Partitioning planar graphs: a fast combinatorial approach for max-cut

F. Liers and G. Pardella

Institut für Informatik, Universität zu Köln, Pohligstraße 1, D-50969 Köln,Germany,
{liers, pardella}@informatik.uni-koeln.de,
http://cophy.informatik.uni-koeln.de

Abstract. The MAX-CUT problem asks for partitioning the nodes V of a graph $G = (V, E)$ into two sets (one of which might be empty), such that the sum of weights of edges joining nodes in different partitions is maximum. Whereas for general instances the MAX-CUT problem is NP-hard, it is polynomially solvable for certain classes of graphs. For planar graphs, there exist several polynomial-time methods determining maximum cuts for arbitrary choice of edge weights. Typically, the problem is solved by computing a minimum-weight perfect matching in some associated graph. The most efficient known algorithms are those of Shih et al. [45] and that of Berman et al. [9]. The running time of the former can be bounded by $O(|V|^{\frac{3}{2}} \log |V|)$. The latter algorithm is more generally for determining T-joins in graphs. Although it has a slightly larger bound on the running time of $O([V|^{\frac{3}{2}}(\log |V|)^{\frac{3}{2}})\alpha(|V|)$, where $\alpha(|V|)$ is the inverse Ackermann function, it can solve large instances in practice. In this work, we present a new and simple algorithm for determining maximum cuts for arbitrary weighted planar graphs. Its running time is bounded by $O(|V|^{\frac{3}{2}} \log |V|)$, similar to the bound achieved by [45]. It can easily determine maximum cuts in huge random as well as real-world graphs with up to 10^6 nodes. We present experimental results for our method using two different matching implementations. We furthermore compare our approach with those of [45] and [9]. It turns out that our algorithm is considerably faster in practice than [45]. Moreover, it yields a much smaller associated graph. Its expanded graph size is comparable to that of [9]. However, whereas the procedure of generating the expanded graph in [9] is very involved (thus needs a sophisticated implementation), implementing our approach is an easy and straightforward task.

1 Introduction

Graph partitioning problems in graphs have many relevant real-world applications. In its most basic version, the problem is to partition the nodes of a graph into two disjoint sets such that the weight of the edges connecting them is either minimum or maximum (assuming uniform weights in case the graph is unweighted). The former is denoted by MIN-CUT and the latter by MAX-CUT. Cut problems have many applications, e.g. in VIA minimization in the layout of

electronic circuits, [7], in physics of disordered systems [24, 23, 31], or in network reliability [2]. Furthermore, the problem is equivalent to unconstrained quadratic 0-1 optimization [11, 15]. Several important combinatorial optimization tasks can naturally be formulated as constrained quadratic optimization problems. Investing knowledge from the unconstrained case often drastically speeds up the solution algorithms [12].

For nonnegative edge weights, the MIN-CUT problem can be solved using network flow techniques due to the famous duality of maximum flows and minimum cuts in networks [18], or by the algorithm proposed in [46].

For general edge weights, the MAX-CUT problem (and by inversion of the sign of the edge weights also the MIN-CUT problem) is NP-hard. We refer to [35] and the references therein for a detailed study of different classes of instances marking the boundary between easy and hard ones. When restricting to certain graph classes, polynomial-time solution algorithms are known. This is true especially for planar graphs which are the subject of this article.

In 1972 Orlova and Dorfman [39] noticed that a maximum cut in a planar graph can be determined by finding shortest paths between odd-degree nodes in its dual. Using this observation, they designed a branch-and-bound algorithm for this task. Hadlock's algorithm [22, 3] was the first (combinatorial) polynomial-time algorithm for MAX-CUT on planar graphs with nonnegative edge weights. In [4, 8] Barahona proposes a MAX-CUT algorithm for (arbitrary weighted) planar grid graphs, focussing on solving the two-dimensional planar Ising spin glass problem from theoretical physics. Furthermore, [5, 6] present a method that reduces the task to the Chinese-Postman problem. The latter can be solved in $O(|V|^{\frac{3}{2}} \log |V|)$ on planar graphs. Moreover, Barahona introduced a polynomial-time algorithm for MAX-CUT on graphs not contractible to K_5. In 1990, Mutzel [37] proposed an algorithm using T-joins. In the T-join problem, let $T \subseteq V$ be a subset of nodes of graph $G = (V, E)$. The task is to find an edge set $J \subseteq E$ of minimum weight such that in (V, J) all nodes in T have odd degree, and all others have even degree. In the same year, Shih, Wu, and Kuo [45] presented a mixed MAX-CUT algorithm for arbitrary weighted planar graphs, which generalizes the algorithm for optimal layer assignment of Kuo, Chern, and Shih [29]. It solves the problem in time bounded by $O(|V|^{\frac{3}{2}} \log |V|)$ which is presently the algorithm with the best worst-case running time. The method constructs the dual graph. It is then expanded such that the complementary edge set of matchings in the latter correspond to cuts in the former. Moreover, a minimum-weight perfect matching in the latter yields an optimum cut in the primal graph.

In this work, we follow this general algorithmic scheme which leads to an algorithm with the same asymptotic running time as the one of Shih, Wu, and Kuo. However, in our transformation the expanded dual graph has a simpler structure and contains a considerably smaller number of both nodes and edges. As the bulk of the running time is spent in the matching computation and the latter scales with the size of the graph, the algorithm is much faster in practice. This is reflected in the computational results to be found in Section 8. This work is the full version of an earlier technical report [32]. The proposed MAX-

CUT algorithm for arbitrary weighted planar graphs is a generalization of the methods from [47, 41]. The latter focused on the determination of minimum cuts in two-dimensional grid graphs for determining ground states of Ising spin glasses in physics. The methods are based on the work of Kasteleyn [26] from the 1960s. Working independently, Schraudolph and Kamenetsky partly arrived at a similar method (cf. the recent technical report [43].) Their approach can be interpreted as a generalization of the work of Shih et al. [45]. They use the same idea proposed in [45], i.e. that the complementary edge set of matchings in an expanded dual graph yields an optimum cut in the primal graph. In contrast to [45] they do not rely on a triangulation of the primal graph as first step, but present an expansion rule for each dual node depending on its degree.

A max-cut on a planar graph can be found by solving the T-join (with T= $\emptyset$) problem on the dual graph. This fact was already utilized in the 1990s by Barahona and by Mutzel. Berman et al. [9] proposed an algorithm for T-joins on biconnected planar graphs. Using this algorithm, solutions for large instances could be determined. The authors reduced the T-join problem to a perfect matching problem on an auxiliary graph. Due to the studied application, they restricted themselves to nonempty T-sets. This restriction is not satisfied when studying max-cuts on planar graphs. However, according to our knowledge, the restriction to nonempty T-sets does not seem necessary for the correctness of the method. We will discuss a potential application of the algorithm [9] for determining max-cuts in more detail in Section 5. The method of Berman et al. then amounts to determining a perfect matching in a graph of roughly comparable size. However, the construction that we outline here is considerably easier to use and to implement than the one suggested in [9].

In the following, we introduce some basic definitions and notations. In Section 3 we introduce and illustrate the algorithm and prove its correctness in Section 4. An analysis of the running time and space demand is presented afterwards in Section 5. A comparison with known algorithms is given in Section 6. In Section 7 possible algorithmic modifications are proposed. Finally, we present running times on realistic and random instances. It turns out that the algorithm can routinely solve the problem for random maximum planar graphs with up to 500,000 nodes and for realistic instances on planar graphs with up to 1,200,000 nodes. For grid graphs coming from the physics application, i.e. determination of ground states for two-dimensional planar Ising spin glasses, we can solve instances with up to 3000^2 nodes. Compared to the method introduced by Shih et al. which has the best known worst-case running time, our method is considerably faster. It reaches comparable graph sizes than the method by [9], but is much easier to implement.

