A HYBRID GRASP WITH PERTURBATIONS FOR THE STEINER PROBLEM IN GRAPHS*

CELSO C. RIBEIRO[†], EDUARDO UCHOA[‡], AND RENATO F. WERNECK[§]

Abstract. We propose and describe a hybrid GRASP with weight perturbations and adaptive path-relinking heuristic (HGP+PR) for the Steiner problem in graphs. In this multi-start approach, the greedy randomized construction phase of a GRASP is replaced by the use of several construction heuristics with a weight perturbation strategy that combines intensification and diversification elements, as in a strategic oscillation approach. The improvement phase circularly explores two different local search strategies. The first uses a node-based neighborhood for local search, while the second uses a key-path-based neighborhood. An adaptive path-relinking technique is applied to a set of elite solutions as a post-optimization strategy. Computational results on a broad set of benchmark problems illustrate the effectiveness and the robustness of our heuristic, which is very competitive when compared to other approximate algorithms.

1. Introduction. Let $G = (V, E)$ be a connected undirected graph, where V is the set of nodes and E denotes the set of edges. Given a non-negative weight function $w : E \rightarrow \mathbb{R}_+$ associated with its edges and a subset $X \subseteq V$ of terminal nodes, the Steiner problem in graphs (SPG) consists in finding a minimum weighted connected subtree of G spanning all terminal nodes in X. The solution of SPG is a Steiner minimum tree. The non-terminal nodes that end up in the Steiner minimum tree are called Steiner nodes.

The Steiner problem in graphs is a classic combinatorial optimization problem, see e.g. Hwang et al. [18], Maculan [23], and Winter [35] for surveys. Karp [19] showed earlier that its decision version is NP-complete in the general case. Applications can be found in many areas, such as telecommunication network design, VLSI design, and computational biology, among others.

Several heuristics are available for the approximate solution of the Steiner problem in graphs, see e.g. Duin and Voss [10], Hwang et al. [18], and Voss [34] for recent surveys. Effective preprocessing algorithms based on variable fixation theorems which allow strong graph reductions have been described and implemented by Duin [7], Duin and Volgenant [9], Koch and Martin [20], and Uchoa et al. [31], among others. The shortest-path heuristic of Takahashi and Matsuyama [30] is one of the most effective algorithms for computing greedy approximate solutions to SPG. Improvement heuristics based on the insertion of Steiner nodes were proposed by Minoux [25] and Voss [33]. An efficient local search procedure based on the exchange of key-paths was described by Verhoeven et al. [32]. We also find effective implementations of metaheuristics such as tabu search (Bastos and Ribeiro [1], Duin and Voss [10], Gendreau

*September 27, 2001

[†]Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, RJ 22453-900, Brazil. Work of this author was sponsored by FAPERJ and CNPq research grants 302281/85-1 and 202005/89-5. E-mail: celso@inf.puc-rio.br

[‡]Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, RJ 22453-900, Brazil. E-mail: uchoa@inf.puc-rio.br

[§]Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, RJ 22453-900, Brazil. E-mail: rwerneck@inf.puc-rio.br

et al. [14], and Ribeiro and Souza [28]), and GRASP (Martins et al. [24]), or greedy reconstruction algorithms such as the Pilot method (Duin and Voss [11]).

GRASP is a metaheuristic for combinatorial optimization [12], in which each iteration has two phases: construction and local search. The best solution over all iterations is returned as the result. In this paper, we describe HGP+PR, a hybrid GRASP procedure for the Steiner problem in graphs. This procedure follows the perturbation strategy proposed by Canuto et al. [3] for the prize-collecting Steiner tree problem and integrates other elements from different metaheuristics. In this multi-start approach, the construction phase of a GRASP is replaced by the combination of several construction heuristics with a weight perturbation strategy. A strategic oscillation scheme combining intensification and diversification elements is used for the perturbation of the original weights. The improvement phase circularly explores two different local search strategies: the first uses a node-based neighborhood for local search, while the second uses a key-path-based neighborhood. An adaptive path-relinking technique is applied to a set of elite solutions as an intensification strategy. The resulting heuristic is a state-of-the-art algorithmic scheme, combining successful existing techniques with new strategies and leading to a robust procedure capable of finding solutions of very high quality.

In the next section, we present the basic structure of the hybrid GRASP procedure based on weight perturbations. Section 3 describes the weight perturbation strategy. In Section 4 we describe the heuristics used in the construction phase and their combination. The local search procedure based on the cyclic investigation of node-based and key-path-based neighborhoods is described in Section 5. The hybrid path-relinking procedure and the construction of the pool of elite solutions are described in Section 6. Test problems and preprocessing results illustrating the effectiveness of reduction procedures are introduced in Section 7. Computational experiments on a large number of benchmark problems with different characteristics are reported in Section 8. We give computational evidence that each algorithmic feature contributes independently to improve solution quality. Results illustrating the effectiveness and the robustness of the new algorithm are also discussed. Comparisons with other approaches are made in Section 9. Concluding remarks are drawn in the last section.

2. Hybrid GRASP with perturbations. We apply the basic structure and the main concepts of a GRASP (greedy randomized adaptive search procedure) to devise a hybrid algorithm for the approximate solution of the Steiner problem in graphs. Each of its iterations begins by a construction phase, followed by a local search phase. Path-relinking is applied as a post-optimization intensification strategy.

In the construction phase, a feasible solution is built by a randomized greedy algorithm. The solutions generated by a GRASP construction are not likely to be locally optimal. Hence, it is almost always beneficial to apply local search as an attempt to improve each constructed solution. A local search algorithm works in an iterative fashion by successively replacing the current solution by a better one from its neighborhood. Success for a local search algorithm depends on the suitable choice of a neighborhood structure, efficient neighborhood search techniques, and the starting solution.

The construction phase of our hybrid GRASP is implemented using a number of greedy heuristics, combined with weight perturbations randomly recomputed at each iteration. The idea of introducing some noise into the original weights is similar to that in the so-called "noising method" of Charon and Hudry [4]. It adds more flexibility into algorithm design and may be even more effective than the greedy randomized construction of the basic GRASP procedure, in circumstances where the construction algorithms are not very sensitive to randomization. This is indeed the case for the shortest-path heuristic of Takahashi and Matsuyama, described in Section 4 and used as one of the main building blocks of our construction phase. Figure 2.1 illustrates the pseudo-code of a generic hybrid GRASP implementation. The customization of these generic steps into an approximate algorithm for the Steiner problem in graphs will be described in the following sections.

```
procedure HGP+PR
    Read and preprocess the input data;
    for k = 1,..., max_iterations do
        Apply a perturbation strategy to the original weights;
        Construct a greedy solution using the weights after perturbation;
        Apply local search to this solution using the original weights;
        Update the pool of elite solutions;
    end for;
    Apply path-relinking to the pool of elite solutions;
    Return the best solution found;
end Hybrid_GRASP_with_perturbations;
```

FIG. 2.1. *Pseudo-code of the generic hybrid GRASP with perturbations algorithm*

3. Weight perturbation strategy.

The construction phase of our hybrid procedure relies on weight randomization to build different solutions at different iterations. Edge weights are randomly perturbated at each iteration and one of the construction heuristics described in the next section is applied to the original graph with the modified weights.

The methods used for weight randomization incorporate learning mechanisms associated with intensification and diversification strategies originally proposed in the context of tabu search. Three distinct weight randomization methods (D, I, U) are applied along the algorithm. At a given GRASP iteration i, the modified weight w_e^i of each edge $e \in E$ is randomly selected from a uniform distribution between w_e and $r_i(e) \cdot w_e$, where the coefficient $r_i(e)$ depends on the selected weight randomization method applied at iteration i. Let $t_{i-1}(e)$ be the number of locally optimal solutions in which edge $e \in E$ appeared, after $i-1$ iterations of the hybrid GRASP procedure have been performed. Clearly, $0 \leq t_{i-1}(e) \leq i-1$. Table 3.1 displays how the coefficients $r_i(e)$ are computed by each randomization method.

TABLE 3.1
Maximum randomization coefficients

Method	$r_i(e)$
D	$1.25 + 0.75 \cdot t_{i-1}(e)/(i-1)$
I	$2 - 0.75 \cdot t_{i-1}(e)/(i-1)$
U	2

In method D, values of the coefficients $r_i(e)$ are larger for edges which appeared more frequently in previously found local optima. This scheme leads to a diversification strategy, since more frequently used edges are likely to be penalized with stronger augmentations. Contrarily, method I is an intensification strategy penalizing less frequent edges with larger coefficients $r_i(e)$. Finally, the third randomization method U uses a uniform penalization strategy, independent of frequency information.

