A GRASP FOR GRAPH PLANARIZATION*

MAURICIO G.C. RESENDE† AND CELSO C. RIBEIRO‡

Abstract. A greedy randomized adaptive search procedure (GRASP) is a metaheuristic for combinatorial optimization. In this paper, we describe a GRASP for the graph planarization problem, extending the heuristic of Goldschmidt and Takvorian [*Networks*, v. 24, pp. 69–73, 1994]. We review basic concepts of GRASP: construction and local search algorithms. The implementation of GRASP for graph planarization is described in detail. Computational experience on a large set of standard test problems is presented. On almost all test problems considered, the new heuristic either matches or finds a better solution than previously described graph planarization heuristics. In several cases, previously unknown optimal solutions are found.

Key words. Combinatorial optimization, graph planarization, local search, GRASP, graph drawing, probabilistic algorithm, computer implementation

1. Introduction. A graph is said to be *planar* if it can be drawn on the plane in such a way that no two of its edges cross. Given a graph $G = (V, E)$ with vertex set V and edge set E, the objective of *graph planarization* is to find a minimum cardinality subset of edges $F \subseteq E$ such that the graph $G' = (V, E \setminus F)$ resulting from the removal of the edges F from G, is planar. This problem is also known as the *maximum planar subgraph* problem. A *maximal planar subgraph* is a planar subgraph $G' = (V', E')$ of $G = (V, E)$, such that the addition of any edge $e \in E \setminus E'$ to G' destroys the planarity of the subgraph.

Applications of graph planarization include graph drawing, such as in CASE tools [29], automated graphical display systems, and numerous layout problems, such as circuit layout, and layout of industrial facilities [13]. A survey of some of these applications is given in Mutzel [21].

Graph planarization is known to be NP-hard [20] and most algorithms to date attempt to find good approximate solutions. Several graph planarization heuristics have been proposed in the literature. Jayakumar *et al.* [15] describe two $O(|V|^2)$ planarization algorithms, based on the Lempel, Even, and Cederbaum planarity testing algorithm [19]. The first algorithm constructs a spanning planar subgraph of a given non-planar graph by embedding one vertex at a time and, at each step, adds the largest set of edges that does not lead to a non-planar graph. The second algorithm starts from a biconnected spanning planar subgraph and constructs a maximal planar subgraph containing it. Cai, Han, and Tarjan [3] proposed an $O(|E| \log |V|)$ algorithm for finding a maximal planar subgraph based on the Hopcroft and Tarjan planarity testing algorithm [14]. Another $O(|E| \log |V|)$ algorithm was also proposed by Di Battista and Tamassia [7], based on an $O(\log |V|)$ time-per-operation strategy to the problem of maintaining a planar graph under edge additions.

Cimikowski [4] proposed a heuristic based on finding, for each biconnected component of a non-planar graph, a pair of edge-disjoint spanning trees whose union is planar. Although no computational results are given by the author, the main interest

*Technical report version of paper published in *Networks*, vol. 29, pp. 173–189, 1997 – Latest version can be found in URL = `ftp://www.research.att.com/~mgcr/doc/gmpsg.ps.Z`

†Information Sciences Research Center, AT&T Research, Murray Hill, NJ 07974 USA. e-mail: `mgcr@research.att.com`

‡Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, RJ 22453 Brazil. e-mail: `celso@inf.puc-rio.br`. Work of this author was sponsored by the Brazilian National Research Council for Scientific and Technological Development, in the framework of project *ProTeM*.

of this approach is due to the fact that, under certain conditions, the number of edges of the generated planar subgraph is at least 2/3 of the optimum.

Takefuji and Lee [26] described a parallel heuristic for planarization based on neural networks. They use an arbitrary sequencing of the vertices, placing them along a line, followed by the use of a neural network for determining two sets of edges that may be represented without crossings above and below that line, respectively. The authors claimed superior performance with respect to previously published algorithms [27]. The two-phase approach of Takefuji and Lee was recently extended and improved by Goldschmidt and Takvorian [11]. We describe this approach in more detail in Section 2. Some components of this heuristic are used in the construction of the greedy randomized adaptive search procedure (GRASP) introduced in this paper.

Jünger and Mutzel [16, 17, 21] designed an exact branch-and-cut algorithm for the maximum planar subgraph problem using facet-defining inequalities. A heuristic based on the branch-and-cut algorithm stops with a feasible, though not necessarily optimal, solution. Their code has been used to find good approximate solutions (in many cases provably optimal solutions) for sparse and very dense graphs having up to 200 vertices or 700 edges.

Cimikowski [6] performed an empirical evaluation of heuristics for the graph planarization problem. Several heuristics were tested on a large and comprehensive set of test problems. These included random graphs with unknown maximum planar subgraph size, non-planar graphs containing a maximum planar subgraph of size $3|V|-6$, random Hamiltonian non-planar graphs, and a few special graphs already considered in the literature or possessing interesting structures or relevant applications. Cimikowski's experiments show that the branch-and-cut heuristic of Jünger and Mutzel consistently outperformed the other heuristics for the graphs tested. The two-phase heuristic of Goldschmidt and Takvorian [11] markedly outperforms the remaining in terms of solution quality, although its running time makes it prohibitive for very large graphs. If the computation time is critical, then the approaches based on planarity testing [15] and edge embedding [3] are recommended.

We next describe the organization of this paper. In Section 2 we review the two-phase heuristic of Goldschmidt and Takvorian, which we refer to in the description of our GRASP strategy. Section 3 describes the GRASP for graph planarization. A local search heuristic that seeks to enlarge a planar subgraph is described in Section 4. In Section 5, we present computational results on instances used in [6, 11]. Concluding remarks are made in Section 6.

2. A Two-Phase Heuristic. In this section, we review the main components of GT, the two-phase heuristic of Goldschmidt and Takvorian [11]. The first phase of GT consists in devising a sequence Π of the set of vertices V of the input graph G. The vertices of G are placed on a line according to the sequence Π obtained in the first phase. Let $\pi(v)$ denote the relative position of vertex $v \in V$ within vertex sequence Π. Furthermore, let $e_1 = (a, b)$ and $e_2 = (c, d)$ be two edges of G, such that, without loss of generality, $\pi(a) < \pi(b)$ and $\pi(c) < \pi(d)$. These edges are said to *cross* with respect to sequence Π if $\pi(a) < \pi(c) < \pi(b) < \pi(d)$ or $\pi(c) < \pi(a) < \pi(d) < \pi(b)$. The second phase of GT partitions the edge set E of G into subsets $\mathcal{B}$, $\mathcal{R}$, and $\mathcal{P}$ in such a way that $|\mathcal{B} + \mathcal{R}|$ is large (ideally maximum) and that no two edges both in $\mathcal{B}$ or both in $\mathcal{R}$ cross with respect to the sequence Π devised in the first phase.

Goldschmidt and Takvorian note that if the vertex sequence Π used in the second phase corresponds to a Hamiltonian cycle in a maximum planar subgraph H of G or in some planar edge-augmentation of H, then the planar subgraph it yields has at

procedure `FirstPhaseGT`(V,E,Π)
    $\underline{d} = \min_{v \in V}\{\deg_G(v)\}$;
    RCL $= \{v \in V \ : \ \deg_G(v) = \underline{d}\}$;
    Select v_1 from RCL;
    $\mathcal{V} = V \setminus \{v_1\}$; G_1 = graph induced on G by $\mathcal{V}$;
    do $k = 2, \ldots, |V| \rightarrow$
        $\underline{d} = \min_{v \in \mathcal{V}}\{\deg_{G_{k-1}}(v)\}$;
        if $\mathrm{ADJ}_{G_{k-1}}(v_{k-1}) \neq \emptyset \ \rightarrow$
            RCL $= \{v \in \mathrm{ADJ}_{G_{k-1}}(v_{k-1}) \ : \ \deg_{G_{k-1}}(v) = \underline{d}\}$
        fi;
        if $\mathrm{ADJ}_{G_{k-1}}(v_{k-1}) = \emptyset \ \rightarrow$
            RCL $= \{v \in \mathcal{V} \ : \ \deg_{G_{k-1}}(v) = \underline{d}\}$
        fi;
        Select v_k from RCL;
        $\mathcal{V} = \mathcal{V} \setminus \{v_k\}$; G_k = graph induced on G by $\mathcal{V}$;
    od
    return $\Pi = (v_1, v_2, \ldots, v_{|V|})$
end `FirstPhaseGT`;

FIG. 2.1. *Pseudo-code of the first phase of (GT greedy) heuristic*

least 3/4 of the number of edges of a maximum planar subgraph of G. Accordingly, the authors attempt to use as the sequence Π linear permutations of the vertices associated with Hamiltonian cycles of G. Two strategies are used in the first phase: (i) a randomized algorithm [1] that almost certainly finds a Hamiltonian cycle if one exists, and (ii) a greedy deterministic algorithm that seeks a Hamiltonian cycle. In the latter, the first vertex in the sequence Π is a minimum degree vertex in G. After the first k vertices of the sequence have been determined, say $v_1, v_2, \cdots, v_k$, the next vertex v_{k+1} is selected from the vertices adjacent to v_k in G having the least adjacencies in the subgraph G_k of G induced by $V \setminus \{v_1, v_2, \cdots, v_k\}$. If there is no vertex of G_k adjacent to v_k in G, then v_{k+1} is selected as a minimum degree vertex in G_k. Pseudo-code for the first-phase heuristic is described in Figure 2.1, where $\deg_G(v)$ is the degree of vertex v with respect to G, $\underline{d} = \min_{v \in V}\{\deg_G(v)\}$, and $\mathrm{ADJ}_{G_{k-1}}(v_{k-1})$ is the set of vertices in G_{k-1} adjacent to v_{k-1} in G.