2 Preliminaries

We consider simple, undirected, planar graphs $G = (V, E)$ with node set V and edge set E. We assume G is connected and real-weighted, i.e. each edge $e \in E$ is assigned a weight $w(e) \in \mathbb{R}$. Multiple edges between two nodes can

be contracted to a single edge with weight equal to the sum of the weights of these multiple edges. Graphs with more than one connected component can easily be solved independently. Self-loops can be omitted, as those edges will never be cut-edges. If not stated otherwise, we assume $|V| = n$ and $|E| = m$. Let $deg(v)$ denote the degree of a node $v \in V$, i.e. the number of edges incident to node v (counting self loops twice). A *path*, $\pi = v_1, v_2, \ldots, v_k,\ v_i \in V,\ i \in \{1, \ldots, k\},\ k \leq n$, is a sequence $\{v_1, v_2, \ldots, v_k\}$ of pairwise different v_i such that $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)$ are edges of G. A *closed path* $\pi = v_1, v_2, \ldots, v_k$ with distinct v_i $(i = 1, \ldots, k-1)$, (v_i, v_{i+1}) $(i = 1, \ldots, k-1)$ edges, and $v_1 = v_k$, is called a cycle. A *subgraph* H of G is a graph such that every node of H is a node of G, and every edge of H is an edge in G also. We denote with K_n the complete graph with n nodes. Let $G = (V, E)$ be a weighted graph. For each (possibly empty) subset $Q \subseteq V$, the *cut* $\delta(Q)$ is the set of all edges $e = (u, v)$ with $u \in Q$ and $w \in V \setminus Q$. The weight of a cut is given by $w(\delta(Q)) = \sum_{e \in \delta(Q)} w(e)$. A *minimum cut* (MIN-CUT) asks for a cut $\delta(Q)$ with minimum weight $w(\delta(Q))$. As MAX-CUT is equivalent to MIN-CUT by negating weights, we concentrate on the minimization version of the problem. A connected graph $G = (V, E)$ is called *Eulerian* if and only if E can be partitioned into edge-disjoint cycles which is equivalent to saying that each node of G has even degree. A graph G is *planar* if it can be embedded in the plane in such a way that no two edges meet each other except at a node to which they are both incident. If a graph G is planar, then any embedding of G divides the plane into regions, called *faces*. One of these faces is unbounded, and called the *outer face*. A *geometric dual graph* G_D of a planar graph G is a planar graph with the following properties: G_D has a node for each face of G, and an edge for each edge joining two neighboring faces (including self-loops and multiple edges). A *matching* in a graph $G = (V, E)$ is a set of edges $M \subseteq E$ such that no node of G is incident with more than one edge in M. If some edge $m \in M$ is incident with a node $v \in V$, then v is *M-covered*, otherwise v is *M-exposed*. A matching M is *perfect* if every node is M-covered. The weight of M is the sum of weights of the edges in M.

3 The Algorithm

In the following we assume we are given a planar embedding of G. At first, we calculate its dual graph $G_D = (V_D, E_D)$, where the weight of a dual edge is chosen as $w(\tilde{e}) = w(e)$ if $\tilde{e} \in E_D$ is the dual edge crossed by $e \in E$. Subsequently, we split all dual nodes $\tilde{v} \in V_D$ with degree $deg(\tilde{v}) > 4$ into $\lfloor (deg(\tilde{v}) - 1)/2 \rfloor$ nodes and connect the copies by a path of new edges receiving zero weight. Let *split nodes* denote nodes created by a splitting operation. Edges incident to the primal node are equally distributed among the split nodes such that the degree of each node is at most four, cf. Figure 1. We denote the resulting graph by $G_t = (V_t, E_t)$.

It is easy to see that after the splitting operations, no node in G_t has a degree smaller than three. Indeed, each face in a connected planar graph G (with $n > 2$) is bounded by at least two edges. Bounding a face by exactly two

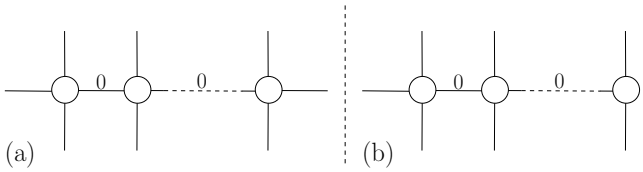

Fig. 1. (a) Node with even degree > 4 is split up in $\lfloor (deg(\tilde{v}) - 1)/2 \rfloor$ nodes. Edges are equally distributed among the split nodes. (b) Node with odd degree > 4 is split into $\lfloor (deg(\tilde{v}) - 1)/2 \rfloor$ nodes, each with degree four, except one receiving degree three.

edges is only possible if G has multiple edges which contradicts its simplicity. As G_D is the dual of G, we conclude that each node in G_D has degree at least three. Furthermore, a node in the transformed graph G_t has degree three or four.

The connectedness of G and G_D means that G_t is also connected. Moreover, certain structures will never occur in G_t. For example, degree-four nodes with all edges being self-loops contradict the connectedness of the primal graph G. From now on, we assume that G_t does not contain degree-four nodes with only self-loop edges.

We note that in the special case that G is a path graph of length two or three, P_2 (P_3), a node with degree two having a self-loop (with degree four having two self-loops, respectively) occurs. The algorithm we are going to present works here as well, so we do not have to take special care of this case.

Next, we expand each node in G_t to a K_4 (a so-called Kasteleyn city [26]), while keeping the weights of the edges. Newly generated edges again receive zero weight. A node in G_t of degree three is expanded as displayed in Fig. 2(a), a degree-four node as shown in Fig. 2(b). Nodes with self-loops are expanded as shown in Fig. 2(c) and (d). We denote the resulting graph by G_E. Next,

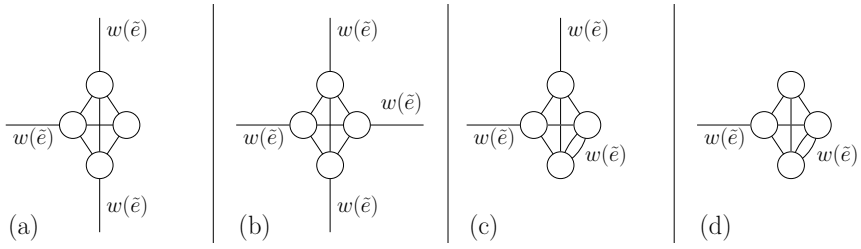

Fig. 2. Expansion of the nodes in G_t to K_4's. (a) shows the subgraph for a node with degree three. (b) is generated in case the node has degree four. All edges in K_4 receive zero weight. Expansion for nodes having self-loops is shown in (c) and (d). (c) is the subgraph for a node with degree four and one self-loop. (d) is generated in case the node has degree three and one self-loop. All edges in K_4 receive zero weight.

we calculate a minimum-weight perfect matching M in G_E. Subsequently, we

undo the transformation, i.e., shrink back all K_4's and all (possibly created) split nodes, while keeping track of the matched edges. Consider the subgraph induced by the matching edges that are still present in the dual graph after shrinking. We will show in the next section that each node in this subgraph has even degree. This means that it is a minimum weight Eulerian graph which yields an optimum cut in the primal graph. We state the complete algorithm in Alg. 1.