The original weights without any penalization are used in the first three iterations, once for each construction heuristic. The weight randomization methods are then cyclically applied, one at each of the remaining iterations, starting with method I, next D, then U, then I again, and so on. The alternance between diversifying (method D) and intensifying (method I) iterations characterizes a strategic oscillation approach [16]. Computational results illustrating the effectiveness of this strategy will be reported in Section 8.1.

4. Construction phase. At each iteration i of the hybrid GRASP procedure, one of the heuristics described below is randomly selected to build a new solution in the construction phase. The constructive heuristics make use of the modified edge weights w_e^i obtained as described in the previous section.

The first one is the shortest-path heuristic (SPH) of Takahashi and Matsuyama [30], using a randomly selected terminal node as the root of the initial tree. At any iteration, let H be the set of terminal nodes already spanned by the current tree and $s \in X \setminus H$ be the closest non-spanned terminal node. The algorithm appends to the current tree all nodes in the shortest path from s to it and inserts s into H. The algorithm stops as soon as all terminal nodes have been spanned. Since the shortest-path heuristic performs exactly $|X| - 1$ iterations and Dijkstra's algorithm [5, 6] is applied at each iteration for the computation of shortest paths, the overall complexity of SPH is $O(|X||E| \log |V|)$.

The second heuristic resembles Kruskal's algorithm for finding the minimum spanning tree of a graph. Initially, there are $|X|$ connected components, each made up by a single terminal. In each iteration, we find the shortest path joining two connected components. The vertices in both components and those in this path are combined into a new, bigger component. The algorithm stops when a single component spanning all terminals is created. A straightforward implemenation of this heuristic runs in $O(|X||E| \log |V|)$ time.

Finally, the third algorithm is a minimum-spanning-tree-based heuristic. First, a minimum spanning tree of the original graph $G = (V, E)$ is computed. If all leaves are terminal nodes, then the algorithm returns the current tree. Otherwise, all degree one non-terminal nodes are removed from the tree and a new minimum spanning tree of the graph induced in G by the nodes remaining after pruning is computed. These steps are repeated until no further pruning of the current tree is possible. Since at most $|V| - |X|$ iterations are necessary and each minimum spanning tree can be computed in time $O(|E| \cdot \alpha(|E|, |V|))$ using Kruskal's algorithm [22] with edges preordered at each GRASP iteration according to their perturbated weights, then the overall complexity of the complete algorithm is $O(|V||E|\alpha(|E|, |V|))$, where $\alpha(.,.)$ is the inverse of the Ackerman function (see e.g. Cormen et al. [5]). This strategy is also applied together with the two previous heuristics.

We note that other construction heuristics could be used together with or instead of those outlined above, with their application alternated at different iterations. The motivation for combining different heuristics is the fact that they generate structurally different solutions, driving the local search procedure to explore different regions of the solution space which otherwise would not be investigated. This leads to a more robust algorithm, since a single heuristic is not likely to be appropriate to all classes of instances. Multiple heuristics can be chosen so as to deal more efficiently with a wider variety of classes. Computational results reported later in Section 8.2 have shown that, in the specific case of our hybrid GRASP with perturbations, the above combination was able to ensure enough solution diversity and consistently led the procedure to find quite good solutions.

5. Local search. As proposed by Martins et al. [24], two neighborhood definitions are combined to yield a more effective local search strategy: a node-based neighborhood and a key-path-based neighborhood. The local search procedure starts from the solution produced in the construction phase by applying one of the two local search algorithms described in Sections 5.1 and 5.2 below. Next, the other local search algorithm is applied to the locally optimal solution produced by the latter. These two local search algorithms are successively applied until one of them is not able to improve the solution produced by the other.

5.1. Node-based neighborhood. Let S^* be the set of Steiner nodes in the optimal solution of SPG. Then, the optimal solution T^* to SPG is a minimum spanning tree of the graph induced in G by the node set $S^* \cup X$. In fact, any solution of the Steiner problem may be characterized by the associated set of Steiner nodes and one of the corresponding minimum spanning trees. Therefore, the search for the Steiner minimum tree T^* can be reduced to the search for the optimal set S^* of Steiner nodes.

Let $S \subseteq V \setminus X$ be the set of Steiner nodes, to which we associate a solution of SPG given by a minimum spanning tree $T(S)$ of the graph induced in G by $S \cup X$. In the *node-based neighborhood*, the neighbors of this solution are defined by all sets of Steiner nodes that can be obtained either by adding to S a new non-terminal node, or by eliminating from S one of its Steiner nodes.

Each solution S has at most $|V| - |X|$ feasible neighbors. Nodes are examined for either insertion or elimination in circular order. In the case of a node $s \notin S$ candidate for insertion, we first check in $O(|V|)$ time if it may lead to a feasible solution by investigating if there are at least two edges connecting s with the nodes currently in S. Next, we compute a minimum spanning tree $T(S \cup \{s\})$ of the subgraph induced in G by $(S \cup \{s\}) \cup X$. There are $O(|V|)$ candidate edges to appear in $T(S \cup \{s\})$: those already in $T(S)$ and those connecting s to the nodes in $S \cup X$. If the edges of the graph and the edges adjacent to each node are stored in ascending order of original weights, then the new minimum spanning tree can be quickly found in time $O(|V| \cdot \alpha(|V|, |V|))$ using Kruskal's algorithm [5, 22]. In the case of a node $s \in S$ candidate for elimination, we calculate in $O(|E| \cdot \alpha(|E|, |V|))$ time the minimum spanning tree $T(S \setminus \{s\})$ of the subgraph induced in G by $(S \setminus \{s\}) \cup X$.

In both cases (insertion or elimination), all degree one non-terminal nodes of each neighbor are pruned. A feasible neighbor solution replaces the current one whenever its cost is not greater than that of the latter. The local search resumes from the next to be examined node. The procedure stops at a local optimum after a full pass of all nodes without improvement in the cost of the best solution. Since the current solution is replaced by the first improving neighbor, the order in which nodes are investigated may affect the solution found. So as to drive different applications of the local search to different solutions, at each iteration the nodes are investigated in a circular order defined by a different random permutation of their original indices.

5.2. Key-path-based neighborhood. A *key-node* is a Steiner node with degree at least three. A *key-path* is a path in a Steiner tree T whose extremities are either terminals or key-nodes, and whose intermediate nodes (if there are any) are Steiner nodes with degree two in T. A Steiner tree has at most $|X| - 2$ key-nodes and $2|X| - 3$ key-paths. A minimum Steiner tree consists of key-paths that are shortest paths between key-nodes or terminals. We use the key-path-based local search, proposed by Verhoeven et al. [32].

Let $T = \{l_1, l_2, \ldots, l_K\}$ be a Steiner tree, where each l_i, $i = 1, \ldots, K$, denotes a key-path. Also, let C_i and C'_i denote the two components that result from the removal of the key-path l_i from T. The *key-path-based neighborhood* of the current tree T is defined as the set $N(T) = \{C_i \cup C'_i \cup \mathrm{sp}(C_i, C'_i) \mid i = 1, \ldots, K\}$ of trees, where $\mathrm{sp}(C_i, C'_i)$ is the shortest path between C_i and C'_i. Observe that $C_i \cup C'_i \cup \{l_i\} = T$ and that $N(T)$ has at most $2|X| - 3$ elements. The replacement of a key-path in T can lead to the same Steiner tree if no shorter path exists. This implies that local minima have no neighbors and that the neighborhood is not connected.

As in the case of the node-based neighborhood, nodes are examined in a random, circular order. Since the current solution is always replaced by the first improving neighbor, this adds some diversity to the local search. We investigate all key-paths originating at each node, avoiding the investigation of paths already examined. Local search is performed by the computation of the shortest path between the two subsets of nodes that remain after the removal of each key-path. The search always moves to improving neighbors and stops after a full pass of all nodes without improvement in the weight of the best solution. The criteria for determining whether a neighbor is "improving" or not are:

1. If the weight of the new neighbor is strictly smaller than that of the current solution (i.e., if the weight of the new shortest path is smaller than that of the candidate key-path for elimination), it replaces the latter.
2. In case both paths have the same weight, we replace the current solution by the new neighbor if the new shortest path has more terminals as extremities than the original key-path. The rationale for this criterion is that a key-path with more terminals as extremities leads to the reduction of the degree of at least one key-node and, eventually, to a reduction in the number of key-nodes by a contraction of two key-paths. This longer key-path will have a greater chance of being replaced at some future iteration.
3. Finally, in case both the original key-path and the new shortest path have the same weight and the same number of terminals as extremities, we keep the solution whose key-path has more nodes. Longer key-paths are more likely to yield a reduction in solution weight at future iterations of the local search.

