GRAPH PLANARIZATION

MAURICIO G.C. RESENDE AND CELSO C. RIBEIRO

Abstract. We survey graph planarization and related problems. We first describe variants and applications of graph planarization. Then we focus on algorithms. We begin by describing the branch-and-cut algorithm of Jünger and Mutzel (1996). Then, we review work on heuristics based on planarity testing and those based on two-phase procedures. Finally, computational results comparing algorithms for graph planarization are presented.

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 in F from G, is planar. This problem is also known as the *maximum planar subgraph* problem. A related and simpler problem is that of finding a *maximal planar subgraph*, which is a planar subgraph $G' = (V, E')$ of G such that the addition of any edge $e \in E \setminus E'$ to G' destroys its planarity.

Graph planarization is known to be NP-hard [15]. The proof of NP-completeness of its decision version is based on a transformation from the Hamiltonian path problem restricted to bipartite graphs. Although exact methods for solving the maximum planar subgraph problem have been recently proposed, most algorithms to date attempt to find good approximate solutions.

In this article, we survey graph planarization and related problems. In the next section, we describe variants and applications of the basic problem formulated above. Next, we describe the branch-and-cut algorithm of Jünger and Mutzel [11]. We then review work on heuristics based on planarity testing and those based on two-phase procedures. Finally, computational results are considered.

2. Variants and applications

An application of graph planarization arises in the design of integrated circuits, in which a graph describing the circuit has to be decomposed into a minimum number of layers, each of which is a planar graph [13]. Other applications arise from variants of the basic graph planarization problem.

One such variant is the *maximum weighted planar graph* problem, in which positive weights are associated with the edges of the graph and one seeks a planar subgraph of maximum weight. Note that the basic graph planarization problem is a special case of the maximum weighted planar graph problem, in which all edge

Date: July 6, 1998.
To appear in *Encyclopaedia of Optimization*, Kluwer Academic Publishers, 1999.
AT&T Labs Research Technical Report: 98.15.1.

weights are equal to one. An application of this problem to facility layout is described in [9]. A graph is built in which the vertices represent the facilities and the edges define the relationships between them. The weight of each edge is the desirability that the two facilities that define the edge be adjacent in the design. A maximum weighted planar subgraph corresponds to a feasible layout with maximum benefit. In this paper, the authors also propose simulated annealing and tabu search heuristics for the approximate solution of the maximum weighted planar graph problem. Constructive heuristics based on maintaining a triangulated subgraph while making node and edge insertions are given in Foulds and Robinson [7], Eades, Foulds, and Giffin [4], and Leung [14].

Another related variant is that of drawing a given graph such that the number of edge crossings is minimized. The *crossing number* problem has practical applications in circuit design and graph drawing, such as in CASE tools [20] and automated graphical display systems. One particular case is that of minimizing straight-line crossings in layered graphs. A GRASP and path relinking approach for the two-layer case is given in Laguna and Martí [12], where one can also find a survey of the literature. Algorithms for graph drawing are reviewed in Battista, Eades, Tamassia, and Tollis [3].

In the *planar augmentation* problem, one wants to determine the minimum number of edges that need to be added to a planar graph such that the resulting graph is still planar and at least k-connected, where k is usually fixed to two or three. This variant has applications in automatic graph drawing, as well as in the design of survivable networks [17].

3. An exact algorithm

An exact branch-and-bound algorithm for the weighted graph planarization problem was introduced by Foulds and Robinson [6], but was limited to small dense graphs. Only recently has there been a leap in the performance of exact methods for graph planarization with the branch-and-cut algorithm of Jünger and Mutzel [11], which we describe next.

Given a graph $G = (V, E)$, their approach uses facet-defining inequalities for the planar subgraph polytope $\mathcal{PLS}(G)$. Let x_e be a 0-1 variable associated with each edge $e \in E$, such that $x_e = 1$ if and only if edge e appears in the maximum planar subgraph of G. Furthermore, let $x(F) = \sum_{e \in F} x_e$, for $F \subseteq E$.