Algorithm 1 MAX-CUT algorithm for planar graphs

Input: EMBEDDING OF A SIMPLE, CONNECTED PLANAR GRAPH G
Ouput: MAX-CUT $\delta(Q)$ OF G

1. BUILD DUAL GRAPH G_D
2. SPLIT EACH NODE $v \in G_D$ WITH $deg(v) > 4$ AND CALL RESULTING GRAPH G_t
3. EXPAND EACH NODE $v \in G_t$ TO A K_4 AND CALL RESULTING GRAPH G_E
4. COMPUTE MINIMUM-WEIGHT PERFECT MATCHING M IN G_E
5. SHRINK BACK ALL ARTIFICIAL NODES AND EDGES WHILE KEEPING TRACK OF MATCHED DUAL EDGES
6. MATCHED DUAL EDGES IN G_D INDUCE OPTIMUM EULERIAN SUBGRAPHS AND THUS OPTIMUM MAX-CUT $\delta(Q)$ OF G
7. **return** $\delta(Q)$

The following section clarifies the algorithmic flow on some small example.

3.1 Example

Consider as an example the planar drawing of a graph $G = (V, E)$ in Fig. 3.

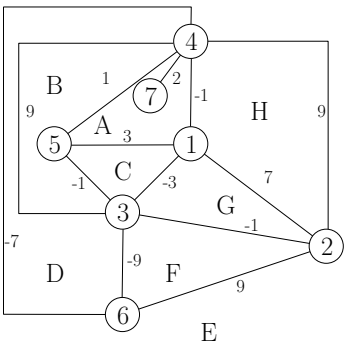

Fig. 3. Planar drawing of a planar graph G. Nodes are labeled by numbers. Edge weights are given as subscripts. Capital letters represent faces.

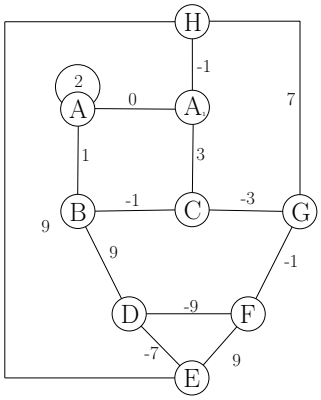

Fig. 4. The transformed dual graph G_t in which node A is split.

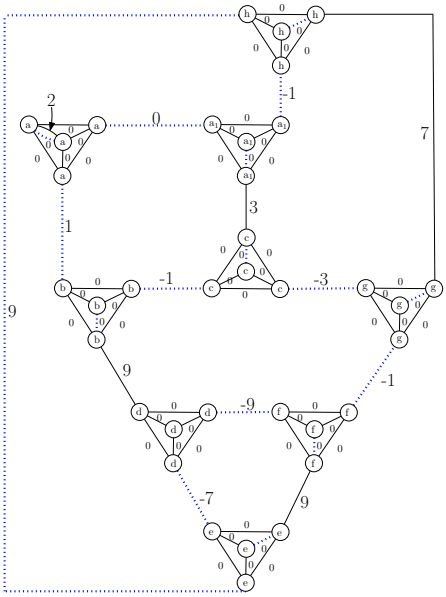

Fig. 5. The transformed dual graph G_E after expansion, together with a minimum-weight matching (dotted (red) edges).

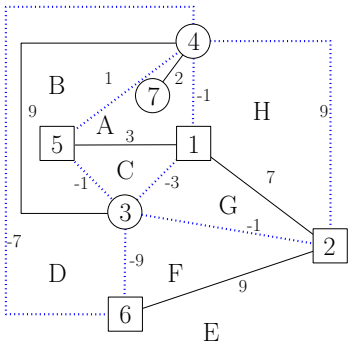

Fig. 6. After shrinking back the node copies, we have an optimum MIN-CUT $\delta(Q)$ with weight $w(\delta(Q)) = -12$. Dotted (red) edges are cut-edges. Node partitions are indicated by different node shapes.

G consists of seven nodes $V = \{1, 2, \ldots 7\}$ and eight faces $V_D = \{A, B, \ldots H\}$. Nodes are labeled with numbers and faces with capital letters. Edge weights are given as edge subscripts. The dual node A (representing face A) has degree greater four, and is split into two nodes (step 2 of Alg. 1), cf. Fig. 4. No other dual node has degree greater four; thus, no more split operations are needed. Fig. 5 shows the graph after having expanded each node in G_t to K_4's (step 3). On the transformed and expanded graph (Fig. 5) we calculate a minimum-weight perfect matching (dotted edges), which is step 4 of Alg. 1. Shrinking back all artificial nodes (step 5) yields a minimum-weight Eulerian subgraph of the dual as the dotted edges form edge disjunct cycles forming the edge set of the induced subgraph (step 6), and thus a MIN-CUT of the primal graph (cf. Fig. 6).

4 Correctness of the Algorithm

It is well known that there is a one-to-one correspondence between Eulerian subgraphs in the dual and cuts in its primal graph. In fact, for a cut edge (u, v) in G, its dual edge 'crossing' (u, v) is contained in the corresponding Eulerian subgraph, and vice versa.

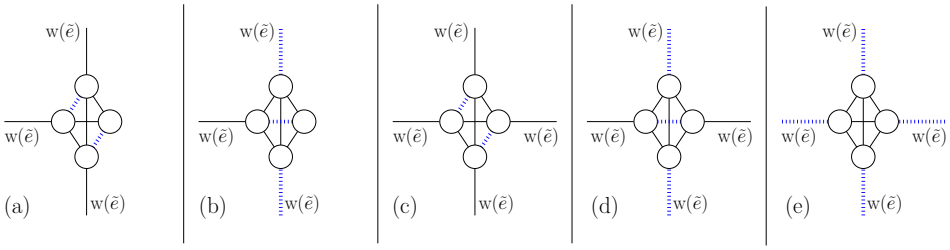

Fig. 7. Different cases for matched edges in a K_4 subgraph together with outgoing edges, modulo cases in which the same number of outgoing edges is matched. (a) and (b) show possible matchings for subgraphs representing nodes with degree three in G_t. Figures (c), (d) and (e) show the possible matchings for subgraphs representing a node with degree four. Dotted (blue) edges are matching edges.

In this section, we show that the edge set induced by the minimum-weight perfect matching in G_E corresponds to a minimum-weight Eulerian subgraph in the dual and therefore to a minimum cut in the primal graph.

To this end, we first need to show that there always exists a perfect matching M in the expanded graph G_E. Then, we need to prove that the constructed perfect matching in G_E induces a subgraph in the dual in which all node degrees are even.

We postpone for a moment the proof that a perfect matching exists. We call edges not contained in a K_4 *outgoing* and count the number of matched outgoing edges on some K_4 for an arbitrary perfect matching in G_E. Modulo analogous cases, we show in Figs. 7 and 8 all different possibilities for a matching covering all nodes of a K_4-subgraph in G_E together with its different number of outgoing edges. Analogous cases are those yielding the same number of outgoing matching edges.

Clearly, any possible matching in a K_4 subgraph leads to either zero, two or four outgoing matching edges with all nodes are M-covered. An odd number of outgoing matching edges always leaves an odd number of K_4 nodes unmatched, i.e. M-exposed, which contradicts the matching's perfectness.

Now we prove that the transformed graph G_E indeed has a perfect matching M. We first note that the graph is connected and has an even number of nodes (due to the inflation of each node to a K_4). A trivial perfect matching exists as in each K_4 all nodes can be covered by matching edges contained in the K_4 (cf. Figs. 7 (a) and (c) and 8 (a) and (c)). Therefore, a perfect matching in G_E always exists. One might ask whether there also always exists another perfect matching in which not only artificial edges contained in the K_4's are matched. Indeed, as we deal with a geometric dual graph G_D, any two adjacent nodes in G_D (i.e., adjacent faces in G) are connected by at least one simple cycle. This cycle is expanded but preserved during the transformation of G_D to G_E. Thus, a possible nontrivial matching in G_E may match the edges in the cycle and additionally in each K_4 subgraph (representing a node on the cycle in G_D) an

edge connecting two unmatched K_4 nodes, as shown in Figs. 7 (b), (d) and (e). For all other Kasteleyn cities, edges contained in the K_4 can be matched. Each K_4 subgraph then has an even number of possible outgoing matching edges, cf. Figs. 7-8.