In case the two solutions are different, but equivalent with respect to these selection heuristics, we replace the current solution by the new neighbor. The above criteria are quite effective whenever there are multiple paths connecting the same pair of nodes, which happens very often in instances where many edges have the same weight. To make them even more effective, they are embedded in the shortest path algorithm itself. Whenever the algorithm has to choose between two paths with the same weight, it chooses the one connecting more terminal nodes or that with the largest number of nodes.

6. Path-relinking and elite solutions. Path-relinking is an improving strategy that generates new solutions by exploring trajectories that connect elite solutions [15, 17]. Starting from one or more of these solutions, paths in the solution space leading towards other guiding elite solutions are generated and explored in the search for better solutions.

Path-relinking makes use of a pool of elite solutions, whose construction is described below. Two alternative path-relinking strategies exploring this pool are described in Sections 6.2 and 6.3. An adaptive hybrid path-relinking strategy is presented in Section 6.4.

6.1. Pool construction. To implement a post-optimization strategy based on path-relinking, HGP+PR maintains and handles a pool with at most p elite solutions found along the search. We start with an empty pool of elite solutions. The solution obtained at the end of each GRASP iteration is considered as a candidate to be inserted into the pool if it is different from every other solution currently in the pool. If the pool already has p solutions and the candidate is better than the worst of them, then the former replaces the latter. If the pool is not full, the candidate is simply inserted.

At the end of all GRASP iterations, the pool contains the first generation of elite solutions. The relinking schemes described in the next sections are then applied to the solutions in the pool. The candidate solutions created by this process, also subject to the above criteria, will constitute a new generation of elite solutions. If the best solution in a generation does not improve the best solution previously found, the algorithm stops. Otherwise, path-relinking is applied once again, to the new generation.

This procedure is somewhat related with the idea of a genetic algorithm. The initial generation of elite solutions is used as the original population and the crossover operator corresponds to the application of path-relinking.

6.2. Path-relinking by complementary moves. This strategy is similar to that already used by Bastos and Ribeiro [1] in their reactive tabu search algorithm for the Steiner problem in graphs. For each pair of elite solutions (initial and guiding) in the pool, we first compute their symmetric difference, i.e., the set of all nodes which are Steiner nodes in one of them, but not in the other. This set defines the moves (insertions and/or eliminations) which should be applied to the initial solution until the guiding solution is attained. Starting from the initial solution, we always perform the best (most decreasing or least increasing) remaining move still in this list, until the guiding solution is attained. The best solution found along this trajectory is the candidate for insertion in the next generation pool.

6.3. Path-relinking by weight penalization. In this strategy, path-relinking is applied to each pair of elite solutions by means of a perturbation of edge weights (described below), followed by the application of the shortest-path heuristic using the modified weights. Once a solution has been constructed, the original weights are restored and the hybrid local search procedure described in Section 5 is applied. The resulting solution is the candidate for insertion in the next generation pool.

The weight perturbation function is devised so as to make the constructive heuristic find solutions preserving characteristics that are shared by both solutions in each pair. The weights of edges common to both solutions are kept unchanged. Weights of edges appearing in only one solution are multiplied by a strong penalty factor, randomly generated with a uniform distribution in the interval $[50, 100]$. Finally, weights of edges appearing in none of them are made artificially high (they are multiplied by 2000 in our implementation), in order to avoid their use by the construction procedure.

6.4. Adaptive hybrid path-relinking. The relative efficiency of these path-relinking schemes depends heavily on the instances they are applied to. To make the path-relinking procedure more effective and robust, independently of instance characterization, we propose the following hybrid adaptive procedure. In a first pass after the construction of the first generation pool, both schemes are applied to relink the best solution in the pool with every other elite solution in the pool. The path-relinking scheme with the smallest average computation time is then applied to the remaining pairs of elite solutions in the pool. Path-relinking by weight penalization is chosen whenever the average computation times are equal.

7. Test problems and preprocessing. The computational experiments reported in the next section were performed on three sets of test problems with quite different characteristics. All test instances are available from the SteinLib repository [21]. Different preprocessing strategies were applied to each set. The hybrid GRASP with perturbations procedure described in the previous sections was applied exclusively to the instances not solved to optimality by the preprocessing procedures themselves.

7.1. OR-Library instances. This set is made up by 20 instances from each of the series C, D, and E of problems available from the OR-Library [2], which are often used as benchmark instances for the Steiner problem. Each series contains randomly generated instances, with edge weights taken from a uniform distribution in the interval $[1, 10]$.

These instances may be significantly reduced by preprocessing. We applied the classical tests from Duin and Volgenant [9] and a new test proposed by Uchoa et al. [31]. An outline of the preprocessing procedure follows. We first apply the Nearest Special Vertex (NSV) test [9] (also known as Terminal Distance test [20], for the fixation of short edges at one), until no further reduction is possible. Next, we apply the Special Distance (SD) test [9] once for each edge (for the fixation of long edges at zero) and the simple Degree (D) test [9] (which is actually a group of tests consisting of the elimination of non-terminal nodes with degree one, fixation at one of edges adjacent to terminals with degree one, and collapse of edges adjacent to non-terminal nodes with degree two by a single edge). After that we exhaustively apply tests D, NSV, and SD, i.e., these tests are cyclically applied until no further reduction is possible. Finally, we apply tests D, NSV, and SD with Expansion (SDExp) [31] exhaustively.

7.2. VLSI instances. We consider a set of 116 instances defined over grid graphs with holes, extracted from real VLSI layout problems by Koch and Martin [20]. Instance sizes range from a few hundred vertices and edges, to larger problems with up to 36,711 vertices and 68,117 edges (alut2625), with the number of terminals varying from 10 to 2344.

VLSI instances cannot be significantly reduced by the traditional Duin and Volgenant's tests [9]. Uchoa et al. [31] presented new reduction tests particularly suited to such instances, by enhancing the existing tests with the idea of expansion introduced by Winter [36] for the rectilinear Steiner tree problem. While the traditional tests only remove edges satisfying some fixed condition, the tests with expansion adopt an "on-line theorem proving" approach. They start by assuming that a certain edge e belongs to every optimal solution and use the traditional conditions to derive a chain of logical implications, demonstrating that certain edges do or do not belong to an optimal solution. If a contradiction is reached, the initial assumption is proved to be false, i.e., there exists an optimal solution that does not use edge e and the latter may then be removed. Since the resulting SD with Expansion (SDExp) and Bottleneck Degree 3 with Expansion (BD3Exp) tests may be computationally expensive, we used two restricted, but faster versions of each of them, by limiting the amount of "look ahead". We begin with the restricted versions of these tests to find the "easy" reductions, leaving the complete and expensive tests to be applied to the already reduced, smaller instances. The outline of our preprocessing procedure can be described as follows. We start with test D and the most restricted versions of tests SDExp and BD3Exp exhaustively applied. Next, we apply (also exhaustively) test D along with the less restricted versions of tests SDExp and BD3Exp. Next, we apply the NSV test. Finally, we exhaustively apply tests D, SDExp (complete), BD3Exp (complete), and NSV until no further reductions can be obtained.

7.3. Incidence instances. The so-called incidence problems were created by Duin and Voss [10] so as to make currently known reduction tests ineffective, which was confirmed in practice by Koch and Martin [20]. These instances are divided into four series (i080, i160, i320, and i640), according to their number of vertices (80, 160, 320, and 640). Within each series, there are 20 different combinations of edge and terminal densities, each with 5 instances.

7.4. Reductions. In general, preprocessing helps the heuristics to find improved solutions in smaller computation times, due to variable fixations and graph reductions. In some extreme cases, preprocessing may even find by itself the optimal solutions to some instances. The preprocessing procedures outlined in Sections 7.1 and 7.2, based on the reduction tests proposed by Duin and Volgenant [9] and Koch and Martin [20], together with the enhanced tests described by Uchoa et al. [31], have been coded in C++ and compiled with GCC using the full optimization -O3 flag. They have been applied only to the OR-Library and VLSI instances, since the incidence instances are almost insensitive to reduction tests.