Let $H = (E, I)$ be a graph where each of its vertices corresponds to an edge of the input graph G. Vertices e_1 and e_2 of H are connected by an edge if the corresponding edges of G cross with respect to sequence Π. A graph is called an *overlap graph* if its vertices can be placed in one-to-one correspondence with a family of intervals on a line. Two intervals are said to overlap if they cross and none is contained in the other. Two vertices of the overlap graph are connected by an edge if and only if their corresponding intervals overlap. Hence, the graph H as constructed above is the overlap graph associated with the representation of G defined by sequence Π.

The second phase of GT consists in two-coloring a maximum number of the vertices of the overlap graph H such that each of the two color classes $\mathcal{B}$ (blue) and $\mathcal{R}$ (red) forms an independent set. Equivalently, the second phase seeks a maximum induced bipartite subgraph of the overlap graph H, i.e. a bipartite subgraph having the largest number of vertices. This problem is equivalent to drawing the edges of the input graph G above or below the line where its vertices have been placed, according to sequence Π. Since the decision version of the problem of finding a maximum induced

procedure `SecondPhaseGT`(V,E,Π,$\mathcal{B}$,$\mathcal{R}$)
    `BuildOverlapGraph`(V, E, Π, I);
    `FindMaxIndependentSet`($E, I, \mathcal{B}$);
    `ReduceOverlapGraph`($E, I, \mathcal{B}, I'$);
    `FindMaxIndependentSet`($E \setminus \mathcal{B}, I', \mathcal{R}$);
    return $\mathcal{B}, \mathcal{R}$
end `SecondPhaseGT`;

FIG. 2.2. *Pseudo-code of the second phase of GT heuristic*

bipartite subgraph of an overlap graph is NP-complete [24], a greedy algorithm is used in GT to construct a maximal induced bipartite subgraph of the overlap graph. This algorithm finds a maximum independent set $\mathcal{B} \subseteq E$ of the overlap graph $H = (E, I)$, reduces the overlap graph by removing from it the vertices in $\mathcal{B}$ and all edges incident to vertices in $\mathcal{B}$, and then finds a maximum independent set $\mathcal{R} \subseteq E \setminus \mathcal{B}$ in the remaining overlap graph $H' = (E \setminus \mathcal{B}, I')$. The two independent sets obtained induce a bipartite subgraph of the original overlap graph, not necessarily with a maximum number of vertices. This procedure has polynomial time complexity, since finding the maximum independent set of an overlap graph has been shown by Gavril [10] to be polynomially solvable in time $O(|E|^3)$, where $|E|$ is the number of vertices of the overlap graph $H = (E, I)$ (see also Golumbic [12] for another description of this algorithm). Pseudo-code for the second phase heuristic of GT is given in Figure 2.2.

As discussed above, this two-phase algorithm is not guaranteed to produce an optimal planar subgraph. Furthermore, under a simple neighborhood definition, it does not necessarily produce even a locally optimal solution. In Section 4, we describe a search procedure that finds a locally optimal planar subgraph, often improving on the solution found by GT. The first phase of GT is based on a greedy algorithm to produce a vertex sequence. As noted by Goldschmidt and Takvorian, the vertex sequence appears to affect the size of the planar subgraph found in the second phase of GT. Moreover, it is not clear that the sequence produced by the greedy algorithm is the best. To produce other, possibly better, sequences, randomization and local search can be introduced in the greedy algorithm. In the following section, we explore these ideas introducing a greedy randomized adaptive search procedure (GRASP) for graph planarization.

3. GRASP. In this paper, we apply the concepts of GRASP to the graph planarization problem. A GRASP [8] is an iterative process, where each GRASP iteration consists of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is explored by local search. The best solution over all GRASP iterations is returned as the result.

In the construction phase, a feasible solution is built, one element at a time. At each construction iteration, the next element to be added is determined by ordering all elements in a candidate list with respect to a greedy function that measures the (myopic) benefit of selecting each element. The adaptive component of the heuristic arises from the fact that the benefits associated with every element are updated at each iteration of the construction phase to reflect the changes brought on by the selection of the previous elements. The probabilistic component of a GRASP is characterized by randomly choosing one of the best candidates in the list, but usually not the top candidate. This way of making the choice allows for different solutions to be obtained

```
procedure GRASP(ListSize,MaxIter,RandomSeed)
1    InputInstance();
2    do k = 1,..., MaxIter →
3        ConstructGreedyRandomizedSolution(ListSize,RandomSeed);
4        LocalSearch(BestSolutionFound);
5        UpdateSolution(BestSolutionFound);
6    od;
7    return BestSolutionFound
end GRASP;
```

FIG. 3.1. *A generic GRASP pseudo-code*

at each GRASP iteration, but does not necessarily jeopardize the power of GRASP's adaptive greedy component.

The solutions generated by a GRASP construction are not guaranteed to be locally optimal with respect to simple neighborhood definitions. Hence, it is almost always beneficial to apply a local search to attempt to improve each constructed solution. A local search algorithm works in an iterative fashion by successively replacing the current solution by a better solution from its neighborhood. It terminates when there is no better solution found in the neighborhood with respect to some cost function. Success for a local search algorithm depends on the suitable choice of a neighborhood structure, efficient neighborhood search techniques, and the starting solution. The GRASP construction phase plays an important role with respect to this last point, since it produces good starting solutions for local search. Normally, a local optimization procedure, such as a two-exchange, is employed. While such procedures can require exponential time from an arbitrary starting point, empirically their efficiency significantly improves as the initial solutions improve [23]. Through the use of customized data structures and careful implementation, an efficient construction phase that produces good initial solutions for efficient local search can be created. The result is that often many GRASP solutions are generated in the same amount of time required for the local optimization procedure to converge from a single random start. Furthermore, the best of these GRASP solutions is generally significantly better than the solution obtained from a random starting point.

Figure 3.1 illustrates a generic GRASP implementation in pseudo-code. The GRASP takes as input parameters for setting the candidate list size, maximum number of GRASP iterations and the seed for the random number generator. After reading the instance data (line 1), the GRASP iterations are carried out in lines 2–6. Each GRASP iteration consists of the construction phase (line 3), the local search phase (line 4) and, if necessary, the incumbent solution update (line 5).

As outlined above, a GRASP possesses four basic components: a greedy function, an adaptive search strategy, a probabilistic selection procedure, and a local search technique. These components are linked together into an iterative method that constructs a feasible solution one element at a time and then feeds the solution to the local search procedure.