Trivial inequalities $0 \leq x_e \leq 1$ are implicitly handled by the linear programming (LP) solver. The inequality $x(E) \leq 3|V| - 6$ is added to the initial linear program. Let x be the optimal solution of the LP relaxation associated with some node of the enumeration tree. For $0 \leq \epsilon \leq 1$, let $E_\epsilon = \{e \in E \mid x_e \geq 1 - \epsilon\}$ and consider the graph $G_\epsilon = (V, E_\epsilon)$, to which the planarity-testing algorithm of Hopcroft and Tarjan [10] is applied. The algorithm stops if it finds an edge set F which induces a nonplanar graph in G. If the inequality $x(F) \leq |F| - 1$ is violated, it is added to the set of constraints of the current LP. The back edge of the path which proved the nonplanarity of the graph induced in G by F is removed and the planarity-testing algorithm proceeds, eventually identifying other forbidden subgraphs of the graph G_ϵ. Although these forbidden subgraphs do not necessarily define facets of $\mathcal{PLS}(G)$, they must contain facet-defining subgraphs. Facet-defining inequalities are identified as follows. Once a forbidden set F is found, where the inequality $x(F) \leq |F| - 1$ is violated, one successively deletes each edge $f \in F$ and applies the planarity-testing algorithm. If the graph induced by $F \setminus \{f\}$ is planar, then edge f is returned to F. In at most $|F|$ steps, F is reduced to a smaller edge set which induces a minimal planar subgraph, leading to the facet-defining inequality $x(F) \leq |F| - 1$ still violated by the current LP solution. Another simple heuristic searches for violated Euler facet-defining inequalities $x(F) \leq 3|V'| - 6$ or $x(F) \leq 2|V'| - 4$, where (V', F) is, respectively, a clique or a complete bipartite subgraph of G.

After an LP has been solved, its solution is exploited by the planarity-testing algorithm, to produce a feasible solution for the graph planarization problem. Such feasible solutions are used as lower bounds that are used not only for fathoming nodes in the branch-and-cut tree, but also for fixing variables using their reduced costs during a cutting plane phase. Other heuristics are implemented to enhance the practical performance of the algorithm.

Branching is done if no cutting plane has been found for the current infeasible solution. The variable chosen for branching is one with fractional value closest to 1/2, among those with maximum cost coefficient in the objective function.

4. Two-phase heuristics

The heuristics described in this section are based on the separation of the computation into two phases. The first phase consists in devising a linear permutation of the nodes of the input graph, followed by placing them along a line. The second phase determines two sets of edges that may be represented without crossings above and below that line, respectively. Takefuji and Lee [19] were the first to propose a heuristic using this idea. They use an arbitrary sequence of nodes in the first phase and apply a parallel heuristic using a neural network for the second phase. Takefuji, Lee, and Cho [18] claimed superior performance of the two-phase approach of Takefuji and Lee [19] with respect to the heuristics described in he previous section.

Their approach was later extended and improved by Goldschmidt and Takvorian [8]. In the first phase, these authors attempt to use a linear permutation of the nodes associated with an Hamiltonian cycle of G. Two strategies are used: (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 node in the linear permutation is a minimum degree node in G. After the first k nodes of the permutation have been determined, say $v_1, v_2, \cdots, v_k$, the next node v_{k+1} is selected from the nodes 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 node of G_k adjacent to v_k in G, then v_{k+1} is selected as a minimum degree node in G_k.

Let $H = (E, I)$ be a graph where each of its nodes corresponds to an edge of the input graph G. Nodes e_1 and e_2 of H are connected by an edge if the corresponding edges of G cross with respect to linear permutation of the nodes established during the first phase. A graph is called an *overlap graph* if its nodes 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 nodes 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 the linear permutation of its nodes.