Computational results obtained for the 60 OR-Library instances and the 116 VLSI instances on a 350 MHz AMD K6-2 with 128 MB of memory are available from [29]. The preprocessing procedures are quite effective, being able to strongly reduce the number of vertices and edges of the original graphs. They are even able to solve to optimality by themselves alone three OR-Library instances (c20, d20, e20) and 39

VLSI instances.

In the next section, we report computational results obtained with the hybrid GRASP procedure described in Sections 2 to 6 for the instances not solved to optimality by preprocessing (57 reduced OR-Library instances, 77 reduced VLSI instances, and 400 original incidence instances).

8. Computational results. The hybrid GRASP procedure with weight perturbations and adaptive path-relinking was also coded in `C++` and compiled with `GCC` using the full optimization `-O3` flag. In this section, we report computational results obtained with the hybrid GRASP procedure on a 350 MHz AMD K6-2 with 128 MB of memory for all incidence instances and for the OR-Library and VLSI instances not solved to optimality after preprocessing. Computation times reported in this section do not include preprocessing.

We investigate the behavior of several components of the hybrid GRASP procedure and compare the results obtained by this new algorithm with respect to other heuristics in the literature. We note that the computational study reported below in Sections 8.1 to 8.4 should be considered much more as an illustration of the robustness and of the effectiveness of different strategies described throughout this work, instead of as a parameter tuning experiment. Final results for all instances are discussed in detail in Section 8.5, where the issues of solution quality and algorithm robustness are also addressed.

The most time-consuming instances were not tested in Sections 8.1, 8.2, and 8.4, where different variations of HGP+PR are studied. However, we do take these instances into account when comparing the final version of HGP+PR with other algorithms in the literature. The reduced set used in Sections 8.1, 8.2, and 8.4 contains 370 instances: all 57 OR-Library instances, 73 VLSI instances (the four largest ones were discarded), and the 240 incidence instances with up to 320 nodes (complete incidence instances were discarded).

8.1. Weight randomization strategies. In this section, we evaluate the effectiveness of the weight randomization strategy proposed in Section 3, showing that the combination of the three perturbation methods D, I, and U is quite robust and most often leads to better results than if they were otherwise used.

Table 8.1 reports results obtained after 128 hybrid GRASP iterations, followed by the application of the adaptive path-relinking strategy described in Section 6.4. The three heuristics described in Section 4 are used in the construction phase, combined as described in Section 3.

Each instance in the reduced set was run five times, with five different random seeds. We discarded all instances for which the seven weight perturbation methods led to the same final average solution over the five runs. We report three different statistics concerning the effectiveness of each strategy combining one or more weight perturbation methods. The first result is the average *relative error* obtained with the corresponding strategy over all instances tested. For each run, the relative error observed with some strategy is computed with respect to the best average solution value among those found for this instance with the seven different strategies compared. The second result is the *score* achieved by each strategy, considering all instances. For a given instance, the score of a strategy Y is defined as the number of strategies that found better average solutions than Y. In other words, if Y found the best solution, its score is 0; if it found the second best, it is 1; and so on, until 6. In case of ties, all strategies involved receive the same score, equal to the number of strategies strictly better than all of them. The score of a strategy is the sum of the scores obtained for all instances. Finally, the third result indicates the number of instances for which each perturbation strategy led to the best average solution value among those found by the seven combinations.

TABLE 8.1

Combinations of weight perturbation methods: results after 128 hybrid GRASP iterations followed by adaptive path-relinking, using the mix of three construction heuristics

Series	Strategy	Rel. error (%)	Score	Best
OR-Library (5 instances)	D	**0.016**	**5**	**3**
	DU	0.070	19	0
	I	0.050	13	2
	ID	0.076	11	2
	IDU	0.019	6	2
	IU	0.046	12	2
	U	0.062	15	0
VLSI (18 instances)	D	0.028	31	9
	DU	0.022	26	9
	I	0.085	78	2
	ID	0.045	43	8
	IDU	0.020	22	11
	IU	0.036	48	7
	U	**0.005**	**13**	**12**
Incidence (128 instances)	D	0.115	237	43
	DU	0.122	232	36
	I	0.149	312	26
	ID	0.122	297	28
	IDU	0.113	222	43
	IU	**0.086**	**191**	**46**
	U	0.00125	268	34
Overall (151 instances)	D	0.099	273	55
	DU	0.106	277	45
	I	0.136	403	30
	ID	0.110	351	38
	IDU	0.096	**250**	**56**
	IU	**0.077**	251	55
	U	0.106	296	46

For each class of instances, results in bold face indicate the best strategy with respect to each of the three evaluation criteria. Even though the combination IDU, proposed in Section 3, is not the one which led to the best results for each specific class of instances, it is most often the second best and quite close to the best one. This is the most robust combination of perturbation methods, leading to the best overall results, as Table 8.1 shows. This strategy will be maintained in the remaining of this computational study.

8.2. Combination of construction heuristics. We now evaluate the influence of the mix of heuristics used in the construction phase. Since Takahashi and Matsuyama's shortest-path heuristic (T) is by far the fastest and leads to the best individual results among the three heuristics presented in Section 4, it appears in all tested combinations. These combinations also involve the Kruskal-based (K) and the minimum-spanning-tree-based (M) heuristics.

The reduced set of 370 instances was used in this computational study. Table 8.2 reports results obtained after 128 hybrid GRASP iterations, followed by the application of the adaptive path-relinking strategy described in Section 6.4. The weight perturbation strategy adopted was IDU.

As before, each instance was ran five times and all instances for which the four tested combinations of construction heuristics led to the same final average solution over the five runs have been discarded. The same statistics described in the previous section assess the effectiveness of each combination of construction methods.

TABLE 8.2

Combinations of construction heuristics: results after 128 hybrid GRASP iterations followed by adaptive path-relinking, using the three randomization methods for weight perturbation

Series	Strategy	Rel. error (%)	Score	Best
OR-Library	T	0.060	7	2
(5 instances)	TK	0.098	9	1
	TM	0.072	5	1
	TMK	**0.021**	**2**	**4**
VLSI	T	0.054	24	4
(15 instances)	TK	0.061	19	6
	TM	**0.018**	22	5
	TMK	0.022	**10**	**9**
Incidence	T	0.098	122	35
(93 instances)	TK	**0.097**	116	**38**
	TM	0.126	136	27
	TMK	0.113	**113**	33
Overall	T	**0.091**	153	41
(113 instances)	TK	0.092	144	45
	TM	0.109	163	33
	TMK	0.097	**125**	**46**

Again, results in bold face indicate the best combination of heuristics with respect to each evaluation criterion, for each class of instances. The combination TMK using all three construction heuristics described in Section 4 is clearly the best for the OR-Library and the VLSI instances. Since it also led to some of the best results for the incidence instances, this combination will be selected and used in the sequel.

8.3. Effectiveness of the local search procedure. We now investigate and compare the behavior of different local search strategies, when applied to solutions found by SPH: node-based local search (N), path-based local search (P), hybrid local search starting with the node-based neighborhood (NP), and hybrid local search starting with the path-based neighborhood (PN). Detailed computational results for all instances in each series of test problems are reported in Table 8.3.

We report three different statistics concerning the effectiveness of each local search strategy. We discarded all instances for which none of the local search strategies have been able to improve the initial solution. The first result is the average improvement obtained with the corresponding local search strategy over all instances. For each instance, the improvement observed with some local search strategy is computed with respect to the initial solution found by the shortest-path heuristic. The second result is a score achieved by each local search strategy, computed as described in Section 8.1. The third result indicates the number of times each local search strategy led to the best solution among those found by the four strategies. As before, results in bold face indicate the best strategy for each evaluation criterion. Finally, we also give the average relative time of each local search strategy, with respect to the time taken by the pure node-based local search. To avoid the effect of small numbers, in this computation we discarded all instances for which the latter took less than 0.3 seconds.