As discussed earlier, one variant of the two-phase GT heuristic uses a greedy algorithm to produce the vertex sequencing of its first phase. We show, in the following, an alternative to the greedy algorithm: a GRASP for the first phase vertex sequencing problem. The construction phase of this GRASP is described in the pseudo-code of Figure 3.2. The procedure takes as input the graph $G = (V, E)$ to be planarized, the

procedure `ConstructGreedyRandomizedSolution`(α,`seed`,V,E,Π)
1 $\underline{d} = \min_{v \in V}\{\deg_G(v)\}$; $\bar{d} = \max_{v \in V}\{\deg_G(v)\}$;
2 $\text{RCL} = \{v \in V \ : \ \underline{d} \leq \deg_G(v) \leq \alpha(\bar{d} - \underline{d}) + \underline{d}\}$;
3 $v_1 = \texttt{random}(\texttt{seed}, \text{RCL})$;
4 $\mathcal{V} = V \setminus \{v_1\}$; $G_1 =$ graph induced on G by $\mathcal{V}$;
5 **do** $k = 2, \ldots, |V| \rightarrow$
6 $\underline{d} = \min_{v \in \mathcal{V}}\{\deg_{G_{k-1}}(v)\}$; $\bar{d} = \max_{v \in \mathcal{V}}\{\deg_{G_{k-1}}(v)\}$;
7 **if** $\text{ADJ}_{G_{k-1}}(v_{k-1}) \neq \emptyset \rightarrow$
8 $\text{RCL} = \{v \in \text{ADJ}_{G_{k-1}}(v_{k-1}) \ : \ \underline{d} \leq \deg_{G_{k-1}}(v) \leq \alpha(\bar{d} - \underline{d}) + \underline{d}\}$
9 **fi**;
10 **if** $\text{ADJ}_{G_{k-1}}(v_{k-1}) = \emptyset \rightarrow$
11 $\text{RCL} = \{v \in \mathcal{V} \ : \ \underline{d} \leq \deg_{G_{k-1}}(v) \leq \alpha(\bar{d} - \underline{d}) + \underline{d}\}$
12 **fi**;
13 $v_k = \texttt{random}(\texttt{seed}, \text{RCL})$;
14 $\mathcal{V} = \mathcal{V} \setminus \{v_k\}$; $G_k =$ graph induced on G by $\mathcal{V}$;
15 **od**
16 **return** $\Pi = (v_1, v_2, \ldots, v_{|V|})$
end `ConstructGreedyRandomizedSolution`;

FIG. 3.2. *Pseudo-code of GRASP construction phase*

restricted candidate list (RCL) parameter α ($0 \leq \alpha \leq 1$), and a seed for the pseudo random number generator. As before, let $\deg_G(v)$ be the degree of vertex v with respect to G, and let $\underline{d} = \min_{v \in V}\{\deg_G(v)\}$ and $\bar{d} = \max_{v \in V}\{\deg_G(v)\}$. The first vertex in the sequence is determined in lines 1–3, where all vertices having degree in the range $[\underline{d}, \alpha(\bar{d} - \underline{d}) + \underline{d}]$ are placed in the RCL and a single vertex is selected at random from the list. The working vertex set $\mathcal{V}$ and graph G_1 are defined in line 4.

The loop from line 5 to 15 determines the sequence of the remaining $|V| - 1$ vertices. To assign the k-th vertex (iteration k of the loop) two cases can occur. Define G_k to be the graph induced on G by $V \setminus \{v_1, v_2, \cdots, v_k\}$. Let $\text{ADJ}_{G_{k-1}}(v_{k-1})$ be the set of vertices of G_{k-1} adjacent to v_{k-1} in G. The RCL is made up of all vertices in $\text{ADJ}_{G_{k-1}}(v_{k-1})$ having degree in the range $[\underline{d}, \alpha(\bar{d} - \underline{d}) + \underline{d}]$ in G_k. Otherwise, if $\text{ADJ}_{G_{k-1}}(v_{k-1}) = \emptyset$, the RCL is made up of all unselected vertices having degree in the range $[\underline{d}, \alpha(\bar{d} - \underline{d}) + \underline{d}]$ in G_k. In line 13, the k-th vertex in the sequence is determined by selecting a vertex, at random, from the RCL. The greedy function is adapted in line 14 where the working vertex set and graph is updated. The vertex sequence $\Pi = (v_1, v_2, \ldots, v_{|V|})$ is returned.

The first phase of the GT heuristic seeks a sequence of the vertices, followed by a second phase minimizing the number of edges that need to be removed to eliminate all edge crossings with respect to the first-phase sequence. One possible strategy (not taken in GT) is to attempt to reduce the number of crossing edges by locally searching a neighborhood of the current vertex sequence prior to the second phase. We next describe such a local search procedure. To do so, let us define the neighborhood $\mathcal{N}(\Pi)$ of the vertex sequence Π to be formed by all vertex sequences Π' differing in exactly two positions, i.e.

$$\mathcal{N}(\Pi) = \{\Pi' = (v'_1, v'_2, \ldots, v'_{|V|}) \ : \ v'_i = v_i, \forall i \neq j, k, \ v'_j = v_k, \ v'_k = v_j\}.$$

Let $\chi(\Pi)$ be the number of edge-pairs that cross with respect to vertex sequence Π. The pseudo-code in Figure 3.3 describes the local search procedure used in the

procedure `LocalSearch`(V,E,Π)
1 **do** Π is not locally optimal $\rightarrow$
2 Find $\Pi' \in \mathcal{N}(\Pi)$ such that $\chi(\Pi') < \chi(\Pi)$;
3 $\Pi = \Pi'$;
4 **od**
5 **return** $\Pi = (v_1, v_2, \dots, v_{|V|})$
end `LocalSearch`;

FIG. 3.3. *Pseudo-code of GRASP local search phase*

procedure `GRASPforGP`(α,`seed`,`MaxIter`,V,E,Π^*,$\mathcal{B}^*$,$\mathcal{R}^*$)
1 **do** $k = 1, 2, \dots,$ `MaxIter` $\rightarrow$
2 `ConstructGreedyRandomizedSolution`(α,`seed`,V,E,Π);
3 `LocalSearch`(V,E,Π);
4 `SecondPhaseGT`(V,E,Π,$\mathcal{B}$,$\mathcal{R}$);
5 `UpdateSolution`(Π,$\mathcal{B}$,$\mathcal{R}$,Π^*,$\mathcal{B}^*$,$\mathcal{R}^*$);
6 **od**
7 **return** $\Pi^*, \mathcal{B}^*, \mathcal{R}^*$
end `GRASPforGP`;

FIG. 3.4. *Pseudo-code of GRASP for graph planarization*

GRASP. In our implementation, we use a slightly more restricted neighborhood, in which only consecutive vertices can be exchanged.

By incorporating the second phase of the GT heuristic to the above GRASP for vertex sequencing, we have a GRASP for graph planarization, whose pseudo-code is given in Figure 3.4. The procedure is repeated `MaxIter` times. In each iteration, a greedy randomized solution (vertex sequence Π) is constructed in line 2. In line 3, the local search attempts to produce a vertex sequence that has fewer edge-pair crossings than the one generated in line 2. The vertex sequence Π is given as input to the second phase heuristic of GT in line 4 to produce a planar subgraph of G. If the cardinality of the edge-set of the subgraph found in line 4 is the largest found so far, it is recorded in line 5.

The computational complexity of phases 1 and 2 of this GRASP is discussed next. The construction phase takes $|V| - 1$ iterations, each of which has complexity $O(|V|)$, resulting in an $O(|V|^2)$ procedure. To analyze the local search phase, we need an upper bound on the number of edge crossings $\chi(\Pi)$ for a given vertex sequence Π. The number of edge crossings in a complete graph is bounded above by $O(|V|^4)$. Each iteration of the do loop 1–4 reduces the value of $\chi(\Pi)$ by at least one. Consequently, the number of iterations is bounded above by $O(|V|^4)$. Since each iteration has time complexity $O(|E|\ |V|^2)$, the time complexity for the local search is $O(|E|\ |V|^6)$. However, in the experiments carried out in this study, phase 2 always converged in a small number of iterations, as illustrated in Figure 5.5 in Section 5. This is because a reduction of more than one unit in the value of $\chi(\Pi)$ is often achieved in a single iteration of the procedure, the number of initial crossings is usually small, and the minimum number of crossings is usually larger than zero.

4. Enlarging the Planar Subgraph. As pointed out in Section 2, there is no guarantee that the planar subgraph produced by `SecondPhaseGT` is optimal. Three edge sets are output: $\mathcal{B}$ (blue edges), $\mathcal{R}$ (red edges), and $\mathcal{P}$ (the remaining edges,

```
procedure EnlargePlanar(B, R, P, V)
1   do p ∈ P →
2       B̂_p = ∅;
3       do b ∈ B →
4           if p and b cross →
5               B̂_p = B̂_p ∪ {b};
6               do r ∈ R →
7                   if r and b cross → goto 12 fi;
8               od;
9           fi;
10      od;
11      (B, R, P) = (B ∪ {p} \ B̂_p, R ∪ B̂_p, P \ {p});
12  od;
13  return B, R
end EnlargePlanar;
```

FIG. 4.1. *Pseudo-code of local search procedure to enlarge planar subgraph*

which we refer to as the *pale* edges). By construction, $\mathcal{B}$, $\mathcal{R}$, and $\mathcal{P}$ are such that no red or pale edge can be colored blue. Likewise, pale edges cannot be colored red. However, if there exists a pale edge $p \in \mathcal{P}$ such that all blue edges that cross with p (let $\hat{\mathcal{B}}_p \subseteq \mathcal{B}$ be the set of those blue edges) do not cross with any red edge $r \in \mathcal{R}$, then all blue edges $b \in \hat{\mathcal{B}}_p$ can be colored red and p can be colored blue. This reassignment of color classes increases the size of the planar subgraph by one edge.

