%	67%
0.2*m*	6.32	6647.4	37553	0.29	3.82	14.52	1%	43%	52%
0.3*m*	6.31	6656.1	37572	0.26	2.93	12.04	1%	36%	44%

Tables 2 shows that as α 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 3 also shows an improving trend in terms of solution quality as the procedure is allowed to run longer. By comparing the results of tables 2 and 3, it is clear that the effect in solution quality of increasing STOP is more significant than the effect of increasing α. Table 4 shows some mixed results. On one hand, the solution quality increases when δ is changed from 1/3 to 2/3 and m equals 50. On the other, the solution quality decreases when δ is changed from 1/3 to 2/3 and m equals 250 and 500. This seems to indicate that δ is sensitive to the density of the graph. Finally, Table 5 shows that in general, solution quality is higher for smaller values of β. One exception to this occurs when $\beta = 0.1m$ and $m = 50$. Note that the last three columns of Table 5 show the percentage of time that the path relinking (PR) phase improves upon the best solution found. As expected, this percentage goes down as the β value increases.

Given the results of the preliminary experimentation, we compare two versions of our procedure: 1) GRASP without path relinking (labeled GRASP) and 2) GRASP with path relinking (labeled PR). For GRASP, we use STOP = 10, $\alpha = 3$ and $\delta = 2/3$, while for PR we use STOP = 20, $\alpha = 3$, $\beta = 0.03m$ and $\delta = 2/3$. The first version is designed to compete in speed and generate high-quality solutions. The PR version is designed to find the best possible solutions (while somewhat sacrificing computational speed).

In the first experiment, we employ the problem instances reported in Table 5 of Jünger and Mutzel[12]. These instances have 10 vertices in each layer and a number of edges ranging from 10 to 90. There are a total of 900 instances (i.e., 100 instances for each edge density). We compare the performance of GRASP, BC*n* (the barycenter method started from $n = 1$, 10 and 15 random points), TS (tabu search as implemented in Martí[16]), and PR. We compare our procedure to BC, because independent studies by Jünger and Mutzel[12] and Martí and Laguna[17] have concluded that this method outperforms rival approaches based on relatively simple vertex positioning rules. We also compare against the TS procedure in Martí[16], because it consistently provides higher quality solutions than BC-based approaches (although TS requires more computational effort). Tables 6, 7 and 8 report, respectively, the average number of crossings, the average deviation from the optimal solutions (see OPT in Table 6), and the average CPU seconds

Table 6. Average number of crossings.

	Number of edges									
Method	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**Average**
OPT	0.29	11.62	56.60	146.89	276.78	463.17	698.35	1008.4	1405.6	451.96
BC1	2.21	20.00	66.51	159.10	288.07	475.52	711.08	1024.3	1422.1	463.21
BC10	0.43	12.78	57.94	148.28	277.74	464.22	699.14	1008.9	1406.5	452.88
BC15	0.37	12.31	57.18	147.64	277.38	463.97	699.02	1008.7	1405.8	452.49
GRASP	0.29	11.87	57.29	147.56	277.05	463.63	698.52	1008.5	1405.7	452.26
TS	0.29	11.62	56.60	146.89	276.78	463.17	698.35	1008.4	1405.6	451.96
PR	0.29	11.66	56.81	147.00	276.81	463.20	698.35	1008.4	1405.6	452.01

Table 7. Average percent deviation from optima.

	Number of edges									
Method	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**Average**
BC1	188.33	81.81	18.05	8.43	4.06	2.63	1.81	1.58	1.18	34.21
BC10	14.00	12.83	2.28	0.92	0.34	0.22	0.11	0.05	0.07	3.43
BC15	8.00	7.59	0.96	0.50	0.21	0.17	0.10	0.03	0.02	1.95
GRASP	0.00	2.09	1.17	0.45	0.09	0.10	0.02	0.01	0.01	0.44
TS	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
PR	0.00	0.37	0.35	0.07	0.01	0.01	0.00	0.00	0.00	0.09

Table 8. Average CPU seconds.

	Number of edges									
Method	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**Average**
BC1	0.00	0.00	0.01	0.00	0.00	0.01	0.01	0.01	0.02	0.01
BC10	0.02	0.02	0.04	0.05	0.07	0.09	0.12	0.14	0.16	0.08
BC15	0.02	0.04	0.06	0.08	0.11	0.14	0.17	0.21	0.24	0.12
GRASP	0.00	0.02	0.03	0.05	0.05	0.07	0.10	0.12	0.12	0.06
TS	0.43	0.82	1.51	2.50	3.93	8.12	17.50	39.35	60.81	15.00
PR	0.01	0.08	0.13	0.18	0.25	0.35	0.43	0.53	0.56	0.28

These tables show that the best solution quality is obtained by the TS method, which is able to match all known optimal solutions, while GRASP matches 750 and PR matches 866. However, TS employs more computational time than the other methods reported in these tables. GRASP is very competitive, considering its average percent deviation from optima of 0.44% achieved on an average of 0.06 seconds. PR achieves an improved average percent deviation of 0.09% with a modest increase in computer time (0.28 seconds).