Shrinking back the artificial nodes to the corresponding split nodes does not affect the number of outgoing matching edges. Consequently, after having collapsed all split nodes back to its dual nodes, each dual node has an even number of adjacent matching edges, too. Hence the matching induced subgraph is Eulerian and therefore defines a cut $\delta(Q)$ in the primal graph G.

Yet the minimality of the cut is to be proven. It is

$$\begin{aligned} w(M) &= \sum_{\tilde{e} \in E_D \cap M} w(\tilde{e}) \\ &\underbrace{=}_{w(\tilde{e})=w(e)} \sum_{e \in \delta(Q)} w(e) \\ &= w(\delta(Q)) \end{aligned}$$

As $w(M)$ is the weight of a minimum-weight perfect matching, the weight of the induced Eulerian subgraph is minimum, and thus the weight of the cut $\delta(Q)$, too. We summarize this in the next theorem.

Theorem 1. *The algorithm described above computes a* MIN-CUT *(or* MAX-CUT*) in an arbitrarily weighted planar graph.*

5 Running-Time Analysis

After having shown the correctness of the method, we now concentrate on establishing bounds on its running time. We consider a maximum planar graph with n nodes, i.e. a triangulated planar graph in which each face is enclosed by a simple cycle of three edges. We will argue in the following that among all planar

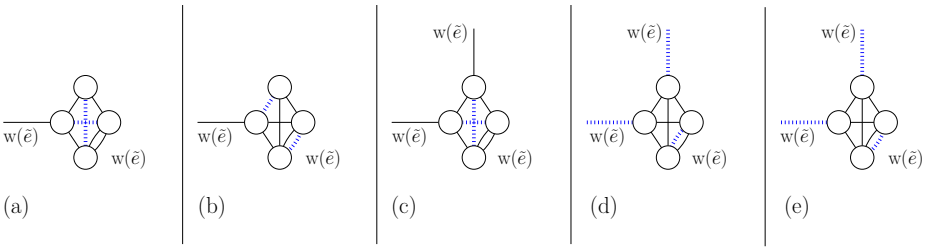

Fig. 8. Different cases for matched edges in a K_4 subgraph representing a node with self-loops (modulo analogous cases), together with outgoing edges. Dotted (blue) edges are matching edges. In Figures (b) and (e) the number of outgoing matching edges is two or four, resp., as the edge is matched that represents a former self-loop. This is in contrast to case (d) where the number of outgoing matching edges is two.

graphs with a specific number of nodes, the transformed graph G_E contains the maximum number of nodes and edges if G is triangulated.

Indeed, among all planar graphs with n nodes triangulated graphs have the maximum number of faces. For this class of graphs, the algorithm does not split any dual node. A triangulation of a face with k nodes leads to $k-2$ new faces (new dual nodes, respectively), whereas a splitting operation yields only $\lfloor (k-1)/2 \rfloor$ nodes for the same face. As splitting operations result in a smaller number of nodes, we also need fewer edges (connecting split nodes) as in the triangulated case.

Obviously, given an embedding of a (maximum) planar graph, one can calculate the geometric dual in time bounded by $O(n)$. Furthermore, the described expansion of the dual graph can be done in time linear in n, (there are at most $3n-6$ edges). Next, the most time consuming step is performed - the calculation of a minimum-weight perfect matching.

Edmonds [17, 16] introduced one of the fundamental results in combinatorial optimization, i.e. the polynomial-time blossom algorithm for computing minimum-weight perfect matchings. In its original version the algorithm runs in time bounded by $O(mn^2)$. Improved to $O(n^3)$ by Lawler [30] and Gabow [19] and later on by Gabow to $O(n(m + n \log n)$ [20]. Focusing on planar graphs, Lipton and Tarjan [33] presented an $O(n^{\frac{3}{2}} \log n)$ divide-and-conquer algorithm for finding maximum-weight matchings using the planar separator theorem.

As the transformed graph G_E is not planar, this algorithm cannot be applied directly. However, a good separator of size $O(\sqrt{n})$ can be found for the planar dual graph G_D which directly implies a good separator of size $O(\sqrt{n})$ for G_t. Therefore, we can use the algorithms [34, 19, 17, 16], and can calculate a minimum-weight perfect matching in G_E in time bounded by $O(n^{\frac{3}{2}} \log n)$. Similar arguments were already used by Shih et al. and others.

Finally, all nodes blown up in the transformation are shrunk back. Unshrinking can be done again in time $O(n)$. With these considerations we state the following theorem.

Theorem 2. *Using the method described above, a* MIN-CUT *(or* MAX-CUT*) in a planar graph can be determined in time bounded by* $O(n^{\frac{3}{2}} \log n)$.

6 Comparison with Existing Algorithms

It is interesting to compare our algorithm with that of [45] as well as with that of [9]. We show next that the method outlined above is less space demanding than the construction of [45] and leads to an algorithm that is faster in practice. Let F denote the set of faces of a maximum planar graph. The method constructs a graph G_E with at most $|V_E| = 4|F| = 4(2n-4)$ nodes, as for each dual node we create four nodes, and the number of dual nodes in a maximum planar graph is at most $2n-4$. Its number of edges is $6|F| + |E_D| = 15n - 30$, as we need to consider the original dual edges and those edges that are generated by the transformation of each dual node to a K_4 with six edges.

The transformation by Shih, Wu, and Kuo transforms each dual node of the triangulated primal graph to a "star" graph of seven nodes and nine edges, cf. Fig. 9. On this graph a minimum-weight perfect matching is calculated which

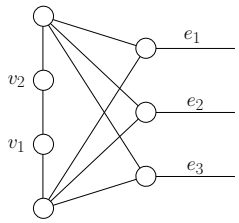

Fig. 9. By the method of Shih et al. every dual node of a triangulated primal graph is transformed to a so-called star graph of seven nodes and nine edges. Original dual edges are labeled $e_i, i = 1, 2, 3$.

yields a maximum even-degree edge set of the dual graph, and therefore a MAX-CUT of the primal graph. Thus, the method of Shih, Wu, and Kuo [45], yields an expanded dual graph with at least $7(2n - 4)$ nodes and $21n - 42$ edges. These bounds are sharp as the first step of Shih, Wu, and Kuo is a triangulation of the graph. The transformation outlined above, in comparison, computes a matching on a much smaller and sparser graph, even in the case the graph is a triangulation. This makes the proposed method in practice faster than the latter. Finally, the algorithm is easier to implement as we only need to take the dual graph, expand it accordingly using easy rules, compute a matching and undo the transformation steps again.

The star graph expansion by Shih et al. is necessary in order to enforce the determination of non-trivial, i.e., nonempty, optimum cuts as the matching induced even-degree edge set may otherwise be empty. In this case an additional $O(n)$ time step is needed to compute a nontrivial even-degree edge set. A modification of the "star" subgraphs is performed, and the matching is recalculated.

We note that in case the empty set is a valid optimum solution, the star graph in [45] can be replaced by a simple K_3 which also results in smaller expanded graphs G_E (with $6n - 12$ nodes and $9n - 18$ edges).

However, the triangulation step is still needed. Generally, G_E will be of larger size compared to the expanded graph constructed with the method proposed in Section 3, even in the case when using K_3 instead of star graphs. We show results for the resulting algorithm in Section 8.