Figure 4.1 shows pseudo-code for procedure `EnlargePlanar` that seeks pale and blue edges allowing the above color class reassignment and enlarges the planar subgraph when such edges are encountered. The pale edges are scanned in **do** loop 1–12, while loop 3–10 scans the blue edges to construct the set $\hat{\mathcal{B}}_p$, whose elements are saved in line 5. If a blue edge $b \in \hat{\mathcal{B}}_p$ crosses any red edge, the set $\hat{\mathcal{B}}_p$ is discarded (line 7) and a new pale edge is scanned. If no edge $b \in \hat{\mathcal{B}}_p$ crosses red edges, then all blue edges $b \in \hat{\mathcal{B}}_p$ are colored red and the pale edge p is colored blue in line 11. This procedure has time complexity $O(|E|^3)$.

Procedure `EnlargePlanar` can be incorporated into the GRASP described in Section 3. Figure 4.2 shows pseudo-code for a complete GRASP that incorporates the local search procedure to enlarge planar subgraphs.

5. Experimental Results. In this section, we present experimental results with a Fortran implementation of the GRASP for graph planarization `GRASPforGP` shown in Figure 4.2. A set of 75 test problems [1] described in the literature [6, 11] was used in the experiments. Tables 5.1 to 5.3 summarize the test problem statistics. For each instance the tables list its name, number of vertices, number of edges, the Euler upper bound $(3|V| - 6)$ on the number of edges in a maximum planar subgraph, the number of edges in the best known planar subgraph published prior to this paper, and indications as to whether the best known solution is optimal, and as to whether the input graph is Hamiltonian. Solutions indicated as optimal are those that have been solved by exact methods or for which a solution equal to an upper bound has been found. Graphs for which Hamiltonian cycles were produced using the randomized

[1] input data: `ftp://netlib.att.com/math/people/mgcr/data/planar-data.tar.gz`

```
procedure GRASPforGP(α,seed,MaxIter,V,E,Π*,ℬ*,ℛ*)
    do k = 1, 2, ..., MaxIter →
        ConstructGreedyRandomizedSolution(α,seed,V,E,Π);
        LocalSearch(V,E,Π);
        SecondPhaseGT(V,E,Π,ℬ,ℛ);
        𝒫 = E \ (ℬ ∪ ℛ);
        EnlargePlanar(ℬ,ℛ,𝒫,V);
        UpdateSolution(Π,ℬ,ℛ,Π*,ℬ*,ℛ*);
    od
    return Π*, ℬ*, ℛ*
end GRASPforGP;
```

FIG. 4.2. *Pseudo-code of GRASP for graph planarization*

TABLE 5.1
Goldschmidt and Takvorian test problem statistics

	graph dimension					
name	$\|V\|$	$\|E\|$	$3\|V\|-6$	sol'n†	optimal?	Hamiltonian?
g1	10	22	24	20	yes	yes
g2	45	85	82	82	yes	yes
g3	10	24	24	24	yes	yes
g4	10	25	24	24	yes	yes
g5	10	26	24	24	yes	yes
g6	10	27	24	24	yes	yes
g7	10	34	24	24	yes	yes
g8	25	69	69	69	yes	yes
g9	25	70	69	69	yes	yes
g10	25	71	69	69	yes	yes
g11	25	72	69	69	yes	yes
g12	25	90	69	69	yes	yes
g13	50	367	144	131	no	yes
g14	50	491	144	138	no	yes
g15	50	582	144	142	no	yes
g16	100	451	294	183	no	yes
g17	100	742	294	219	no	yes
g18	100	922	294	237	no	yes
g19	150	1064	444	297	no	yes

† best known previously published solution

algorithm of Angluin and Valiant [1] are indicated with a "yes," while those for which the algorithm failed have the "no?" indication. We repeated the Angluin and Valiant algorithm at most 100 times (each using a different pseudo-random number generator seed), with an inner loop for each repetition in which each of the $|V|$ vertices is used as the starting node for the algorithm.

Table 5.1 summarizes problem statistics for instances `g1` to `g19`. These are Hamiltonian graphs proposed by Goldschmidt and Takvorian [11] to test their two-phase heuristic. We were unable to obtain the instances `g20` and `g21` that were also included in [11]. Table 5.2 summarizes seven classes of test problems proposed in [6]. Graphs `cimi-g1` to `cimi-g6` are special graphs collected by Cimikowski, either presented in other papers or relevant to applications. Instances `rg50.1` to `rg300.5`, generated by Cimikowski, are random non-planar graphs with unknown optimal solution. Table 5.3 summarizes another class of graphs introduced in [6], in which instances `tg100.1` to `tg200.10` are non-planar graphs having a known maximum planar subgraph of size

TABLE 5.2
Cimikowski test problem statistics – part 1

name	graph dimension $\|V\|$	$\|E\|$	$3\|V\|-6$	sol'n†	optimal?	Hamiltonian?
`cimi-g1`	10	21	24	19	yes	yes
`cimi-g2`	60	166	174	165	yes	yes
`cimi-g3`	28	75	78	73	yes	yes
`cimi-g4`	10	22	24	20	yes	yes
`cimi-g5`	45	85	129	82	yes	yes
`cimi-g6`	43	63	123	59	yes	no?
`rg50.1`	50	123	144	91	?	no?
`rg50.2`	50	145	144	94	no	yes
`rg50.3`	50	157	144	100	no	no?
`rg50.4`	50	171	144	102	no	yes
`rg50.5`	50	183	144	105	?	yes
`rg75.1`	75	196	219	129	no	no?
`rg75.2`	75	202	219	137	?	no?
`rg75.3`	75	215	219	135	?	no?
`rg75.4`	75	256	219	136	no	no?
`rg75.5`	75	266	219	140	no	no?
`rg100.1`	100	261	294	161	no	no?
`rg100.2`	100	271	294	163	no	no?
`rg100.3`	100	297	294	164	no	no?
`rg100.4`	100	334	294	171	no	no?
`rg100.5`	100	373	294	186	no	no?
`rg150.1`	150	387	444	231	?	no?
`rg150.2`	150	402	444	227	?	no?
`rg150.3`	150	453	444	229	no	no?
`rg150.4`	150	473	444	234	no	no?
`rg150.5`	150	481	444	241	no	no?
`rg200.1`	200	514	594	284	?	no?
`rg200.2`	200	519	594	283	no	no?
`rg200.3`	200	644	594	295	no	no?
`rg200.4`	200	684	594	297	no	no?
`rg200.5`	200	701	594	300	no	no?
`rg300.1`	300	814	594	398	no	no?
`rg300.2`	300	1159	594	420	no	no?
`rg300.3`	300	1176	594	428	no	no?
`rg300.4`	300	1474	594	469	no	no?
`rg300.5`	300	1507	594	472	no	no?

† best known previously published solution

$3|V|-6$.

Procedure `GRASPforGP` is compared to results published in [6, 11, 21] as well as our implementation (procedure `2PhaseImpl`, illustrated in Figure 5.1) of the (UT greedy) two-phase heuristic [11]. Our Fortran implementation of `2PhaseImpl` uses the code for our GRASP construction phase with parameter α set to 0, forcing a greedy construction as described in [11]. Ties are broken at random. No local search is used. The portable pseudo random number generator of Schrage [25] was used, with the initial seed 270001 to generate the pseudo random number stream used to select the candidates from the restricted candidate list.

The experiment was conducted on a Silicon Graphics Challenge computer (250 MHz MIPS M4400 processor), whose hardware configuration is summarized in Figure 5.2. The code was compiled on the SGI Fortran compiler `f77` using compiler flags `-O2 -Olimit 800 -static`. Processes were limited to a single processor. CPU times in seconds were computed by calling the system routine `etime()`. Reported CPU

TABLE 5.3
Cimikowski test problem statistics – part 2

	graph dimension					
name	$\|V\|$	$\|E\|$	$3\|V\|-6$	sol'n†	optimal?	Hamiltonian?
`tg100.1`	100	304	294	294	yes	no?
`tg100.2`	100	314	294	294	yes	no?
`tg100.3`	100	324	294	294	yes	no?
`tg100.4`	100	334	294	294	yes	no?
`tg100.5`	100	344	294	292	no	no?
`tg100.6`	100	354	294	287	no	no?
`tg100.7`	100	364	294	279	no	no?
`tg100.8`	100	374	294	281	no	no?
`tg100.9`	100	384	294	279	no	no?
`tg100.10`	100	394	294	277	no	no?
`tg200.1`	200	604	594	594	yes	no?
`tg200.2`	200	614	594	594	yes	no?
`tg200.3`	200	624	594	594	yes	no?
`tg200.4`	200	634	594	591	no	no?
`tg200.5`	200	644	594	567	no	no?
`tg200.6`	200	654	594	572	no	no?
`tg200.7`	200	664	594	543	no	no?
`tg200.8`	200	674	594	550	no	no?
`tg200.9`	200	684	594	537	no	no?
`tg200.10`	200	694	594	536	no	no?