In our second experiment we undertake to compare the performance of our proposed procedures (GRASP and the variant with path relinking) using relatively sparser graphs (as compared to those in the first experiment). In specific, we generate 900 additional instances with number of vertices in each layer equal to the number of edges. (All instances in this and subsequent experiments were also generated with the `random_bigraph` code.) This is the same generator used by Jünger and Mutzel[12] in their experiments, including the instances that we employed to construct Tables 6, 7 and 8. We generate 100 instances with number of edges ranging from 10 to 90 in increments of 10. We again use the solutions obtained by BC10 and TS for comparison purposes. The BEST row represents the minimum number of crossings found for each instance after running all procedures during the experiment. (We cannot assess how close the BEST values are from the optimal solutions, and we are only using these values as a way of comparing the methods.)

The issue of graph connectivity becomes relevant when dealing with sparse graphs. Sparse graphs will tend to have disjoint components. This was verified by an anonymous referee who generated graphs with $n_1 = n_2 = m = 90$ and found that approximately 30-35 vertices from each layer were of degree zero, another 10 to 20 vertices were of degree one, and the remainder of the graph were about a dozen larger components. In his (her) report, this referee also indicated that the largest component found in these experiments had about 28 vertices from each layer (Anonymous Referee[1]).

To address this situation, we have added a pre-processing phase that finds the disjoint components in the graph. After the preprocessing, the solution procedure (either BC, TS, GRASP or PR) is then applied to each component. We compare the performance of the procedures with and without the preprocessing. We indicate the use of the preprocessing by adding –PP to each procedure's label in tables 9, 10 and 11.

Table 9. Average number of crossings.

	Number of edges									
Method	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**Average**
BEST	0.29	1.07	2.25	3.46	4.88	6.60	8.00	9.24	13.41	5.47
BC10	0.43	3.43	8.03	15.28	22.88	32.48	42.38	51.87	65.17	26.88
GRASP	0.29	1.16	2.50	4.06	6.15	8.76	10.86	13.36	20.30	7.49
TS	0.29	1.07	2.39	4.91	7.88	12.02	15.82	20.14	27.40	10.21
PR	0.29	1.07	2.25	3.55	5.04	7.03	8.85	10.30	16.13	6.06
BC10-PP	0.31	1.48	3.63	5.72	8.97	12.00	15.44	17.70	26.12	10.15
GRASP-PP	0.29	1.11	2.31	3.73	5.46	7.68	9.29	11.11	16.96	6.44
TS-PP	0.29	1.07	2.25	3.52	4.97	6.85	8.30	9.73	13.96	5.66
PR-PP	0.29	1.07	2.28	3.52	4.92	6.83	8.30	9.38	13.91	5.61

Table 10. Average percent deviation from the best known.

	Number of edges									
Method	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**Average**
BC10	14.0	200.5	370.3	586.1	731.4	781.4	732.1	763.1	788.8	551.95
GRASP	0.0	3.3	10.4	16.0	27.2	35.8	39.8	49.4	63.6	27.27
TS	0.0	0.0	11.3	73.4	122.5	167.6	173.4	200.5	211.0	106.65
PR	0.0	0.0	0.0	1.4	2.2	4.7	10.3	11.0	23.1	5.85
BC10-PP	1.3	31.2	50.8	57.0	88.6	84.8	108.6	95.3	104.7	69.15
GRASP-PP	0.0	1.5	2.3	5.0	7.9	11.3	11.1	13.9	18.5	7.94
TS-PP	0.0	0.0	0.0	1.1	0.8	2.0	2.4	3.4	3.5	1.46
PR-PP	0.0	0.0	0.7	0.7	0.4	1.9	1.8	1.0	2.4	0.99

Table 11. Average CPU seconds.