As mentioned in the Introduction, a max-cut on a planar graph can be determined by finding an optimum T-join (with T= $\emptyset$). Berman et al. [9] proposed a reduction of the T-join problem on planar graphs to a perfect matching problem in an auxiliary graph. They restrict themselves to nonempty T-sets. However, this restriction does not seem necessary for ensuring algorithmic correctness. Therefore, we will give a comparison between their method and the one presented here. In [9], the auxiliary graph is built by expanding all nodes to gadgets.

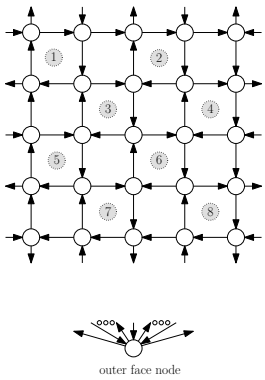

Fig. 10. The dual graph of a 6×6 primal two-dimensional planar grid, together with an edge orientation. For ease of presentation, edges to the outer face node are omitted. Gray-shaded numbers indicate the orientation order of cycles.

The structure of the gadget depends on the node degree. As the construction is rather complicated, we refer to [9] for the details. The size of the resulting graph also depends on a so-called fan-out direction (edge orientation) which is computed in the first step, cf. Lemma 4 in [9]. If the dual graph has n_D nodes and m_D edges, the auxiliary graph has at most $2m_D$ nodes and $6m_D - 5n_D$ edges, (cf. Theorem 3 in [9]). Let us compare the graph sizes for maximum planar graphs which constitute the worst case for our method. Then, the number of dual edges is $m_D = 3n_D - 6$. Our construction based on K_4 gadgets leads to a graph with $4n_D$ nodes and $9n_D - 6$ edges. This has to be compared with the size of the auxiliary graph of Berman et al. which has at most $6n_D - 12$ nodes and $13n_D - 36$ edges.

In order to compare the sizes of the expanded graphs of our and that of the method by Berman et al. in more detail, we compare in the following the expanded graph sizes for two-dimensional grid graphs of size $L \times L$.

First, we orient the edges in a straightforward way. An example is given in Fig. 10, where a possible orientation is shown for the dual of a 6×6 grid graph.

The orientation is chosen in the following way respecting Lemma 4 by Berman et al.: first orient the C_4 in a chessboard manner, see Figure 10. We are then left with edges at the border of the grid and edges incident to the outer-face node which form cycles of length 3 or 2. They can be oriented either clockwise or counter-clockwise.

All nodes have degree 4, except that of the outer face which has degree $4(L-1)$. For the outer face node, the gadget $S_{4(L-1)}$ has to be built recursively using basic gadgets, cf. Lemma 5 in [9]. All other nodes are replaced by S_4-gadgets. Then, the overall number of nodes $|V_{\mathrm{Ber}}|$ and edges $|E_{\mathrm{Ber}}|$ in the expanded graph is

$$|V_{\mathrm{Ber}}| = 4L^2 - 4L$$

and

$$|E_{\text{Ber}}| = 8L^2 - 6L - 5$$

This has to be compared to the size of the expanded graph obtained with the transformation presented here. For two-dimensional grids of size $L \times L$, the expanded graphs have

$$|V_E| = 4L^2 - 4$$

many nodes and

$$|E_E| = 8L^2 - L - 13$$

many edges.

Therefore, the expanded graphs are almost equal in size. Consider for example grid graphs of size 10×10 (100×100, 1000×1000). Using the orientation as introduced above, the expanded graphs using the method by Berman et al. consist of 360 (39 600, 3 996 000) nodes and of 735 (79 395, 7 993 995) edges. Using the method presented here, it consists of 392 (39 992, 3 999 992) nodes and 777 (79 887, 7 998 987) edges. It follows that our construction needs for these sizes $< 3\%$ more nodes and edges, which is a negligible number.

This shows that the resulting auxiliary graphs are of the same order of magnitude. As the largest node and edge numbers are obviously reached for different graph classes, there is no clear winner with respect to the sizes of the graphs. However, the major advantage of our method is that it avoids complicated graph constructions. All expansion steps are straightforward. The correctness of the algorithm can easily be understood. Moreover, the method is considerably easier and quicker to implement than that of [9]. Finally, our algorithm runs in $O(|V|^{\frac{3}{2}} \log |V|)$, which is faster than the running time $O(|V|^{\frac{3}{2}} (\log |V|)^{\frac{3}{2}}) \alpha(|V|)$ (Theorem 4) of the algorithm by Berman et al. This difference is due to the fact that in our case the arguments from the planar separator theorem are applicable, as noted above.

7 Algorithmic variants

In this section we present straightforward algorithmic modifications for the computation of nonempty optimum cuts and for optimum cuts respecting further constraints. These variants are motivated by the application of max-cut in physics. In the latter, energy-minimum states of so-called Ising spin glasses are determined. Additionally, some predefined nodes are forced to be in the same or in different shores. Therefore, we need to be able to either force a certain subset of edges to be contained in the cut or to prohibit certain edges to be contained in it.

The first variant, which we call FCE algorithm, deals with the task of determining an optimum cut respecting the additional requirement that a certain subset of edges must be contained in it. For example, this allows the calculation of nonempty MIN-CUTs in a graph. On the other hand, it permits to find an $s-t$ cut in the graph if nodes s and t are connected by an edge or can be connected by an edge without destroying the planarity of the graph. The second variant,

called PCE algorithm, can be seen as the reverse operation. Here, we need to determine the best cut that does not contain some specific subset of edges.

Similar results could be achieved by assigning a very large or a very small edge weight to edges that are forced in or out of the cut. However, the methods we propose here are more elegant and avoid numerical problems which may occur by using such large or small edge weights.

All operations explained in the following can be performed in time linear in n. We start with the first variant followed by a brief explanation of the second one.

Fixed cut edges - fce algorithm In order to force an edge $e = (v, w) \in E$ to be contained in $\delta(Q)$, we let $e_E = (v_E, w_E) \in E_{G_E}$ be the corresponding edge in the expanded graph G_E. We denote with $G_E \setminus \{v_E, w_E\}$ the graph that arises from G_E by deleting nodes v_E and w_E and all incident edges to v_E and w_E. Clearly, a minimum-weight perfect matching on $G_E \setminus \{v_E, w_E\}$ yields a constrained MIN-CUT $\delta(Q)$ in G, where nodes v and w belong to different node sets.

Theorem 3. *Algorithm* FCE *calculates a* MIN-CUT $\delta(Q)$ *satisfying the constraint* $e = (v, w) \in \delta(Q)$.

Proof. We only need to consider the case in which $e = (v, w) \notin \delta(Q)$. Let $e_D \in E_D$ be the edge corresponding to e in $G_D = (V_D, E_D)$. As $e \in \delta(Q)$ should hold, e_D and thus $e_E = (v_E, w_E)$ have to be matched. Therefore, removing v_E and w_E from G_E and identifying edge e_E as matching edge, these nodes are M-covered. Their removal yields still a perfect matching in which each dual node has an even degree of matching edges after the shrink operation. Being more concrete, the following situations can occur in $G_E \setminus \{v_E, w_E\}$: a K_4 subgraph is reduced to a K_3 which has either one or three outgoing matching edges. A K_4 subgraph can be reduced to a K_2, here again there are two possibilities for outgoing matching edges. Either zero or two outgoing edges are matched. Finally, a K_4 subgraph can completely be removed. In all cases the sum of removed edges e_D (forced matched) and matched edges is even at each dual node. The minimality follows directly from Theorem 1. □

As a direct consequence we conclude the following corollary.