† best known previously published solution

procedure `2PhaseImpl`(`seed`,`MaxIter`,V,E,Π^*,$\mathcal{B}^*$,$\mathcal{R}^*$)
    do $k = 1, 2, \ldots,$ `MaxIter` $\rightarrow$
        `ConstructGreedyRandomizedSolution`($\alpha = 0$,`seed`,V,E,Π);
        `SecondPhaseGT`(V,E,Π,$\mathcal{B}$,$\mathcal{R}$);
        `UpdateSolution`(Π,$\mathcal{B}$,$\mathcal{R}$,Π^*,$\mathcal{B}^*$,$\mathcal{R}^*$);
    od
    return $\Pi^*, \mathcal{B}^*, \mathcal{R}^*$
end `2PhaseImpl`;

FIG. 5.1. *Pseudo-code of our implementation of the two-phase heuristic for graph planarization*

times exclude problem input time, which is negligible for these test problems.

The performance of most heuristics depends on parameter setting. GRASP requires few parameters to be set. To facilitate reproducibility, as well as to investigate the robustness of the approach, we limit our runs in this experiment to two sets of parameter settings. We use `MaxIter` $= 10,000$ and two settings for the RCL parameter: $\alpha = 0.1$ and $\alpha = 0.5$. In the tables, we report on the GRASP runs corresponding to the α settings that produced the largest planar subgraph. In general, $\alpha = 0.1$ produced the best results. However, on several of the smaller instances, $\alpha = 0.5$ did better.

Statistics of the runs [2] are displayed in Tables 5.4 to 5.6. For GRASP and our implementation `2PhaseImpl` of the greedy variant of the two-phase heuristic, each table lists instance name, iteration on which the best solution was found, running time until the best solution was found, and the number of edges in the largest planar subgraph produced. In addition, for GRASP the tables also indicate the RCL parameter

[2] GRASP solution: `ftp://netlib.att.com/math/people/mgcr/sol/planar-soln.tar.gz`

```
12 250 MHZ IP19 Processors
CPU: MIPS R4400 Processor Chip Revision:  6.0
FPU: MIPS R4010 Floating Point Chip Revision:  0.0
Data cache size:  16 Kbytes
Instruction cache size:  16 Kbytes
Secondary unified instruction/data cache size:  4 Mbyte
Main memory size:  2560 Mbytes, 2-way interleaved
```

FIG. 5.2. *Hardware configuration (partial output of system command* `hinv`*)*

TABLE 5.4
Run statistics on Goldschmidt and Takvorian instances

	GRASP				two-phase heuristic				J-M‡	
name	α	itr	time	sol'n	itr	time	sol'n	sol'n†	time	sol'n
`g1`	0.5	1	0.0	20	1	0.0	20	20	0.3	20
`g2`	0.5	37	1.0	82	37	0.6	82	82	10.2	82
`g3`	0.5	1	0.0	24	14	0.0	22	24	0.1	24
`g4`	0.5	16	0.0	24	4	0.0	22	24	0.0	24
`g5`	0.5	43	0.0	24	1	0.0	22	24	0.5	24
`g6`	0.5	37	0.1	24	14	0.0	23	24	0.1	24
`g7`	0.5	1	0.0	24	10	0.0	24	24	0.2	24
`g8`	0.5	1306	10.3	69	3	0.0	60	68	0.1	69
`g9`	0.5	1306	10.5	69	7	0.0	60	69	0.1	69
`g10`	0.5	2129	18.5	69	6	0.0	60	68	0.6	69
`g11`	0.5	134	1.2	69	13	0.1	61	68	0.6	69
`g12`	0.5	7	0.1	67	871	13.3	65	67	1000.0	68
`g13`	0.1	569	1146.7	135	411	508.2	134	131	1000.0	125
`g14`	0.1	29	136.7	143	29	84.2	143	138	1000.0	133
`g15`	0.1	27	225.4	144	6	51.2	144	142	1000.0	138
`g16`	0.1	8468	28848.6	196	1187	2481.9	192	183	1000.0	187
`g17`	0.1	6286	10805.7	236	4796	47725.2	234	219	1000.0	213
`g18`	0.1	4432	83508.1	246	12	226.6	246	237	1000.0	223
`g19`	0.1	3428	97939.4	311	7832	224113.8	313	297	1000.0	290

† best solution published in [11]
‡ best solution found by Jünger and Mutzel [18]

used for the reported run. The average running time per iteration multiplied by the total number of iterations gives an accurate estimate of the total running time of the heuristic. See also Figure 5.5 which shows running time for 1000 GRASP iterations as a function of the number of edges of the input graph. Tables 5.4 to 5.6 also list the sizes of the largest planar subgraphs using both the GT heuristic [11] and the branch-and-cut approach of Jünger and Mutzel [16, 18], as reported in [18, 21].

Table 5.4 lists results on the Hamiltonian graphs `g1` to `g19`. The GRASP code found provably optimal solutions for problems `g1` to `g11`, and `g15`. The results reported in [11] indicate that the two-phase heuristic found provably optimal solutions for these problems with the exception of `g8`, `g10`, `g11`, and `g15`. On the 11 instances that GT did not produce a provably optimal solution (`g8`, `g10` to `g19`), the GRASP code found better solutions, of which four are provably optimal. Run statistics for problems `cimi-g1` to `cimi-g6` are given in Table 5.5. On four of the six instances, GRASP improved upon the best known solution found by GT in [6], matching GT's solution on the remaining two problems. Table 5.5 also summarizes results for test problems `rg50.1` to `rg300.5`. On all 30 instances, the GRASP improved upon the best known solution found by GT in [6]. Runs on test problems `tg100.1` to `tg200.10`

TABLE 5.5
Run statistics on Cimikowski instances – part 1

	GRASP				two-phase heuristic				J–M
name	α	itr	time	sol'n	itr	time	sol'n	sol'n†	sol'n‡
cimi-g1	0.5	1	0.0	19	1	0.0	19	19	19
cimi-g2	0.5	31	2.9	165	31	1.8	165	149	165
cimi-g3	0.5	388	2.5	73	152	0.8	63	73	73
cimi-g4	0.5	1	0.0	20	2	0.0	20	19	20
cimi-g5	0.5	37	1.0	82	37	0.6	82	80	82
cimi-g6	0.1	1672	20.6	59	456	3.6	58	54	59
rg50.1	0.1	377	28.1	90	4720	206.2	90	83	91
rg50.2	0.1	1956	223.2	95	8247	575.7	95	86	94
rg50.3	0.1	1797	914.9	103	1797	172.2	103	96	100
rg50.4	0.5	1977	257.5	102	4001	538.9	101	92	102
rg50.5	0.1	243	57.7	105	1400	203.2	105	96	105
rg75.1	0.1	2215	579.3	130	2215	358.1	128	115	129
rg75.2	0.1	965	331.3	131	965	158.3	131	124	137
rg75.3	0.1	4881	1718.5	134	4037	907.4	133	128	135
rg75.4	0.1	4453	2660.4	143	4453	1572.5	143	134	136
rg75.5	0.1	1107	768.9	143	1032	432.2	138	129	140
rg100.1	0.1	9583	5795.2	162	856	317.4	161	153	161
rg100.2	0.1	809	565.0	164	1578	888.3	164	157	163
rg100.3	0.1	4149	3912.6	166	485	287.9	167	156	164
rg100.4	0.1	7893	11098.4	175	9794	8286.8	175	163	171
rg100.5	0.1	118	222.3	187	118	138.5	185	171	186
rg150.1	0.1	4365	8593.7	228	1074	1281.5	228	210	231
rg150.2	0.1	3652	8098.9	225	4251	5813.9	225	213	227
rg150.3	0.1	882	2829.6	237	2547	5028.9	237	222	229
rg150.4	0.1	4314	15999.0	238	5258	11780.4	238	223	234
rg150.5	0.1	2877	11821.5	244	5642	13395.3	244	227	241
rg200.1	0.1	1122	3059.7	284	4057	11297.5	283	268	284
rg200.2	0.1	5906	16776.1	288	2324	6685.4	285	278	283
rg200.3	0.1	5544	30753.1	307	1809	10090.7	306	286	295
rg200.4	0.1	4182	28377.8	311	2298	15622.3	310	296	297
rg200.5	0.1	4606	32395.3	315	819	5622.3	313	296	300
rg300.1	0.1	9615	179813.1	414	2828	31176.9	415	395	398
rg300.2	0.1	3951	131783.9	453	8506	285278.3	451	431	420
rg300.3	0.1	4720	165583.8	457	4720	165679.1	457	439	428
rg300.4	0.1	3864	279249.1	484	8325	602284.3	483	458	469
rg300.5	0.1	4547	365024.2	487	6364	509469.0	483	467	472

† solution published in [6]
‡ solution published in [21]

are listed in Table 5.6. On all 20 instances, GRASP found solutions that were better than those found by GT in [6].