TABLE 8.3
Local search strategies

Series	Strategy	Improv. (%)	Score	Best	Rel. time
OR-Library (43 instances)	N	1.094	54	21	1.
	P	2.149	65	17	0.129
	NP	**2.540**	**11**	**35**	1.473
	PN	2.429	14	34	0.882
VLSI (52 instances)	N	0.784	106	14	1.
	P	1.390	68	18	0.209
	NP	1.606	**15**	**42**	1.952
	PN	**1.648**	16	38	1.331
Incidence (359 instances)	N	11.818	453	135	1.
	P	2.818	635	112	1.604
	NP	**12.419**	**174**	231	1.317
	PN	**12.419**	224	**249**	2.829
Overall (454 instances)	N	9.538	613	170	1.
	P	2.591	768	147	1.380
	NP	**10.245**	**200**	308	1.381
	PN	10.239	254	**321**	2.562

The path-based local search is consistently faster than that using the node-based neighborhood for both the VLSI and the OR-Library instances. The complexity of evaluating each neighbor within both neighborhoods is roughly the same. However, the number of neighbors is generally smaller in the case of the path-based neighborhood, in which each solution has at most $\min\{|V|-1, 2|X|-3\}$ neighbors, while each solution has $|V|-|X|$ neighbors within the node-based neighborhood.

As a general rule, we may say that the larger is the graph density or the number of terminal nodes, the better should be the behavior of the node-based local search with respect to the path-based one in terms of solution quality. For very sparse instances, such as those arising from VLSI problems, the first approach is quite ineffective. In these cases, only very rarely a single node may be inserted or removed from the current solution without leading to an infeasible solution. Table 8.3 shows that the path-based neighborhood was more effective for VLSI and OR-Library instances, while the node-based local search led to better results for the incidence instances.

The results in Table 8.3 confirm that the circular hybrid local search strategies are much more effective, finding consistently better solutions than those using single node-based or path-based neighborhoods. Also, they are not too much slower than the others. Due to their effectiveness, we use the circular hybrid local search strategies combining the two neighborhoods in the experiments reported next. At each GRASP iteration, we randomly select between the two alternatives, starting from either the path-based or the node-based neighborhood.

8.4. Path-relinking strategies. We have described in Sections 6.2 and 6.3 two alternative path-relinking schemes, based respectively on complementary moves and on weight randomization. Sparse instances such as those arising from VLSI problems are often degenerated, in which case elite solutions may be quite different and too many complementary moves may exist. Therefore, path-relinking based on complementary moves is likely to be very time consuming for such instances. This observation led to the hybrid adaptive path-relinking strategy described in Section 6.4.

Four different path-relinking strategies are investigated: weight randomization (WR), one-way complementary moves starting from the best among each pair of elite solutions ($CM1$), two-way complementary moves starting from each solution in each pair of elite solutions ($CM2$), and hybrid adaptive path-relinking (HA). Detailed computational results for each series of test problems are reported in Table 8.4.

We report in this table three different statistics concerning the effectiveness of each path-relinking strategy. Starting from the reduced set of instances, we discarded those for which none of the path-relinking strategies was able to improve the solution found by the hybrid GRASP procedure. The first result we present is the average improvement obtained by the corresponding path-relinking strategy over the remaining instances. For each instance, the improvement achieved by a given strategy is computed with respect to the solution found after 128 iterations of the hybrid GRASP procedure. The second result is a score achieved by each path-relinking strategy when compared to the others, computed as described in Section 8.1. The third result indicates the number of times each path-relinking strategy led to the best solution among those found by the four strategies. Once again, results in bold face indicate the best strategy for each evaluation criterion. Finally, we also give the average relative time of each path-relinking strategy, with respect to the time taken by strategy $CM1$. To avoid the effect of small numbers, we discarded all instances for which the latter took less than 0.5 seconds.

Although $CM2$ takes approximately twice as much time as $CM1$, it led to solutions which are only marginally better. $CM1$ systematically leads to better solutions when it starts from the best among the two elite solutions involved in the path-relinking operation. This strategy explores much more carefully the neighborhood of the initial solution than that of the guiding solution. Starting from the best of them, we give the algorithm a better chance to explore more carefully the neighborhood of the most promising solution. We also note that $CM1$ is generally faster than WR, especially for incidence instances. However, although $CM1$ led to better solutions than WR for the incidence instances, the latter found much better solutions for both the OR-Library and VLSI instances.

As mentioned in Section 6.4, the adaptive path-relinking strategy selects the fastest strategy among $CM1$ and WR. However, Table 8.4 shows that usually the

TABLE 8.4
Path-relinking strategies

Instance	Strategy	Improv. (%)	Score	Best	Rel. time
OR-Library (7 instances)	WR	0.155	5	4	3.012
	$CM1$	0.136	8	3	1.
	$CM2$	0.149	3	4	1.970
	HA	**0.161**	**2**	**5**	1.605
VLSI (13 instances)	WR	**0.101**	**3**	**10**	1.135
	$CM1$	0.033	25	2	1.
	$CM2$	0.035	22	2	1.999
	HA	**0.101**	**3**	**10**	1.081
Incidence (75 instances)	WR	0.330	136	23	5.610
	$CM1$	0.411	78	31	1.
	$CM2$	**0.447**	**30**	**53**	1.904
	HA	0.419	50	39	1.599
Overall (95 instances)	WR	0.286	144	37	3.816
	$CM1$	0.339	111	36	1.
	$CM2$	**0.368**	**55**	**59**	1.944
	HA	0.356	**55**	54	1.455

chosen strategy is also the one leading to the best solution. Table 8.5 summarizes the choices made by the adaptive path-relinking strategy, along the five runs of each of the 370 instances involved in this experiment (for a total of 1850 runs in this table).

In conclusion, we note that (i) the scheme which leads to the smallest processing times is most often also that which finds the best solutions after the application of the complete path-relinking strategy and that (ii) the adaptive path-relinking strategy is quite effective and systematically leads to improvements in the solutions found at the end of the 128 hybrid GRASP iterations. The computational results reported in [29] and discussed in the forthcoming section were obtained with the hybrid adaptive path-relinking strategy.

TABLE 8.5
Choices made by the adaptive path-relinking algorithm

Series	WR	$CM1$
OR-Library (57 instances)	20.4%	79.6%
VLSI (73 instances)	59.7%	40.3%
Incidence (240 instances)	6.3%	93.7%
Overall (370 instances)	19.0%	81.0%

8.5. Solution quality and algorithm robustness. In this section, we assess the results obtained by applying the hybrid GRASP with perturbations and adaptive path-relinking to 57 OR-Library instances, 77 VLSI instances, and the 80 incidence instances tested in [20] (instances indexed by “1”). In these tests, we used the strategies selected in Sections 8.1 to 8.4. Detailed results are available in [29].

For each class of test problems, Figure 8.1 depicts the average deviation of the best solution found by iterations 1, 2, 4, 8, 16, 32, 64, and 128 with respect to the best solution found after path-relinking. Also, for all the above runs for which a provably optimal solution is known (all but seven incidence instances), we show in Figures 8.2 the increase in solution quality (percentual number of instances within some deviation with respect to the known optimal value) at the same iterations.

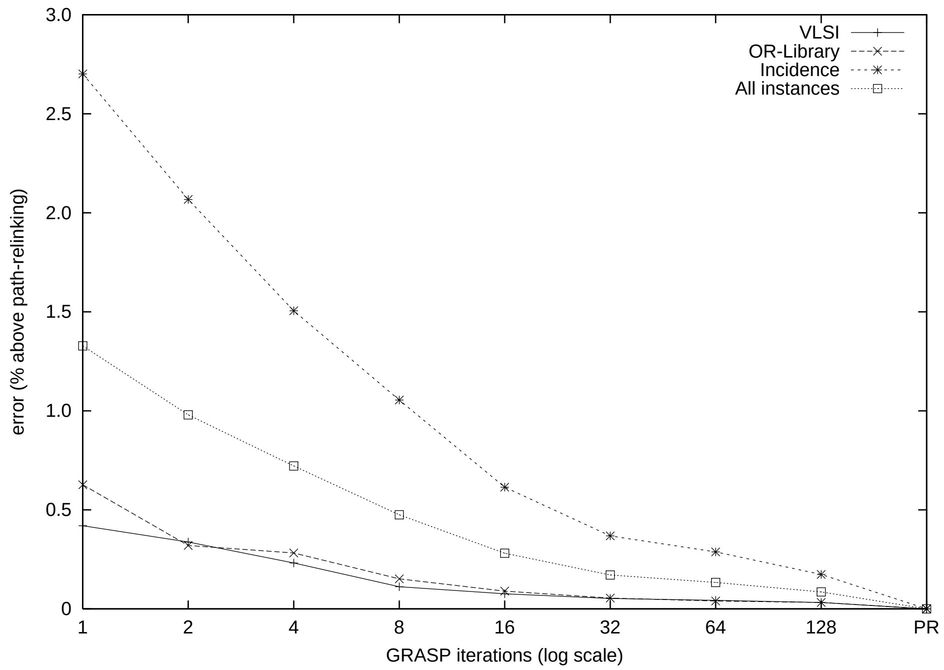

FIG. 8.1. *Solution improvement vs. iteration counter*

The results in Figures 8.1 and 8.2 show that solution quality steadly improves with the increase in the number of hybrid GRASP iterations. In approximately 90% of the cases, the first 16 iterations already led to quite good solutions, within 1% of the optimal value. Although the remaining iterations have a smaller contribution in the improvement of solution quality, they are particularly important in the case of the most difficult instances. The effectiveness of path-relinking may be clearly seen in Figure 8.1.