Corollary 1. *Running the* FCE *algorithm m times, each time with a different fixed cut-edge, a nonempty* MIN-CUT *in G can be computed in time $O(mn^{\frac{3}{2}} \log n)$.*

Moreover, we can state

Corollary 2. *Let $G = (V, E)$ be a planar graph and $s, t \in V$ two distinguished nodes. Using the* FCE *algorithm an s-t cut can be calculated iff s and t lie on the same face of G.*

Proof. Let s and t lie on face f of G. If $e = (s, t) \notin E$, then insert e with weight zero to G and denote the resulting graph by $G \cup \{e\}$. Run the FCE algorithm with e fixed on $G \cup \{e\}$. □

Prohibited cut edges - pce algorithm Algorithm PCE forces certain nodes u, r to the same partition in case either $(u, r) \in E$ or the edge (u, r) can be inserted in E without destroying planarity. Adjacent nodes u and r are forced to be in the same cut set by excluding edge $e = (u, r)$ from the set of potential cut-edges. If edge $e \notin E$ we can add e if u and r lie on the same face of G so that e does not destroy planarity.

Suppose $e \in E$ with the constraint $\delta(Q) \cap e = \emptyset$. Now, let $e_E = (u_E, r_E) \in E_{G_E}$ be the corresponding edge in the expanded graph G_E. We consider the graph $G_E \backslash e_E$ that arises from G_E by removing edge e_E. Apparently, a minimum-weight perfect matching will never match this edge, thus it will never be a cut-edge. The correctness of the algorithm follows using similar arguments as for algorithm FCE.

8 Experiments

We implemented the algorithm from Section 3 and the method proposed by Shih et al. using the OGDF library [38]. We applied the implementations to a variety of problem instances, both realistic and randomly generated. All computational tests were carried out on Intel® Xeon© CPU E5410 2.33GHz (running under Debian Linux 4.1.1-21).

For the matching computations, we used two publicly available implementations as a black box. One implementation is Blossom IV written by Cook and Rohe [13] which is one of the fastest state-of-the-art matching implementations. Kolmogorov [28] very recently presented Blossom V as a new matching implementation. We discuss experimental results for our application with both matching codes.

We verified the correctness of our programs by checking the results for smaller sizes against those returned by the Cologne spin glass server [14], against published results [21] coming from VIA minimization instances, and against results obtained by a MAX-CUT implementation in SCIL [44], which are all based on different methods and programs. Moreover, we compared the results for grid graphs with those obtained by an implementation of the algorithm of Bieche et al. [10].

As triangulated graphs are the worst case for our method, we generated random triangulated graphs with either uniformly distributed or Gaussian distributed edge weights, created using the Standford Graph Base [27] as part of the graph generator rudy. The uniform distributed edge weights range between -100 and $+100$.

We denote by B4 the algorithm proposed here using Blossom IV for determining optimum matchings. B5 denotes our algorithm using Blossom V instead. SWK is our implementation of the original algorithm by Shih et al. using Blossom V without modifications. The modified SWK algorithm using K_3 in the expansion steps is denoted by SWK(K_3). In Table 1 average running times (in sec.) for minimum cut computations are given for various large random maximum planar graphs ($|V| = 5,000,\ 15,000,\ 80,000,\ 100,000,\ 500,000$). We studied 100 in-

$\%(w(e) < 0)$	10			30			50			70			100			Gaussian		
$\|V\|$	B4	B5	SWK	B4	B5	SWK	B4	B5	SWK	B4	B5	SWK	B4	B5	SWK	B4	B5	SWK
5,000	0.36	**0.40**	0.69	0.73	**0.46**	0.79	0.78	**0.47**	0.79	0.78	**0.46**	0.78	0.67	**0.46**	0.81	1.28	**0.35**	0.77
15,000	2.92	**1.53**	3.08	8.19	**1.80**	3.43	9.83	**1.85**	3.42	9.80	**1.85**	3.38	8.52	**1.90**	3.46	13.86	**1.64**	3.24
50,000	16.18	**6.28**	12.25	49.46	**7.40**	13.63	54.03	**7.69**	13.54	53.86	**7.69**	13.75	47.13	**7.70**	13.91	77.25	**6.40**	12.82
80,000	27.47	**9.18**	20.57	88.58	**10.88**	22.76	95.96	**11.10**	23.01	95.02	**11.30**	23.87	82.28	**11.51**	23.74	137.89	**10.21**	22.85
100,000	34.36	**8.67**	22.19	110.82	**10.42**	24.62	120.64	**10.98**	25.14	120.66	**11.04**	25.14	103.15	**11.18**	25.37	178.79	**10.43**	25.14
500,000	127.37	**44.95**	209.32	425.42	**56.06**	223.13	483.78	**60.11**	235.55	484.79	**61.53**	229.38	438.97	**63.49**	239.78	715.08	**58.79**	229.36

Table 1. Random instances. Average running times (in sec.) for various large maximum planar graphs. For the uniform edge weights, "% $(w(e) < 0)$" indicates the percentage of edges with negative weights or a Gaussian distribution. The number of edges is given by $3|V| - 6$. B4 denotes the implementation of the presented algorithm using Blossom IV, B5 uses Blossom V instead, and SWK denotes the implementation of the method by Shih et al. using Blossom V.

stances for each choice of parameters (size and percentage of negative weights). Here and in the following numbers written in bold always indicate the variant with minimum CPU time. It can be seen that B5 is the fastest implementation. The running times of B5 are clearly faster than those of B4 and SWK for every parameter combination. We found that using the modified expansion step with K_3's for Shih et al. (introduced in Sec. 5) the running times are faster compared to the original SWK but still worse when compared to B5. For example, on average B5 needs 10.45 (57.49) seconds on maximum planar graphs of size 100,000 (500,000). In comparison, SWK(K_3) needs 14.97 (180.77) seconds. This can be explained by the triangulation step needed first of all. For general planar graphs this step is always necessary whereas it is not for maximum planar graphs. Maximum planar graphs are already triangulated. Therefore, one could run SWK without the triangulation step that is otherwise necessary. Without triangulation step, the running times of SWK(K_3) are smaller and comparable to B5, e.g. 6.83 (38.77) seconds for $|V| = 100{,}000$ (500,000).

We further applied the algorithm to a special class of planar graphs often studied in the physics application, namely two-dimensional grid graphs. Determining maximum cuts in grid graphs is relevant for the study of so-called Ising spin glasses. The results for grid graphs with edge weights following a bimodal $\pm J$ distribution are given in Table 2, again always averaged over 100 instances. This kind of graphs was also used as a test bed for MIN-CUTS by Kolmogorov [28] when he introduced his Blossom V implementation. He used the method by Shih et al. but with a modified expansion rule. He removed the two nodes v_1 and v_2, see Fig.9. This is similar to the mentioned variant of using K_3 instead of the star graph. He compared his results for two-dimensional Ising spin glasses with those obtained using Blossom IV. On a Pentium III machine with 1322 MHz, the minimum cut computation took on average 645 seconds for a 400×400 grid [28]. This has to be compared with 30 seconds that we need for 500^2 grids (B5), respectively 98 seconds with B4, on our more modern machines (whereas the original SWK needs 124 seconds, and in the K_3 variant 68 seconds).

Bieche et al. [10] proposed an algorithm solving MAX-CUT especially designed for grid graphs. This exact algorithm is well-known to physicists and widely accepted as a standard routine. A main drawback of the method is that a minimum perfect matching has to be computed in a complete graph. Hence, memory requirements increase strongly with increasing system size. Therefore, heuristic but high-quality variants of the approach are presented in the physics literature that reduce memory problems. Using these heuristics, sizes up to 480^2 [25] or 1801^2 grids [40] are reachable. Using the presented approach leads both to a faster and to a less memory-consuming approach. For a comparison between the exact approach and the heuristic variant just mentioned, see [41].