The implementation of the two-phase heuristic used in [6, 11] applies the algorithm of Angluin and Valiant [1] at most $|V|$ times (starting from a different vertex each time) trying to find a Hamiltonian cycle to induce a vertex sequencing for the first phase. If no cycle is found, the greedy algorithm described earlier is used. That algorithm is deterministic, with ties broken up lexicographically [28]. Our implementation of the greedy variant of the two-phase heuristic (`2PhaseImpl`) uses only the greedy algorithm and is randomized, since ties are broken at random. We compared it with the GRASP on all of the instances, running 10,000 iterations on both codes. Of all 75 instances, GRASP found a better solution in 34 instances, while on 17 (14 of which from the `tg100` and `tg200` problem classes) `2PhaseImpl` produced a better solution.

Our implementation of the greedy variant of the two-phase heuristic can be com-

TABLE 5.6
Run statistics on Cimikowski instances – part 2

	GRASP				two-phase heuristic				J–M
name	α	itr	time	sol'n	itr	time	sol'n	sol'n†	sol'n‡
`tg100.1`	0.1	380	197.0	258	9480	5284.5	264	235	294
`tg100.2`	0.1	236	221.6	265	4699	2612.1	257	223	294
`tg100.3`	0.1	3286	3498.6	264	2897	1883.8	265	224	294
`tg100.4`	0.1	1863	2467.5	262	8868	7154.3	257	228	294
`tg100.5`	0.1	5271	7279.1	259	5946	5001.4	258	234	292
`tg100.6`	0.1	2012	3124.8	253	3794	3535.1	255	242	287
`tg100.7`	0.1	1588	2710.3	246	4864	4955.3	248	239	279
`tg100.8`	0.1	2663	5210.5	252	1967	2906.4	250	221	281
`tg100.9`	0.1	4376	8879.3	249	5630	7405.9	245	235	279
`tg100.10`	0.1	5878	13149.8	247	7188	10499.0	239	224	277
`tg200.1`	0.1	7871	28047.2	504	8764	42741.5	512	461	594
`tg200.2`	0.1	1836	7823.7	488	7186	42496.5	509	439	594
`tg200.3`	0.1	5279	20127.3	516	9835	55318.9	519	451	594
`tg200.4`	0.1	5526	25155.7	494	5561	59987.4	504	464	591
`tg200.5`	0.1	6285	29727.1	491	5596	25158.2	498	439	567
`tg200.6`	0.1	2416	12456.0	488	8809	42130.8	511	411	572
`tg200.7`	0.1	1749	9712.3	466	5264	27785.1	470	408	543
`tg200.8`	0.1	351	2036.7	492	677	3796.1	500	423	550
`tg200.9`	0.1	2927	18362.9	477	3977	24616.6	480	417	537
`tg200.10`	0.1	8331	55375.8	476	4735	32445.6	478	461	536

† solution published in [6]
‡ solution published in [21]

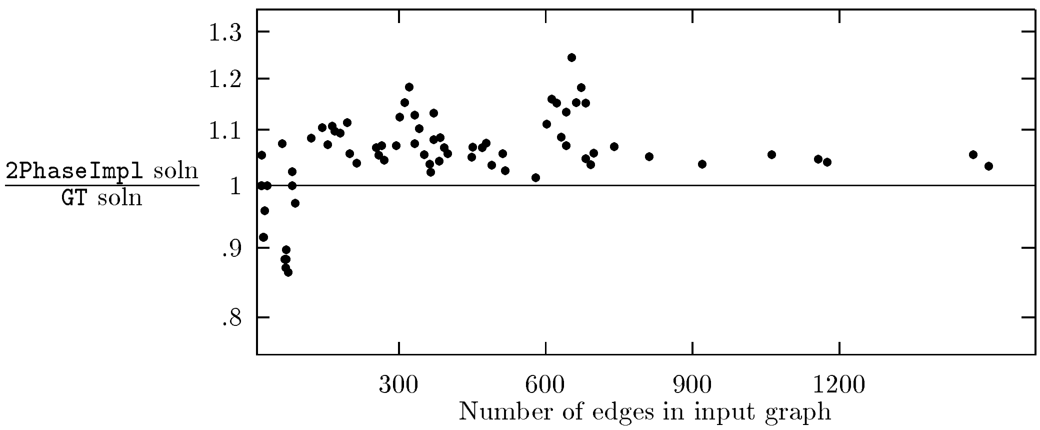

FIG. 5.3. *Ratio (log scale) of number of edges in planar subgraphs found by GT and 2PhaseImpl as a function of the number of edges in the input graph*

pared with results reported in [6, 11]. In [11] two variants of the two-phase heuristic are tested on instances `g1` to `g19`. The first is the greedy variant described in Section 2. Because ties are broken lexicographically, the code is run a single iteration. The second variant generates 20 Hamiltonian cycles and uses each of them to induce a vertex sequence for phase one. Table 5.7 shows the best solutions found by each variant of GT, as well as the solution value found by `2PhaseImpl` after 1, 20, 100, 1000, and 10,000 iterations. After 10,000 iterations `2PhaseImpl` was able to produce solutions at least as good as the greedy variant of GT on all 19 instances. Even with a single iteration `2PhaseImpl` matched or produced a better solution than greedy GT

TABLE 5.7
Comparing 2PhaseImpl with GT.

name	GT solution		2PhaseImpl solution at iteration				
	greedy†	hamilt‡	1	20	100	1000	10000
g1	20	20	20	20	20	20	20
g2	80	82	80	80	82	82	82
g3	21	24	21	22	22	22	22
g4	21	24	21	22	22	22	22
g5	21	24	22	22	22	22	22
g6	21	24	22	23	23	23	23
g7	22	24	23	24	24	24	24
g8	60	68	58	60	60	60	60
g9	60	69	58	60	60	60	60
g10	59	68	59	60	60	60	60
g11	59	68	59	61	61	61	61
g12	62	67	61	63	64	65	65
g13	131	129	126	130	131	134	134
g14	136	138	137	141	143	143	143
g15	142	142	141	144	144	144	144
g16	180	183	180	186	188	190	192
g17	219	215	225	227	230	233	234
g18	237	234	240	246	246	246	246
g19	297	291	299	306	306	310	313

† greedy variant
‡ Hamiltonian cycles variant

on 14 of the 19 instances. Likewise, after 10,000 iterations `2PhaseImpl` produced solutions at least as good as those found by the Hamiltonian cycle variant of GT on 10 of the 19 instances. However, after 20 iterations (the number of cycle-induced vertex orderings used by GT) `2PhaseImpl` was better than the Hamiltonian cycle variant of GT on only 9 of the 19 instances. We should note that on all of the larger instances `g13` to `g19`, the Hamiltonian cycle variant of GT produced smaller planar subgraphs than `2PhaseImpl` did after only 20 iterations. On the remaining instances we compare with the GT runs described in [6]. For those runs, GT attempted to construct a Hamiltonian cycle starting from each vertex [5]. If no cycle is produced, a vertex sequence is determined with the greedy algorithm. On all but one of the instances `cimi-g1` to `cimi-g6` the code `2PhaseImpl` either matched or improved on the value reported for GT in [6]. On all of the instances `rg50.1` to `rg300.5`, `2PhaseImpl` produced larger planar subgraphs than what was reported for GT in [6]. Finally, on test problems `tg100.1` to `tg200.10`, `2PhaseImpl` improved upon all of the GT solutions. Figure 5.3 depicts the ratio of edges in the planar subgraphs found by `2PhaseImpl` to edges in the planar subgraphs found by GT.