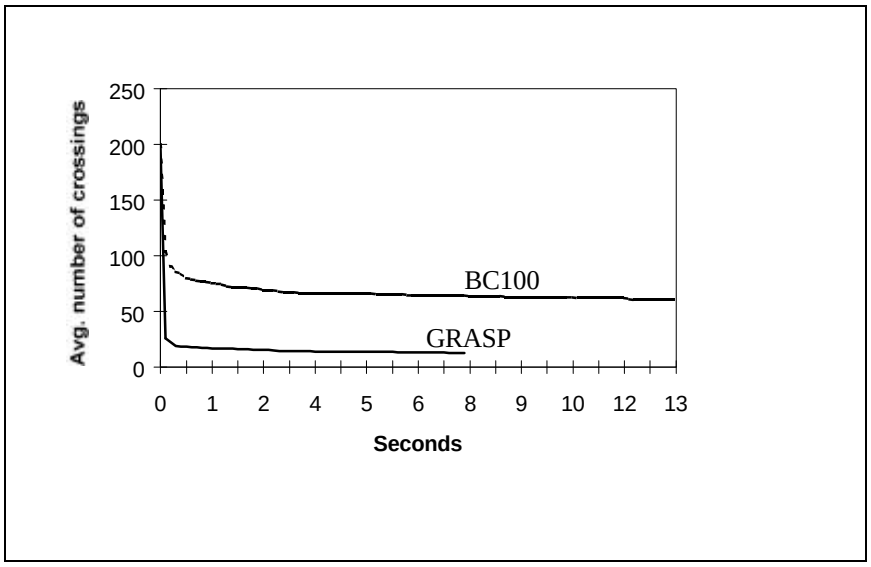
	Number of edges									
Method	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**Average**
BC10	0.01	0.04	0.07	0.13	0.23	0.35	0.50	0.69	0.94	0.33
GRASP	0.00	0.02	0.05	0.11	0.16	0.25	0.32	0.43	0.56	0.21
TS	1.65	3.24	5.87	8.53	11.62	15.14	19.08	21.86	27.09	12.68
PR	0.01	0.45	1.79	2.74	3.78	7.25	8.97	13.34	14.82	5.91
BC10-PP	0.01	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.11	0.05
GRASP-PP	0.00	0.01	0.01	0.01	0.02	0.03	0.03	0.05	0.06	0.03
TS-PP	0.19	0.50	0.82	1.17	1.53	1.86	4.15	2.50	3.09	1.76
PR-PP	0.00	0.02	0.03	0.08	0.14	0.39	0.44	0.88	1.61	0.40

Tables 9, 10 and 11 show that the preprocessing phase enhances the performance of all of the procedures. The path relinking with preprocessing achieves the best average deviation of less than 1%. The computational effort associated with the PR-PP variant is very reasonable (with a worst case average of 1.61 seconds). This experiment shows that the performance of the BC-based approaches is clearly inferior, with average deviations several orders of magnitude larger than those achieved by the GRASP-based and TS approaches. Also note the remarkable improvement of TS when the preprocessor is used, with the average deviation decreasing from 106.65% to 1.46%. In this sense, PR is more robust since its average deviation improves from 5.85% without the preprocessing to 0.99% with the preprocessing.

A third experiment was performed to assess the efficiency of our GRASP variants in denser graphs (relative to our second experiment). An additional set of 1,000 instances were generated with equal number of vertices in each layer (ranging from 10 to 100) and twice as many edges as vertices in each layer (hence, ranging from 20 to 200). Jünger and Mutzel[12] used these instances to construct Table 6 in their paper. We do not produce tables for this experiment, however, we report that the TS procedure outperforms both the BC and the GRASP variants. The average deviation from the best known values is 1.41% for the TS procedure, while PR obtains an average deviation of 8.96%, using similar computational time (i.e., 26 seconds for TS versus 22 seconds for PR). The GRASP approaches, however, were competitive with respect to the BC procedures, since these procedures resulted in an average deviation of 14.07%.

In our last experiment, we generate 100 instances with 100 vertices in each layer and 100 edges. The experiment has the goal of showing how the average solution obtained by GRASP improves over time, and to compare this average solution to the one obtained by BC100 (i.e., a barycenter approach that is started 100 times from a random solution). The results of this experiment are shown in Figure 2.

Fig. 2 GRASP and BC average solution over time.



The first average solution for BC100 has in fact 2,046 crossings, however, it has been deleted from Figure 2 for scaling purposes. It is obvious that the approaches are only comparable for very small computational times (a fraction of a second). GRASP clearly dominates the approach of starting BC from random solutions as the procedures are allowed to run longer.

6. Conclusions

We have developed a heuristic procedure based on the GRASP methodology to provide high quality solutions to the problem of minimizing straight-line crossings in a 2-layer graph. Our GRASP implementation was shown competitive in a set of problem instances for which the optimal solutions are known (Tables 6, 7 and 8). For a set of sparse instances, the proposed basic GRASP and its path relinking variant performed remarkably well (outperforming the best procedures reported in the literature). This result can be explained by the construction phase of GRASP, which places more importance to vertices with higher degree. This is not done by either BC or TS whose performance suffer in sparse graphs. We show, however, that a preprocessing phase improves the performance of all the procedures, including those that typically perform better with denser graphs (e.g., the tabu search implementation).