It is worth mentioning that the performance of the B4 version is comparable to that of B5 for small grid graphs. However, for increasing graph sizes the B5 implementation is considerably faster. As before, using the SWK version for solving MAX-CUT on grid graphs is slower than both versions. B5 is the fastest implementation for any grid graph size. For small graph sizes B4 is comparable

$\|V\|$	B4	B5	SWK
100^2	0.35	**0.31**	1.14
150^2	0.98	**0.76**	3.83
250^2	5.78	**2.55**	12.04
500^2	97.53	**30.04**	123.99
1000^2	1487.90	**543.48**	1268.56
2000^2	16586.95	**5900.25**	13984.45
3000^2	59051.47	**33146.23**	oom

Table 2. Grid graph instances. Average running times (in sec.) over 100 instances. Each instance with $\frac{|E|}{2}$ number of edges having weight < 0 (oom denotes "out of memory"). B4 denotes the implementation of the presented algorithm using Blossom IV, B5 the implementation of it using Blossom V, SWK denotes the implementation of the method by Shih et al. using Blossom V.

to SWK and does not run into memory issues when used on the largest sizes. On a machine with 16 GB the SWK algorithm could not solve any 3000^2 instance due to its high memory requirements. This is also true for the SWK version using K_3 subgraphs in the expansion steps. Using the SWK with K_3's the running times are comparable to B4 for grid graphs of size 100^2 (0.72 sec.), 150^2 (1.76 sec.), 250^2 (6.30 sec.), and 500^2 (68.29 sec.), and worse if compared to B5. For the largest sizes SWK in the K_3 version is faster than B4 (1000^2 (727.50 sec.), 2000^2 (9138.90 sec.)) but again B5 is the fastest implementation. We conclude that for both random planar and grid graphs method B5 yields the best performance, compared to the method of Shih et al. in its original and its modified version.

In order to study realistic instances, we took the TSPLIB library, maintained by Gerhard Reinelt [42]. As the realistic instances represent geographic points all edges have non-negative weights. Thus, we compute MAX-CUTs. First, for all geometric TSPLIB instances with at least $1,000$ nodes Delaunay triangulations (using the LEDA library [36]) are computed and the Euclidean distance between nodes is chosen as edge weights. Results are given in Table 3.

Again, B5 and SWK are fast and only need seconds to solve each instance. B5 again is the fastest implementation (it is on average at least a factor two faster than SWK). For smaller instances the differences between B4 and B5 decrease but do not vanish. Furthermore, using Kolmogorov's new Blossom V implementation leads to a drastic speedup when compared to Blossom IV. Whereas B4 almost needs 3 hours for the most difficult instance, B5 still can solve it within less than 10 seconds.

The Delaunay triangulated instances are almost maximum planar graphs (only the outer face may not be triangulated). The running time results given in Table 3 show that B5 is again the fastest implementation. Using SWK in the K_3 version on Delaunay triangulated graphs leads to shorter running times than of the original version but is still slower than the B5 method, e.g. for pla85900 SWK using K_3 needs 18.46 sec. which has to be compared to 8.31 sec. that are needed with B5.

instance name ($\|E\|$)	B4	B5	SWK
pla85900 (257604)	10248.70	**8.31**	27.72
pla33810 (101367)	390.50	**2.11**	6.11
usa13509 (40503)	169.73	**0.78**	2.21
brd14051 (42128)	140.49	**0.85**	2.31
d18512 (55510)	86.50	**1.48**	3.08
pla7397 (21865)	15.11	**0.49**	0.96
rl11849 (35532)	9.87	**0.83**	1.66
rl5934 (17770)	4.56	**0.33**	0.89
fnl4461 (13359)	3.21	**0.23**	0.58
rl5915 (17728)	2.84	**0.31**	0.70
pr2392 (7125), dsj1000 (2981), vm1748 (4784), rl1889 (5631)	< 2.00	$<$ **0.12**	< 0.20
rl1323 (3950), fl3795 (11326), u1060 (3153), rl1304 (3879), pcb3038 (9101), vm1084 (2869)	< 1.00	$<$ **0.19**	< 0.38
pr1002 (2972), u1432 (4204), d2103 (6290), u2319 (6869), u2152 (6312), d1291 (3845), u1817 (5386), d1655 (4890)	< 0.40	$<$ **0.10**	< 0.23
fl1577 (4643), nrw1379 (4115), fl1400 (4138), pcb1173 (3501)	< 0.20	$<$ **0.10**	< 0.14

Table 3. Realistic instances (tsplib). Running times (in sec.) for MAX-CUT computation on different realistic instances. The number of nodes is encoded in the instance name, and the number of edges is given explicitly. B4 denotes the implementation of the presented algorithm using Blossom IV, B5 the implementation of it using Blossom V, SWK denotes the implementation of the method by Shih et al. using Blossom V.

instance name ($\|V\|$, $\|E\|$)	B4	B5
USA-road-d.FLA (1,070,376 nodes, 2,712,798 edges)	394937.00	**166.91**
USA-road-d.NW (1,207,945 nodes, 2,840,208 edges)	168239.00	**61.15**
USA-road-d.NY (264,346 nodes, 733,846 edges)	117997.27	**11.38**
USA-road-d.BAY (321,270 nodes, 800,172 edges)	90486.00	**13.28**
USA-road-d.COL (435,666 nodes, 1,057,066 edges)	32227.10	**27.50**

Table 4. Realistic instances (DIMACS). Running times (in sec.) for MAX-CUT computation on road network instances. B4 denotes the implementation of the presented algorithm using Blossom IV, B5 the implementation of it using Blossom V.

We also studied road network maps of the USA, taken from the 9th DIMACS Implementation Challenge (Shortest Paths) [1]. From the library, we took all instances with up to $1,200,000$ nodes. The largest instance took around 4.5 days with B4 to compute. The running times are drastically reduced from days to seconds when using B5. Even the most difficult instance can be solved by B5 within a couple of minutes. As a summary, for both random and realistic planar instances the outlined method can determine optimum cuts for graphs with up to one million nodes within minutes.

9 Conclusion

We presented a MAX-CUT algorithm for arbitrary weighted planar graphs. The presented approach is nifty and simpler than the methods presented earlier. Moreover, it is easy to implement. Its worst-case asymptotic running time is the same as that of Shih et al. [45]. Furthermore, we showed in the computational experiments that it is very fast in practice and can compute optimum cuts in graphs with up to a million nodes within several minutes, at most using Kolmogorov's Blossom V implementation for the matching computations. Moreover, it is faster than the method proposed by Shih et al. and easier but not slower than that of Berman et al. An interesting question is to explore whether the usage of planar separator strategies for the matching part could further reduce the computation time and memory in practice.

Acknowledgments

We are grateful to Frank Baumann for verifying some of the computational results. We would like to thank Vladimir Kolmogorov for stimulating discussions. We are grateful to the referees for their valuable suggestions. In particular, we thank a referee who has suggested the algorithmic variant for the method of [45] outlined in the end of Section 5. Financial support from the German Science Foundation is acknowledged under contract Li 1675/1-1.