Mutzel [18, 21] ran a heuristic variant (JM) of the exact branch-and-cut algorithm of Jünger and Mutzel [16], on all of the test problems. A time limit of 1000 seconds (on a SUN SPARCstation 10/41) was imposed on the runs and the best feasible integer solution found was returned as the heuristic solution. We were unable to obtain the branch-and-cut code from the authors to reproduce the experiment on our computer and therefore rely on the results published by Mutzel. A comparison of GRASP with JM depends heavily on the instances. On 49 of the 55 instances `g1` to `g19`, `cimi-g1` to `cimi-g6` and `rg50.1` to `rg300.5`, the GRASP either matched or produced better solutions than JM. On 30 of those 55 instances, the GRASP solution was strictly better than JM. However, on the instances `tg100.1` to `tg200.10`, JM performs remarkably better than on the other instances, and outperforms all other

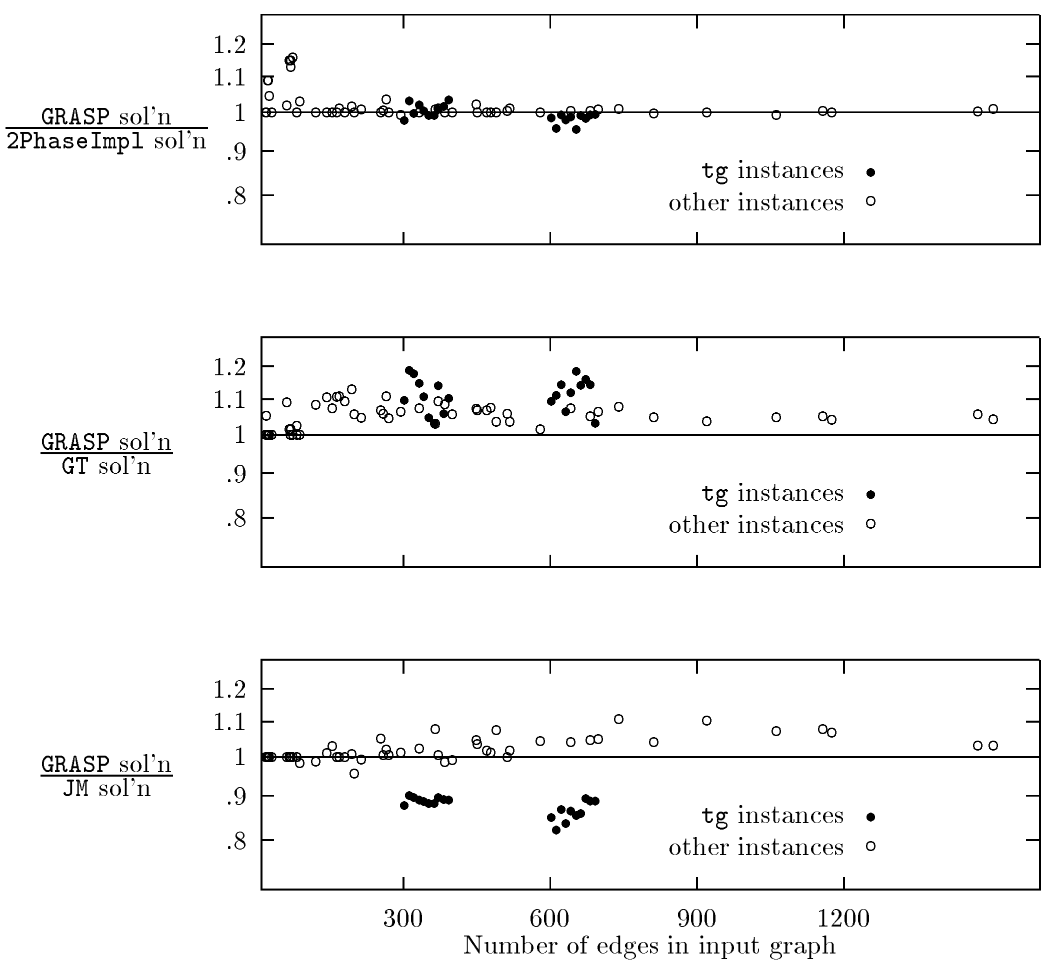

FIG. 5.4. *Ratios (log scale) of number of edges in planar subgraphs found by GRASP and 2PhaseImpl, GT, and JM as a function of the number of edges in the input graph*

algorithms on all 20 instances.

Figure 5.4 compares solutions found by GRASP with those found by `2PhaseImpl`, GT, and JM, showing ratios of number of edges in the planar subgraphs found by each algorithm. The figure shows that GRASP consistently found better solutions except for test problem class `tg`, for which the branch-and-cut code consistently found the largest planar subgraphs.

The GRASP construction phase `ConstructGreedyRandomizedSolution` is not guaranteed to produce locally optimal solutions, even for simple neighborhood structures. This is observed in `GRASPforGP`, where procedure `LocalSearch` frequently reduces the number of edge crossings induced by the vertex sequence. Perhaps of greater interest, is the effect of `LocalSearch` on the size of the planar subgraphs output by `GRASPforGP`. It turned out that on 16 instances (out of 75) the planar graphs produced by GRASP construction followed by `LocalSearch` were larger or a solution of the same size was obtained in fewer iterations, with respect to GRASP construction

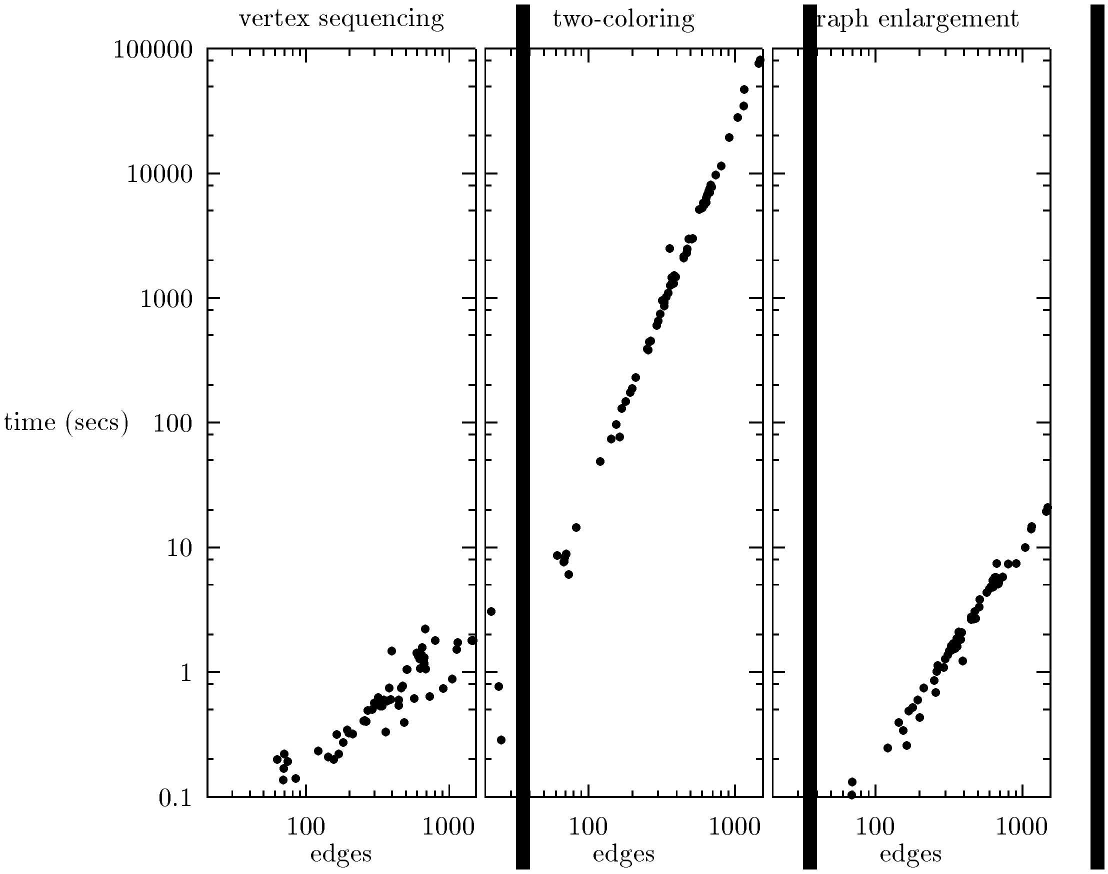

FIG. 5.5. *Average CPU time for 1000 GRASP iterations as a function of the number of edges of the input graph: vertex sequencing, two-coloring, and subgraph enlargement*

without local search. On the other hand, on only six instances, the solutions were worse or a solution of the same size was obtained in more iterations. This tradeoff, along with the fact that the observed running times for `LocalSearch` are negligible compared to the times taken by the two-coloring `SecondPhaseGT` (see Figure 5.5), recommends the use of `LocalSearch` in the GRASP for vertex sequencing.

Procedure `EnlargePlanar` in `GRASPforGP` not always improves the solution produced by `SecondPhaseGT`. Sometimes, however, it does produce subgraphs that improve the current best known solution. Since the observed running times for the graph enlargement phase are negligible compared to the times taken by the two-coloring `SecondPhaseGT` (see Figure 5.5), it is recommended that `EnlargePlanar` be used as described in `GRASPforGP` (Figure 4.2).