The second phase of the heuristic of Goldschmidt and Takvorian consists in two-coloring a maximum number of the nodes 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 bipartite subgraph of the overlap graph H, i.e. a bipartite subgraph having the largest number of nodes. This problem is equivalent to drawing the edges of the input graph G above or below the line where its nodes have been placed, according to their linear permutation. A greedy algorithm is used to construct a maximal 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 nodes in $\mathcal{B}$ and all edges incident to nodes 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 so obtained induce a bipartite subgraph of the original overlap graph, not necessarily with a maximum number of nodes.

The linear permutation obtained in the first phase affects the size of the planar subgraph found in the second phase of the above heuristic. Moreover, it is not clear that the permutation produced by the greedy algorithm is the best. To produce possibly better permutations, randomization and local search have been introduced in the greedy algorithm by Resende and Ribeiro [16] in the form of a greedy randomized adaptive search procedure (GRASP).

A GRASP [5] is an iterative process, in which each 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 estimates the 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 at each 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, even 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.

Resende and Ribeiro [16] propose an extension of the above described heuristic of Goldschmidt and Takvorian, in which a GRASP is used for finding a linear permutation of the nodes. In the construction phase of this GRASP, the greedy algorithm used in the first phase by Goldschmidt and Takvorian is randomized: instead of selecting the node of minimum degree among those yet unselected, the selection is made from a set of low degree nodes. The local search phase of this GRASP explores the neighborhood of the current permutation by swapping the positions of two nodes at a time, attempting to reduce the number of possible edge crossings.

Incorporating the second phase of the Goldschmidt-Takvorian heuristic to the above GRASP for finding a linear permutation of the nodes results in a GRASP for graph planarization.

Each iteration of this GRASP produces three edge sets: $\mathcal{B}$ (blue edges), $\mathcal{R}$ (red edges), and $\mathcal{P}$ (the remaining edges, which are referred 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. In case this reassignment of colors is possible, then the size of the planar subgraph is increased by one edge. This post-optimization procedure is incorporated at the end of each GRASP iteration.

5. Computational Results

Detailed results on a set of 75 test problems described in the literature [2, 8] are reported in [16]. Here, we summarize computational results illustrating the effectiveness of the two-phase heuristics described in the previous section, as well as that of the exact branch-and-cut algorithm. These results are based on a Fortran implementation of the GRASP heuristic of Resende and Ribeiro [16], on the original code of the branch-and-cut algorithm of Jünger and Mutzel [11], and on published results for the heuristics of Takefuji and Lee [19] and Goldschmidt and Takvorian [8] (using the greedy algorithm for building the linear permutation) of the nodes.

Problem	Nodes	Edges	T-L	G-T	R-R	J-M
G1	10	22	20	20	20	20
G2	45	85	80	80	82	82
G3	10	24	21	21	24	24
G4	10	25	22	21	24	24
G5	10	26	22	21	24	24
G6	10	27	22	21	24	24
G7	10	34	23	22	24	24
G8	25	69	58	60	69	69
G9	25	70	59	60	69	69
G10	25	71	58	59	69	69
G11	25	72	60	59	69	69
G12	25	90	61	62	67	68
G13	50	367	70	131	135	125
G14	50	491	100	136	143	133
G15	50	582	101	142	144	138
G16	100	451	92	180	196	187
G17	100	742	116	219	236	213
G18	100	922	115	237	246	223
G19	150	1064	127	297	311	290

We give, in the table above, results comparing the four approaches on a subset of the test problems described in [8]. For each instance, the table lists the number of nodes, the number of edges, and the size of the planar subgraphs produced by each algorithm. A time limit of 1000 seconds (on a SUN SPARCstation 10/41) was imposed on the runs of the branch-and-cut algorithm and the best solution found was returned as a heuristic solution when optimality was not attained in that time limit. This time limit was reached on instances G12 to G19.