References

1. 9th DIMACS, *Implementation challenge - shortest paths*, http://www.dis.uniroma1.it/~challenge9/download.shtml, 2005.
2. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows*, Prentice Hall Inc., 1993, Theory, algorithms, and applications.
3. K. Aoshima and M. Iri, *Comments on Hadlock's Paper: "Finding a maximum cut of a planar graph in polynomial time"*, SIAM J. Comput. **6** (1977), no. 1, 86–87.
4. F. Barahona, *On the computational complexity of Ising spin glass models*, J. Physics A: Mathematical and General **15**, no. 10.
5. ———, *The max-cut problem on graphs not contractible to K_5*, Oper. Res. Lett. **2** (1983), no. 3, 107–111.
6. ———, *Planar multicommodity flows, max-cut, and the Chinese-Postman problem*, Polyhedral combinatorics (Morristown, NJ, 1989), DIMACS Ser. Discrete Math. Theoret. Comput. Sci., vol. 1, Amer. Math. Soc., Providence, RI, 1990, pp. 189–202.
7. F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt, *An application of combinatorial optimization to statistical physics and circuit layout design*, Operations Research **36** (1988), no. 3, 493–513.
8. F. Barahona, R. Maynard, R. Rammal, and J. P. Uhry, *Morphology of ground states of two-dimensional frustration model*, J. Physics A: Mathematical General **15** (1982), no. 2, 673–699.
9. P. Berman, A. B. Kahng, D. Vidhani, and A. Zelikovsky, *The T-join problem in sparse graphs: Applications to phase assignment problem in VLSI mask layout*, WADS (F. K. H. A. Dehne, A. Gupta, J.-R. Sack, and R. Tamassia, eds.), Lecture Notes in Computer Science, vol. 1663, Springer, 1999, pp. 25–36.
10. L. Bieche, J. P. Uhry, R. Maynard, and R. Rammal, *On the ground states of the frustration model of a spin glass by a matching method of graph theory*, J. Phys. A: Math. Gen. **13** (1980), no. 8, 2553–2576.
11. E. Boros and P. L. Hammer, *The max-cut problem and quadratic* 0-1 *optimization; polyhedral aspects, relaxations and bounds*, Ann. Oper. Res. **33** (1991), no. 1-4, 151–180, Topological network design (Copenhagen, 1989).
12. C. Buchheim, F. Liers, and M. Oswald, *A basic toolbox for constrained quadratic 0/1 optimization*, WEA (C. C. McGeoch, ed.), Lecture Notes in Computer Science, vol. 5038, Springer, 2008, pp. 249–262.
13. W. Cook and A. Rohe, *Computing minimum-weight perfect matchings*, INFORMS Journal on Computing **11** (1999), 138–148.
14. COPhy, *Combinatorial Optimization in Physics*, http://cophy.informatik.uni-koeln.de/, 2007.
15. C. de Simone, *The cut polytope and the boolean quadric polytope*, Discrete Mathematics **79** (1989), 71–75.
16. J. Edmonds, *Maximum matching and a polyhedron with 0-1 vertices.*, Journal of Research at the National Bureau of Standards **69B** (1965), 125–130.
17. ———, *Paths, trees, and flowers.*, Can. J. Math. **17** (1965), 449–467 (English).
18. L. R. Ford, Jr. and D. R. Fulkerson, *Maximal flow through a network*, Canad. J. Math. **8** (1956), 399–404.
19. H. N. Gabow, *An efficient implementation of Edmonds' algorithm for maximum matching on graphs*, J. ACM **23** (1976), no. 2, 221–234.
20. ———, *Data structures for weighted matching and nearest common ancestors with linking*, SODA '90: Proceedings of the first annual ACM-SIAM Symposium On Discrete Algorithms, SIAM, 1990, pp. 434–443.

21. M. Grötschel, M. Jünger, and G. Reinelt, *Via minimization with pin preassignments and layer preference*, Z. Angew. Math. Mech. **69** (1989), no. 11, 393–399.
22. F. Hadlock, *Finding a maximum cut of a planar graph in polynomial time*, SIAM J. Comput. **4** (1975), no. 3, 221–225.
23. F. Harary (ed.), *Graph theory and theoretical physics*, Academic Press, London, 1967.
24. A. K. Hartmann and H. Rieger, *Optimization Algorithms in Physics*, Wiley-VCH, 2002.
25. A. K. Hartmann and A. P. Young, *Lower critical dimension of Ising spin glasses*, Phys. Rev. B **64** (1989), no. 18, 180404.
26. P. W. Kasteleyn, *Dimer statistics and phase transitions*, Journal of Mathematical Physics **4** (1963), no. 2, 287–293.
27. D. E. Knuth, *The Stanford GraphBase: a platform for combinatorial computing*, ACM, New York, NY, USA, 1993.
28. V. Kolmogorov, *Blossom V: a new implementation of a minimum cost perfect matching algorithm*, Mathematical Programming Computation **1** (2009), no. 1, 43–67.
29. Y. S. Kuo, T. C. Chern, and W. K. Shih, *Fast algorithm for optimal layer assignment*, DAC '88: Proceedings of the 25th ACM/IEEE conference on design automation (Los Alamitos, CA, USA), IEEE Computer Society Press, 1988, pp. 554–559.
30. E. L. Lawler, *Combinatorial optimization: networks and matroids*, Holt, Rinehart and Winston, New York, 1976.
31. F. Liers, M. Jünger, G. Reinelt, and G. Rinaldi, *Computing exact ground states of hard Ising spin glass problems by Branch-and-Cut*, New Optimization Algorithms in Physics (A. K. Hartmann and H. Rieger, eds.), John Wiley & Sons, 2004, pp. 47–68.
32. F. Liers and G. Pardella, *A simple max-cut algorithm for planar graphs*, Tech. report, Combinatorial Optimization in Physics (COPhy), Sep. 2008, http://www.zaik.uni-koeln.de/~paper/preprints.html?show=zaik2008-579.
33. R. J. Lipton and R. E. Tarjan, *A separator theorem for planar graphs*, SIAM J. Appl. Math. **36** (1979), no. 2, 177–189.
34. ———, *Applications of a planar separator theorem*, SIAM J. Comput. **9** (1980), no. 3, 615–627.
35. S. T. McCormick, M. R. Rao, and G. Rinaldi, *Easy and difficult objective functions for max-cut*, Math. Program. **94** (2003), no. 2-3, Ser. B, 459–466, The Aussois 2000 Workshop in Combinatorial Optimization.
36. K. Mehlhorn and S. Näher, *LEDA: A Platform for Combinatorial and Geometric Computing*, Cambridge University Press, 1999.
37. P. Mutzel, *Implementierung und Analyse eines Max-Cut Algorithmus für planare Graphen (diploma thesis)*, 1990.
38. OGDF, *Open Graph Drawing Framework*, http://www.ogdf.net, 2007.
39. G. I. Orlova and Ya. G. Dorfman, *Finding the maximum cut in a graph*, Engineering Cybernetics **10** (1972), 502–506.
40. R. G. Palmer and J. Adler, *Ground states for large samples of two-dimensional Ising spin glasses*, International Journal of Modern Physics C **10** (1999), 667–675.
41. G. Pardella and F. Liers, *Exact ground states of large two-dimensional planar Ising spin glasses*, Physical Review E (Statistical, Nonlinear, and Soft Matter Physics) **78** (2008), no. 5, 056705.
42. G. Reinelt, *TSPLIB – A Traveling Salesman Problem Library*, ORSA Journal on Computing **3** (1991), 376–384.

43. N. Schraudolph and D. Kamenetsky, *Efficient exact inference in planar Ising models*, Tech. report, Australian National University and NICTA, Oct. 2008, http://arxiv.org/abs/0810.4401.
44. SCIL, http://scil-opt.net.
45. W. K. Shih, S. Wu, and Y. S. Kuo, *Unifying maximum cut and minimum cut of a planar graph*, IEEE Trans. Comput. **39** (1990), no. 5, 694–697.
46. M. Stoer and F. Wagner, *A simple min-cut algorithm*, J. ACM **44** (1997), no. 4, 585–591.
47. C. K. Thomas and A. A. Middleton, *Matching Kasteleyn cities for spin glass ground states*, Physical Review B **76** (2007), no. 22, 220406(R).