Both variants of the two-phase heuristic, as well as the GRASP, require at each iteration the solution of two maximum independent set problems on interval graphs. Though these problems can be solved exactly in polynomial time using Gavril's algorithm [10], the complexity of doing so is cubic in the number of edges of the input graph. As can be seen in Figure 5.5, this two-coloring accounts for most of the work in these algorithms.

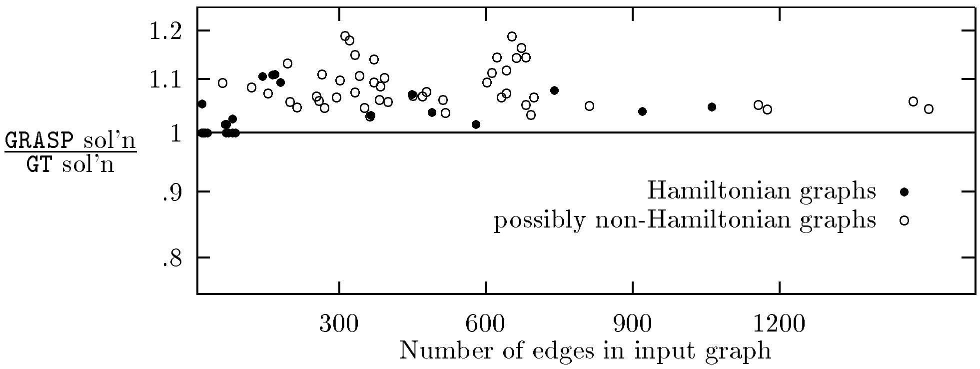

FIG. 5.6. *Ratio (log scale) of number of edges in planar subgraph found by* GRASP *and* GT *on Hamiltonian and possibly non-Hamiltonian input graphs as a function of the number of edges in the input graph*

In [11], the two-phase heuristic based on Hamiltonian cycles almost always outperformed the deterministic greedy variant of the same heuristic. Figure 5.6 compares GRASP with GT on Hamiltonian and possibly non-Hamiltonian input graphs. Our data does not suggest that Hamiltonian input graphs favor GT over GRASP. This may be because the GRASP produces good approximate Hamiltonian cycles, the Hamiltonian graphs are generally the smallest in the test set, or the fact that the deterministic greedy variant in [11] was limited to a single iteration.

6. Concluding remarks. In this paper we described a GRASP for finding approximate solutions to the graph planarization problem. Extensive experimental results indicate that the GRASP is consistently better than the greedy two-phase heuristic described by Goldschmidt and Takvorian [11]. On all instances considered in the experiment, the GRASP matched or outperformed the Hamiltonian cycle variant of the two-phase heuristic. On most of the instances, the GRASP found solutions at least as good as those produced by Mutzel [18, 21] using the branch-and-cut method of Jünger and Mutzel [17, 16]. The exception was on the `tg` test problem class, on which the solutions found by GRASP were worse than those of the branch-and-cut method.

The experiments indicate that approximate solutions of the two-coloring maximum independent set problems, obtained with the GRASP for maximum independent set of [9], is not sufficient to obtain high quality solutions. We use the exact algorithm of Gavril [10] in the current version of this code. This algorithm is responsible for a large portion of the running time of our code. A more efficient algorithm is described by Asano, Imai, and Mukaiyama [2]. Replacing the algorithm of Gavril by this new algorithm will probably lead to a much more efficient GRASP code.

GRASP can be easily implemented on a parallel computer since different GRASP iterations can be assigned to each processor [9, 22]. A parallel GRASP for graph planarization will probably achieve a speedup that, on average, is linear in the number of processors, making it possible to find yet larger planar subgraphs.

Acknowledgement. The authors acknowledge R. Cimikowski, O. Goldschmidt, and A. Takvorian for providing test data as well as offering some insightful discussions, M. Jünger for making suggestions that improved Section 5, P. Mutzel for running her branch-and-cut heuristic on several of the test problems considered in this paper, and two anonymous referees for several suggestions that greatly improved the paper.

REFERENCES

[1] D. Angluin and L. Valiant, *Probabilistic algorithms for Hamiltonian circuits and matchings*, J. Comp. Sys. Sci., 18 (1979), pp. 155–190.
[2] T. Asano, H. Imai, and A. Mukaiyama, *Finding a maximum weight independent set of a circle graph*, IEICE Transactions, E74 (1991), pp. 681–683.
[3] J. Cai, X. Han, and R. Tarjan, *An $O(m \log n)$-time algorithm for the maximal planar subgraph problem*, SIAM J. Comput., 22 (1993), pp. 1142–1162.
[4] R. Cimikowski, *Graph planarization and skewness*, in Proceedings of the 23rd Southeastern International Conference on Combinatorics, Graph Theory, and Computing, 1992, pp. 21–32.
[5] ———, 1995. Personal communication.
[6] ———, *An analysis of heuristics for the maximum planar subgraph problem*, in Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms, 1995, pp. 322–331.
[7] G. Di Battista and R. Tamassia, *Incremental planarity testing*, in Proceedings of the 30th IEEE Symp. FOCS, Chapel Hill, NC, 1989, pp. 436–441.
[8] T. Feo and M. Resende, *Greedy randomized adaptive search procedures*, Journal of Global Optimization, 6 (1995), pp. 109–133.
[9] T. Feo, M. Resende, and S. Smith, *A greedy randomized adaptive search procedure for maximum independent set*, Operations Research, 42 (1994), pp. 860–879.
[10] F. Gavril, *Algorithms for a maximum clique and a maximum independent set of a circle graph*, Networks, 3 (1973), pp. 261–273.
[11] O. Goldschmidt and A. Takvorian, *An efficient graph planarization two-phase heuristic*, Networks, 24 (1994), pp. 69–73.
[12] M. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, 1980.
[13] M. Hassan and G. Hogg, *A review of graph theory applications to the facilities layout problem*, OMEGA International J. of Management, 15 (1987), pp. 291–300.
[14] J. Hopcroft and R. Tarjan, *Efficient planarity testing*, J. ACM, 21 (1974), pp. 549–568.
[15] R. Jayakumar, K. Thulasiraman, and N. Swami, *$O(n^2)$ algorithms for graph planarization*, IEEE Trans. on Computer-Aided Design, 8 (1989), pp. 257–267.
[16] M. Jünger and P. Mutzel, *Maximum planar subgraphs and nice embeddings: Practical layout tools*, Tech. Rep. 93-145, Angewandte Mathematik und Informatik, Universität zu Köln, 1993. To appear in *Algorithmica*.
[17] ———, *Solving the maximum weight planar subgraph problem by branch and cut*, in Proceedings of the Third Conference on Integer Programming and Combinatorial Optimization (IPCO), G. Rinaldi and L. Wolsey, eds., 1993, pp. 479–492.
[18] ———, 1996. Personal communication.
[19] A. Lempel, S. Even, and I. Cedarbaum, *An algorithm for planarity testing of graphs*, in Proceedings of the Theory of Graphs International Symposium, Rome, Gordon and Breach, 1966, pp. 215–232.
[20] P. Liu and R. Geldmacher, *On the deletion of nonplanar edges of a graph*, in Proceedings of the 10th SE Conf. on Comb., Graph Theory, and Comp., Boca Raton, FL, 1977, pp. 727–738.
[21] P. Mutzel, *The maximum planar subgraph problem*, PhD thesis, Universität zu Köln, 1994.
[22] P. Pardalos, L. Pitsoulis, and M. Resende, *A parallel GRASP implementation for the quadratic assignment problem*, in Parallel Algorithms for Irregularly Structured Problems – Irregular'94, A. Ferreira and J. Rolim, eds., Kluwer Academic Publishers, 1995.
[23] M. Resende, L. Pitsoulis, and P. Pardalos, *Approximate solution of weighted MAX-SAT problems using GRASP*, tech. rep., AT&T Research, Murray Hill, NJ, 1996.
[24] M. Sarrafzadeh and D. Lee, *A new approach to topological via minimization*, IEEE Transactions on Computer-Aided Design, 8 (1989), pp. 890–900.
[25] L. Schrage, *A more portable Fortran random number generator*, ACM Transactions on Mathematical Software, 5 (1979), pp. 132–138.
[26] Y. Takefuji and K. Lee, *A near-optimum parallel planarization algorithm*, Science, 245 (1989), pp. 1221–1223.
[27] Y. Takefuji, K.-C. Lee, and Y. Cho, *Comments on "An $O(n^2)$ algorithm for graph planarization"*, IEEE Transactions on Computer Aided Design, 10 (1991), pp. 1582–1583.
[28] A. Takvorian, 1995. Personal communication.
[29] R. Tamassia and G. Di Battista, *Automatic graph drawing and readability of diagrams*, IEEE

Trans. Sys., Man, and Cyber., 18 (1988), pp. 61–79.