The results in this table show that the Goldschmidt-Takvorian algorithm is a substantial improvement over the neural network approach of Takefuji and Lee.

The GRASP consistently outperforms both other two-phase heuristics, not only for the problems reported in this table, but also for all of the remaining instances considered in [16].

A comparison of GRASP with the branch-and-cut algorithm depends heavily on the instances. The results reported in [16] can be separated into two groups. On 49 of the 55 instances in the first group, the GRASP either matched or produced better solutions than the branch-and-cut algorithm. On 30 of those 55 instances, the GRASP solution was strictly better than the branch-and-cut solution. Note that, on these instances, the branch-and-cut algorithm was forced to stop because of the 1000 second time limit. However, on all the remaining 20 instances, the branch-and-cut algorithm performs remarkably well and outperforms all other algorithms.

References

[1] D. Angluin and L.G. Valiant. Probabilistic algorithms for Hamiltonian circuits and matchings. *J. Comp. Sys. Sci.*, 18:155–190, 1979.

[2] R.J. Cimikowski. An analysis of heuristics for the maximum planar subgraph problem. In *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms*, pages 322–331, 1995.

[3] G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis. Algorithms for drawing graphs: An annotated bibliography. *Computational Geometry: Theory and Applications*, 1:235–282, 1994.

[4] P. Eades, L.R. Foulds, and J.W. Giffin. An efficient heuristic for identifying a maximum weight planar subgraph. *Lecture Notes in Mathematics*, 952:239–251, 1982.

[5] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[6] L.R. Foulds and R.W. Robinson. A strategy for solving the plant layout problem. *Operational Research Quarterly*, 27:845–855, 1976.

[7] L.R. Foulds and R.W. Robinson. Graph theoretic heuristics for the plant layout problem. *Int. J. Production Research*, 16:27–37, 1978.

[8] O. Goldschmidt and A. Takvorian. An efficient graph planarization two-phase heuristic. *Networks*, 24:69–73, 1994.

[9] M. Hasan and I.H. Osman. Local search algorithms for the maximal planar layout problem. *International Transactions in Operational Research*, 2:89–106, 1995.

[10] J. Hopcroft and R.E. Tarjan. Efficient planarity testing. *J. ACM*, 21:549–568, 1974.

[11] M. Jünger and P. Mutzel. Maximum planar subgraphs and nice embeddings: Practical layout tools. *Algorithmica*, 16:33–59, 1996.

[12] M. Laguna and R. Martí. Grasp and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 1998. to appear.

[13] T. Lengauer. *Combinatorial algorithms for integrated circuit layout.* John Wiley, 1990.

[14] J. Leung. A new graph-theoretic heuristic for facility layout. *Management Science*, 38:594–605, 1992.

[15] P.C. Liu and R.C. Geldmacher. On the deletion of nonplanar edges of a graph. In *Proceedings of the 10th SE Conf. on Comb., Graph Theory, and Comp.*, pages 727–738, Boca Raton, 1977.

[16] M.G.C. Resende and C.C. Ribeiro. A GRASP for graph planarization. *Networks*, 29:173–189, 1997.

[17] M. Stoer. *Design of survivable networks*, volume 1531 of *Lecture Notes in Mathematics*. Springer-Verlag, 1992.

[18] Y. Takefuji, K.-C. Lee, and Y.B. Cho. Comments on "An $O(n^2)$ algorithm for graph planarization". *IEEE Transactions on Computer Aided Design*, 10:1582–1583, 1991.

[19] Y. Takefuji and K.C. Lee. A near-optimum parallel planarization algorithm. *Science*, 245:1221–1223, 1989.

[20] R. Tamassia and G. Di Battista. Automatic graph drawing and readability of diagrams. *IEEE Trans. Sys., Man., and Cyber.*, 18:61–79, 1988.

Information Sciences Research, AT&T Labs Research, Florham Park, NJ 07932 USA.
E-mail address: mgcr@research.att.com

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