Overall, experiments with 3,200 graphs were performed to assess the merit of the procedure developed here. This extensive experimentation allows us to conclude that our GRASP implementation seems better suited for relatively low-density graphs. As the density increases, procedures such as the one based on the tabu search approach (Martí[16]) seem to be more appropriate. Our experiments show that TS is likely to outperform GRASP when $m \geq n_1+n_2$. Finally, our experience with the path relinking approach shows that, although computationally more expensive, this strategy was able to improve the performance of our basic GRASP implementation.

References

1. Anonymous Referee, 1998. "Referee Report #1."

2. Carpano, M. J., 1980. "Automatic Display of Hierarchized Graphs for Computer Aided Decision Analysis", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 10, No. 11, pp. 705-715.

3. Catarci, T., 1995. "The Assignment Heuristic for Crossing Reduction," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 25, No. 3.

4. Eades, P. and D. Kelly, 1986. "Heuristics for Drawing 2-Layered Networks", *Ars Combinatoria*, Vol. 21, pp. 89-98.

5. Eades, P., X. Lin and R. Tamassia, 1994. "An Algorithm for Drawing a Hierarchical Graph," Proceedings of the Second Canadian Conference on Computational Geometry, University of Otawa, pp. 1-18.

6. Eades, P., and N. C. Wormald, 1986. "The Median Heuristic for Drawing 2-layered Networks," Technical report 69, Department of Computer Science, University of Queensland.

7. Feo, T. and M. G. C. Resende, 1989. "A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem," *Operations Research Letters*, Vol. 8, pp. 67-71.

8. Feo, T. and M. G. C. Resende, 1995. "Greedy Randomized Adaptive Search Procedures," *Journal of Global Optimization*, Vol. 2, pp. 1-27.

9. Fleurent, C. and F. Glover, 1997. "Improved Constructive Multistart Strategies for the Quadratic Assignment Problem Using Adaptive Memory," University of Colorado.

10. Gansner, E. R., E. Koutsofios, S. C. North and K. P. Vo, 1993. "A Technique for Drawing Directed Graphs," *IEEE Transactions on Software Engineering*, Vol. 19, No. 3, pp. 214-230.

11. Glover, F. and M. Laguna, 1997. *Tabu Search*, Kluwer Academic Publishers, Boston.

12. Jünger M. and P. Mutzel, 1997. "2-Layer Straight Line Crossing Minimization: Performance of Exact and Heuristic Algorithms", *Journal of Graph Algorithms and Applications*, Vol. 1, No. 1, pp. 1-25.

13. Knuth, D. E., 1993. *The Stanford GraphBase: A Platform for Combinatorial Computing*, Addison Wesley, New York.

14. Laguna M., R. Martí and V. Valls, 1997. "Arc Crossing Minimization in Hierarchical Digraphs with Tabu Search", *Computers & Operations Research*, Vol. 24, No. 12, pp. 1175-1186.

15. Makinen, E., 1990. "Experiments on Drawing 2-Level Hierarchical Graphs", *International Journal of Computer Mathematics*, Vol. 36.

16. Martí R., 1998. "A Tabu Search Algorithm for The Bipartite Drawing Problem", *European Journal of Operational Research*, Vol. 106, pp. 558-569

17. Martí R. and M. Laguna, 1997. "Heuristics and Metaheuristics for 2-Layer Straight Line Crossing Minimization," Working Paper, University of Valencia, Spain

18. Resende, M. G. C. and C. C. Ribeiro, 1997. "A GRASP for Graph Planarization," *Networks*, Vol. 29, pp. 173-189.

19. Rowe, L. A., M. Davis, E. Messinger, C. Meyer, C. Spirakis and A. Tuan, 1987. "A Browser for Directed Graphs," *Software Practice and Experience*, Vol. 17, No. 1, pp. 61-76.

20. Sugiyama, K., 1987. "A Cognitive Approach for Graph Drawing," *Cybernetics and Systems: An International Journal*, Vol. 18, pp. 447-488.

21. Sugiyama, K., S. Tagawa and M. Toda, 1981. "Methods for Visual Understanding of Hierarchical System Structures," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 11, No. 2, pp. 109-125.

22. Valls, V., R. Martí and P. Lino, 1996. "A Branch and Bound Algorithm for Arc Crossing Minimization in Bipartite Graphs," *European Journal of Operational Research*, Vol. 90, pp. 303-319.

23. Valls, V., R. Martí and P. Lino, 1996. "A Tabu Thresholding Algorithm for Arc Crossing Minimization in Bipartite Graphs", *Annals of Operations Research*, Vol. 63, pp. 233-251.

24. Wilson, R. J., 1972. *Introduction to Graph Theory*, Academic Press, New York.