The two parameters with the strongest influence in solution quality are the number of iterations (which affects the best solution found after the hybrid GRASP itself) and the number of elite solutions kept in the pool (which affects the best solution found after path-relinking). Solution quality improves with the increase of any of them. To illustrate that the results obtained by HGP+PR can be further improved if more computation time is allowed, we display some additional results in Tables 8.6 and 8.7 for a restricted set of 15 instances for which the hybrid GRASP was not able to find the optimal solution after 128 iterations and using a pool of ten elite solutions. For each instance, Table 8.6 shows the value of the best solution found without path-relinking (HGP) and the corresponding computation time for both 128 and 256 GRASP iterations. As expected, computation times for 256 iterations are approximately twice as large as those for only 128 iterations. Improved solutions were found for four instances (alut2610, taq0903, i160-311, and i640-041). However, we note that: (i) path-relinking

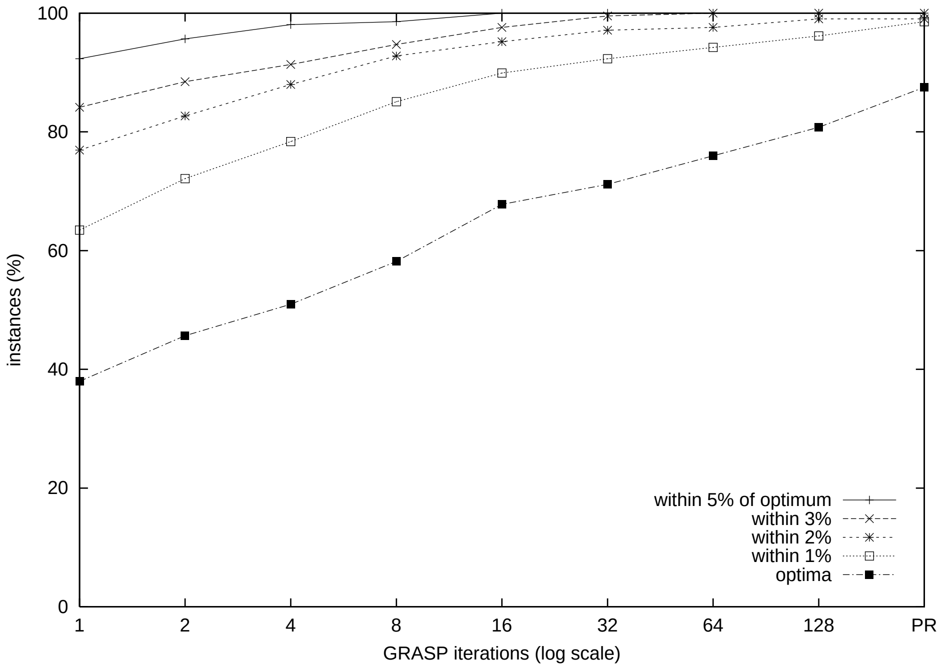

FIG. 8.2. *Solutions within some deviation from optimal value vs. iteration counter*

applied after only 128 iterations of GRASP was able to find by itself equivalent or even better solutions for the first three instances among the latter, and (ii) it required less additional time than the extra 128 iterations (see Table 8.7). Once again, this result illustrates the effectiveness of path-relinking.

Table 8.7 depicts the results obtained when the size of the pool of elite solutions is increased from ten to 20, with the number of hybrid GRASP iterations fixed at 128. For each instance, we give the value of the best solution found after path-relinking (HGP+PR) and the corresponding computation time, for pool sizes of both ten and 20 solutions. It is interesting to note that an increase in the size of the pool does not necessarily lead to a corresponding increase in computation time: in some cases the latter is even reduced, since the number of generations of elite solutions may be smaller. Moreover, improved solutions were found for ten instances out of the 15 selected ones when the pool size was increased from ten to 20. These results show that besides the robustness of HGP+PR, its effectiveness can be further increased if more computation time is allowed.

To give additional evidence concerning this conclusion, we performed long runs of HGP+PR on the 41 open instances from the incidence series i640 (640 nodes) and compared our results with the best known solutions according with the SteinLib [21]. We have loosely chosen higher values for the number of iterations and for the number of elite solutions in the pool. The numerical results are given in Table 8.8. For each instance, we first give its identification, the number of edges, and the number of terminal nodes, followed by the number of iterations and the number of elite solutions in the pool. Finally, we give the computation time in seconds on a Pentium II machine with a 400 MHz clock, the value of the best solution found by HGP+PR, and the

TABLE 8.6
Improved results with more hybrid GRASP iterations

	128 iterations		256 iterations	
Instance	HGP	Time (s)	HGP	Time (s)
d18	225	53.76	225	113.82
d19	311	50.79	311	107.93
e18	570	519.85	570	1050.96
alut2288	3846	57.87	3846	114.54
alut2610	12304	1764.64	12287	4421.75
taq0014	5335	124.46	5335	261.17
taq0903	5108	148.51	5107	324.84
i160-311	9257	7.71	9211	15.54
i320-041	1750	28.48	1750	58.93
i320-241	7111	61.49	7111	126.34
i320-311	18281	32.46	18281	65.90
i320-331	21727	17.53	21727	36.66
i640-041	1945	158.05	1897	329.01
i640-141	5288	251.60	5288	535.30
i640-201	16124	12.78	16124	25.62

TABLE 8.7
Improved results with a larger pool of elite solutions

	10 elite solutions		20 elite solutions	
Instance	HGP+PR	Time (s)	HGP+PR	Time (s)
d18	224	81.24	224	76.36
d19	311	65.61	310	70.06
e18	569	999.16	567	948.14
alut2288	3846	91.19	3846	83.98
alut2610	12284	2939.53	12269	6524.51
taq0014	5335	175.72	5326	236.33
taq0903	5102	463.88	5099	485.79
i160-311	9173	8.83	9167	12.26
i320-041	1750	30.07	1750	32.14
i320-241	7111	63.60	7093	67.74
i320-311	18100	38.51	17998	63.56
i320-331	21565	21.98	21565	39.51
i640-041	1945	165.96	1945	175.88
i640-141	5288	260.68	5252	284.24
i640-201	16124	13.73	16079	20.80

value of the best solution available from the SteinLib on May 1st, 2001. Algorithm HGP+PR improved the best known solution for 33 out of the 41 open instances and found the same solution value for four of them (the best solution values are indicated in bold face). The improved upper bounds helped the branch-and-ascent algorithm described in [26] to solve to optimality 21 of the 41 open instances. HGP+PR itself found the optimal solution for 14 of those instances, marked with an '*' in Table 8.8.

Table 8.8
Results for open instances of series i640

Instance	$\|E\|$	$\|X\|$	Iter	Pool	Time (s)	HGP+PR	SteinLib
i640-113	4135	25	256	120	1611	**6249** *	6298
i640-122	204480	25	128	10	1181	**4911** *	4976
i640-123	204480	25	128	10	1160	**4913** *	4942
i640-124	204480	25	128	10	1132	**4906** *	4955
i640-125	204480	25	128	10	1076	**4920** *	4958
i640-141	40896	25	256	70	6154	**5203**	**5203**
i640-142	40896	25	256	70	3240	5196	**5193** *
i640-143	40896	25	256	70	4713	5195	**5194** *
i640-144	40896	25	256	70	6150	5219	**5218**
i640-145	40896	25	256	70	6139	**5218** *	**5218** *
i640-211	4135	50	256	120	4378	**12023**	12025
i640-212	4135	50	256	120	3929	**11795**	11847
i640-213	4135	50	256	120	3820	**11879**	11910
i640-214	4135	50	256	120	4023	**11898**	**11898**
i640-215	4135	50	256	120	3924	**12103**	12141
i640-221	204480	50	128	10	1643	**9821** *	9917
i640-222	204480	50	128	10	1604	**9798** *	9957
i640-223	204480	50	128	10	1653	**9811** *	9927
i640-224	204480	50	128	10	1534	**9805** *	9938
i640-225	204480	50	128	10	1675	**9807** *	9933
i640-241	40896	50	256	70	8445	**10230** *	**10230** *
i640-242	40896	50	256	70	5263	**10195** *	10197
i640-243	40896	50	256	70	9969	**10215** *	10228
i640-244	40896	50	256	70	8406	**10250**	10263
i640-245	40896	50	256	70	6848	10254	**10234**
i640-311	4135	160	256	120	16480	**35813**	35999
i640-312	4135	160	256	120	19571	**35829**	36057
i640-313	4135	160	256	120	13171	**35543**	35654
i640-314	4135	160	256	120	12638	**35590**	35699
i640-315	4135	160	256	120	14377	**35868**	36003
i640-321	204480	160	128	10	3254	**31094**	31394
i640-322	204480	160	128	10	3385	**31068**	31353
i640-323	204480	160	128	10	3375	**31080**	31349
i640-324	204480	160	128	10	3356	**31092**	31613
i640-325	204480	160	128	10	2533	**31081**	31380
i640-334	1280	160	512	150	8617	**42783**	N/A
i640-341	40896	160	256	70	9187	**32054**	32128
i640-342	40896	160	256	70	11052	**31978**	32065
i640-343	40896	160	256	70	10106	**32032**	32068
i640-344	40896	160	256	70	9149	**31991**	32097
i640-345	40896	160	256	70	7032	**32015**	32074

N/A: not available
*: optimality proven by the branch-and-ascent algorithm in [26], other instances for which the optimal values are known are: i640-141 (5199), i640-144 (5205), i640-244 (10246), i640-245 (10223), and i640-334 (42768)

9. Comparison with other approaches. Algorithm HGP+PR outperforms or is competitive with recent implementations of other approximate algorithms and metaheuristics.

We first give in Table 9.1 some overall results concerning its application to the VLSI instances, which have not been systematically addressed in the literature by other approximate algorithms. For each series we give the number of test problems, the number of instances solved to optimality by HGP+PR, the percentual average relative error, and the computation time in seconds. HGP+PR found optimal solutions for 69 out of the 77 VLSI instances. Average times for series alue and alut are large due to four remarkably big instances.

TABLE 9.1
Results for VLSI instances

Series	Instances	Optimal	Rel. error (%)	Time (s)
alue	14	12	0.0226	3996.93
alut	8	4	0.0872	4747.55
diw	12	12	0	30.80
dmxa	8	8	0	1.62
gap	6	6	0	21.78
msm	19	19	0	4.44
taq	10	8	0.0228	172.32
VLSI	77	69	0.0161	1250.11

We now compare the hybrid GRASP with perturbations proposed in this paper, without (HGP) and with (HGP+PR) the adaptive path-relinking strategy, with the following approximate algorithms: the reactive tabu search algorithm of Bastos and Ribeiro [1], also without (RTS) and with (RTS+PR) path-relinking; the tabu search algorithms of Gendreau et al. [14] (F-tabu) and Ribeiro and Souza [28] (TS); the most effective variant (SV) of procedure SVERTEX+REBUILD of Duin and Voss [10]; and the most effective version (PH2) of the Pilot method (multiple runs of SVERTEX, each of them trying to include a new vertex into the solution), also proposed by Duin and Voss [11].

Run statistics for the above algorithms are summarized in Tables 9.2 to 9.4 for all 60 OR-Library instances and for all incidence instances with up to 320 nodes. The results reported for heuristics RTS and RTS+PR for the OR-Library instances are averages taken over ten runs for each instance. All other results reported here were obtained after only one run for each instance. Results for incidence instances with 640 nodes are provided in Section 8.5. They are not mentioned in this section because they were not addressed by the heuristics listed in the above paragraph.

The first two tables compare the algorithms in terms of solution quality. Table 9.2 displays the number of optimal solutions found by each algorithm, including those found by the preprocessing procedure, while Table 9.3 displays the average percentual relative errors (computed with respect to a lower bound for the instances for which an optimal solution is not known). Since one problem in each of the OR-Library series C, D, and E has been solved to optimality by the preprocessing procedure, the average relative errors observed by algorithms HGP and HGP+PR for these series were computed exclusively over the remaining 19 instances.

Table 9.2
Heuristics for SPG: optimal solutions found

Series	HGP	HGP+PR	RTS	RTS+PR	TS	FT	SV	PH2
C	19	20	17.7	19.8	17	18	—	—
D	18	18	14.2	16.7	9	14	—	—
E	17	19	11.6	13.6	7	10	—	—
OR-Lib	54	57	43.5	50.1	33	42	—	—
i080	94	97	96	99	89	—	—	96
i160	71	78	73	81	58	—	—	90
i320	62	68	≥56	≥64	≥39	—	—	≥82
Incidence	227	243	≥225	≥244	≥186	—	—	≥268

Table 9.3
Heuristics for SPG: average relative errors (%)

Series	HGP	HGP+PR	RTS	RTS+PR	TS	FT	SV	PH2
C	0.05	0	0.13	0.01	0.26	0.02	0.54	—
D	0.06	0.04	0.36	0.10	0.71	0.11	0.75	—
E	0.07	0.05	0.59	0.17	0.83	0.31	—	—
OR-Lib	0.06	0.03	0.36	0.09	0.60	0.15	—	—
i080	0.02	0.01	0.02	0.01	0.08	—	1.03	0.02
i160	0.25	0.13	0.19	0.12	0.35	—	0.98	0.04
i320	0.36	0.15	≤0.48	≤0.34	≤0.89	—	≤1.20	≤0.26
Incidence	0.21	0.10	≤0.23	≤0.16	≤0.44	—	≤1.07	≤0.11

For the sake of completeness, we also report in Table 9.4 the running times in seconds observed with the different algorithms. All algorithms apply preprocessing to the OR-Library instances. Running times refer only to the heuristics themselves, and do not include preprocessing. Since the results have been obtained in different machines, the computation times cannot be directly compared. Results for the hybrid GRASP with perturbations have been obtained on a 350 MHz AMD K6-2 with 128 MB of memory. The OR-Library instances have been processed by RTS and RTS+PR on a Pentium II machine with a 400 MHz clock. Algorithm SV and version PH2 of the Pilot method were run on a Pentium with a 90 MHz clock (running times for the latter were kindly provided by C. Duin [8]). Results for all other algorithms and instances have been obtained on a Sun UltraSPARC 1 machine.

The hybrid GRASP procedure HGP+PR compares favorably with the other heuristics for the OR-Library instances. Algorithm HGP+PR found optimal solutions for all but eight VLSI instances. In the case of the incidence instances, the hybrid GRASP procedure and the reactive tabu search algorithm found similar quality solutions. The PH2 implementation of the Pilot method performed remarkably well for the incidence instances in terms of solution quality, being at least as effective as HGP+PR. Since different machines were used, a direct comparison of the running times is impossible. However, Table 9.4 clearly shows that they grow faster for the PH2 implementation of the Pilot method than for HGP+PR. Also, Table 8.8 illustrates that long runs of HGP+PR achieve excellent results even on larger incidence

TABLE 9.4
Heuristics for SPG: average computation times in seconds

Series	HGP	HGP+PR	RTS	RTS+PR	TS	FT	SV	PH2
C	2.9	3.1	2.0	3.0	1.0	18.0	—	—
D	11.1	11.9	5.3	12.5	3.9	115.0	—	—
E	77.7	91.0	21.3	157.2	17.6	1474.0	—	—
OR-Lib	30.6	35.3	9.6	57.6	7.5	535.7	—	—
i080	2.7	2.8	5.8	6.2	3.3	—	0.4	15.3
i160	15.3	16.0	28.1	29.9	12.2	—	2.7	145.0
i320	104.1	107.9	162.6	168.1	50.5	—	14.8	1841.0
Incidence	40.7	42.2	65.5	68.1	22.0	—	6.0	667.1

instances. For instances with a larger number of nodes (such as the VLSI instances), PH2 tends to be much slower than HGP+PR.

One should be aware that despite these good results, elaborate metaheuristics are unlikely to be the most appropriate approach for many of the instances considered in this computational study. For "easy" instances, exact algorithms such as those described in [20, 26, 27, 31] are already able to find optimal solutions and prove their optimality in reasonable computation times, often smaller than those observed for metaheuristics. However, one might come across more challenging instances for which exact algorithms do not have such a satisfactory performance, and heuristics are probably the best approach. This is indeed the case for some of the instances in our test bed: the optimal values are still unknown for 30 of the instances in Table 8.8. For others, the amount of time required by current exact algorithms is hardly reasonable. Metaheuristics can also be used to accelerate exact algorithms through the computation of tighter primal bounds, as it was done with the branch-and-ascent algorithm [26] used to solve to optimality all formerly open instances of series i320 and 11 instances of series i640.

10. Concluding remarks. We presented a hybrid GRASP with perturbations and adaptive path-relinking algorithm (HGP+PR) for the Steiner problem in graphs. This algorithm incorporates several features of different metaheuristics, such as memory-based construction procedures, strategic oscillation driven by weight perturbations, variable neighborhoods, evolutionary population methods, and path-relinking. We have given computational evidence that each of these features contributes independently to improve solution quality and, consequently, to the effectiveness of the whole algorithm. Our implementation choices were supported by comprehensive computational experiments, encompassing a much broader and more complete set of test problems than other studies involving heuristics that we are aware of.

The substitution of the greedy randomized construction phase of a GRASP by a purely greedy construction using randomized weights had already led to sound computational results for the prize-collecting Steiner tree problem in [3]. The numerical results in this paper illustrate once again that the application of this idea to other problems should be pursued. We have also shown that the use of intelligent construction procedures can improve upon memoryless approaches. We have extended one step further the ideas originally discussed by Glover and Fleurent [13], by proposing the combination of a variety of construction algorithms integrated within a strategic oscillation scheme driven by the weight perturbations. We have also shown that perturbations can be used to implement easily and effectively a wide variety of algorithmic features, such as randomized greedy heuristics; intensification, diversification, and strategic oscillation strategies; and path-relinking.

We note that path-relinking is a quite effective strategy to improve the solutions found by metaheuristics such as GRASP or tabu search [1]. Path-relinking improved the results obtained by the hybrid GRASP with perturbations in terms of both the number of optimal solutions found and the average deviation from the optimal value. The adaptive path-relinking scheme proposed in this work also played a major role in the robustness of the complete algorithm.

Algorithm HGP+PR outperforms or is competitive with recent implementations of other approximate algorithms and metaheuristics. We have also shown that besides the robustness of the HGP+PR algorithm, its effectiveness can be increased even further if more computation time is allowed. With this respect, algorithm HGP+PR improved the best known solution for 33 out of the 41 instances of series i640 that were open in the time of writing.

REFERENCES

[1] M.P. Bastos and C.C. Ribeiro, "Reactive tabu search with path-relinking for the Steiner problem in graphs", in *Essays and Surveys in Metaheuristics* (C.C. Ribeiro and P. Hansen, eds.), pages 39–58, Kluwer, 2001.

[2] J.E. Beasley, "OR-Library: Distributing test problems by electronic mail", *Journal of the Operational Research Society* 41 (1990), 1069–1072.

[3] S.A. Canuto, M.G.C. Resende, and C.C. Ribeiro, "Local search with perturbations for the prize-collecting Steiner tree problem", *Networks* 38 (2001), 50–58.

[4] I. Charon and O. Hudry, "The noising method: A new method for combinatorial optimization", *Operations Research Letters* 14 (1993), 133–137.

[5] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to algorithms*, MIT Press, 1990.

[6] E.W. Dijkstra, "A note on two problems in connexion with graphs", *Numerische Mathematik* 1 (1959), 269–271.

[7] C.W. Duin, "Steiner's problem in graphs: Approximation, reduction, variation", Doctorate Dissertation, Institute of Actuarial Science and Economics, University of Amsterdam, 1994.

[8] C.W. Duin, personal communication, 2001.

[9] C.W. Duin and A. Volgenant, "Reduction tests for the Steiner problem in graphs", *Networks* 19 (1989), 549–567.

[10] C.W. Duin and S. Voss, "Efficient path and vertex exchange in Steiner tree algorithms", *Networks* 29 (1997), 89–105.

[11] C.W. Duin and S. Voss, "The Pilot method: A strategy for heuristic repetition with application to the Steiner problem in graphs", *Networks* 34 (1999), 181–191.

[12] T.A. Feo and M.G. Resende, "Greedy randomized adaptive search procedures", *Journal of Global Optimization* 6 (1995), 109–133.

[13] C. Fleurent and F. Glover, "Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory", *INFORMS Journal on Computing* 11 (1999), 189-203.

[14] M. Gendreau, J.-F. Larochelle, and B. Sansó, "A tabu search heuristic for the Steiner tree problem", *Networks* 34 (1999), 163-172.

[15] F. Glover, "Tabu search and adaptive memory programing - Advances, applications and challenges", in *Interfaces in Computer Science and Operations Research* (R.S. Barr, R.V. Helgason, J.L. Kennington, eds.), 1–75, Kluwer, 1996.

[16] F. Glover, "Multi-start and strategic oscillation methods - Principles to exploit adaptive memory", *Computing Tools for Modeling, Optimization and Simulation: Interfaces in*

Computer Science and Operations Research (M. Laguna and J.L. Gonzáles-Velarde, eds.), 1–24, Kluwer, 2000.

[17] F. Glover, M. Laguna, and R. Martí, "Fundamentals of scatter search and path relinking", *Control and Cybernetics* 39 (2000), 653–684.

[18] F.K. Hwang, D.S. Richards, and P. Winter, *The Steiner tree problem*, North-Holland, Amsterdam, 1992.

[19] R.M. Karp, "Reducibility among combinatorial problems", in *Complexity of Computer Computations* (E. Miller and J.W. Thatcher, eds.), 85–103, Plenum Press, New York, 1972.

[20] T. Koch and A. Martin, "Solving Steiner tree problems in graphs to optimality", *Networks* 32 (1998), 207–232.

[21] T. Koch, A. Martin, and S. Voss, "SteinLib Testdata Library", online document at `http://elib.zib.de/steinlib/steinlib.html`, last visited on May 1st, 2001.

[22] J.B. Kruskal, "On the shortest spanning tree of a graph and the traveling salesman problem", in *Proceedings of the American Mathematical Society* 7 (1956), 48–50.

[23] N. Maculan, "The Steiner problem in graphs", in *Surveys in Combinatorial Optimization* (S. Martello, G. Laporte, M. Minoux, and C.C. Ribeiro, eds.), *Annals of Discrete Mathematics* 31 (1987), 185–212.

[24] S.L. Martins, P. Pardalos, M.G. Resende, and C.C. Ribeiro, "A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy", *Journal of Global Optimization*, 2000, to appear.

[25] M. Minoux, "Efficient greedy heuristics for Steiner tree problems using reoptimization and supermodularity", *INFOR* 28 (1990), 221–233.

[26] M.V. Poggi de Aragão, E. Uchoa, and R.F. Werneck, "Dual heuristics on the exact solution of large Steiner problems", *Electronic Notes in Discrete Mathematics* 7 (2001), online document at `http://www.elsevier.nl/locate/endm`.

[27] T. Polzin and S. Vahdati, "Improved algorithms for the Steiner problem in networks", *Discrete Applied Mathematics*, 112 (2001), 263-300.

[28] C.C. Ribeiro and M.C. Souza, "Tabu search for the Steiner problem in graphs", *Networks* 36 (2000), 138–146.

[29] C.C. Ribeiro, E. Uchoa, and R.F. Werneck, "Detailed computational results for algorithm HGP+PR", online document, `http://www.inf.puc-rio.br/~celso/tabs-hgppr.ps`.

[30] H. Takahashi and A. Matsuyama, "An approximate solution for the Steiner problem in graphs", *Math. Japonica* 24 (1980), 573–577.

[31] E. Uchoa, M.V. Poggi de Aragão, and C.C. Ribeiro, "Preprocessing Steiner problems from VLSI layout", submitted for publication, 1999.

[32] M.G.A. Verhoeven, M.E.M. Severens, and E.H.L. Aarts, "Local search for Steiner trees in graphs", *Modern Heuristics Search Methods* (V.J. Rayward-Smith et al., eds.), 117–129, Wiley, 1996.

[33] S. Voss, *Steiner-Probleme in Graphen*, Anton Hain, 1990.

[34] S. Voss, "Steiner's problem in graphs: Heuristic methods", *Discrete Applied Mathematics* 40 (1992), 45–72.

[35] P. Winter, "Steiner problem in networks: A survey", *Networks* 17 (1987), 129–167.

[36] P. Winter, "Reductions for the rectilinear Steiner tree problem", *Networks* 26 (1995), 187–198.