A constructive and improving heuristic algorithm for the maximum weight planar subgraph problem

Rafael C. Silva, Paulo A. G. Kataki, Les R. Foulds, Humberto J. Longo
Instituto de Informática
Universidade Federal de Goiás
Alameda Palmeiras, Quadra D, Campus Samambaia
CEP 74001-970, Goiânia, Goiás, Brasil
{rafaelcastrosilva,pauloagkataki,lesfoulds,longo}@inf.ufg.br

RESUMO

Este trabalho aborda o problema de se identificar um subgrafo planar, de peso máximo, de um dado grafo G ponderado nas arestas, o qual é importante na modelagem e resolução de problemas das mais diversas áreas. É proposto um novo algoritmo heurístico de busca local, baseado em métodos previamente existentes de construção de soluções e de melhoria das mesmas, para este problema pertencente à classe $\mathcal{NP}$-difícil de problemas de otimização. Quando avaliado com instâncias numéricas geradas sinteticamente, os resultados obtidos com o algoritmo proposto superaram os melhores até então disponíveis, obtidos com a combinação de outros dois métodos heurísticos anteriormente propostos para o problema.

PALAVRAS CHAVE. Heurística, Peso máximo, Planar, Subgrafo.

TAG – Teoria e Algoritmos em Grafos, MH – Metaheurísticas, OC – Otimização Combinatória.

ABSTRACT

Algorithms for the problem of identifying a maximum edge-weight planar subgraph of a given edge-weighted graph G are relevant in a wide variety of application areas. We propose a new local search constructive and improving heuristic algorithm for this $\mathcal{NP}$-hard problem that is based on existing methods for the problem. When implemented on synthetically generated numerical instances, the algorithm outperformed the combination of two earlier, well-established heuristic methods for the problem, both in terms of solution quality and asymptotic time complexity.

KEYWORDS. Heuristic, Maximum-weight, Planar, Subgraph.

TAG – Theory and Algorithms in Graphs, MH – Meta-heuristics, OC – Combinatorial Optimisation.

1. Introduction

The Maximum-weight Planar Subgraph Problem (*MWPSP*) involves searching a finite, simple, undirected, edge-weighted graph $G = (V(G), E(G), w)$ (of order n, with vertex set $V(G)$, edge set $E(G)$ and nonnegative edge weights $w(e), e \in E(G)$, where $w{:}E(G) \rightarrow \mathbb{R}^+ \cup \{0\}$) for a planar, spanning subgraph $G' = (V(G), E(G'))$ with $E(G') \subseteq E(G)$, having the highest sum of

https://proceedings.science/p/157518?lang=pt-br

edge weights $w(G') = \sum_{e \in E(G')} w(e)$ among all such subgraphs of G. An instance of the MWPSP is denoted by *MWPSP*(G, w) and if G' is one of its solutions then $w(G')$ is termed the *value* of G'.

Several existing heuristic algorithms (henceforth termed just *heuristics*) for MWPSP, like the *PMFG* proposed by Tumminello et al. [2005] and the *TMFG* proposed by Massara et al. [2017], are based on an iterative process of applying simple operations involving vertices, edges or faces, such as vertex-by-vertex construction, edge substitution or the relocation of vertices from one face of a plane subgraph to another. These basic operations are commonly referred to as *local* or *topological moves* and transform a given maximal planar graph into a new one. Many heuristics for *MWPSP*(G, w) commence with a tetrahedron (a complete subgraph of G having four vertices) as initial input. In this case, the tetrahedron is termed a *seed*. The TMFG, for instance, constructs a feasible solution to any instance *MWPSP*(G, w) by starting from some seed and progressively increments it by the application of one or more topological moves. At each iteration, a new vertex and the edges incident with it are inserted into the current partial solution. The decision as to which new vertex and its incident edges are chosen to be inserted at each iteration is made according to the highest increase in the objective function among all possible topological moves.

The *All Seeds* constructive heuristic, proposed by Coelho et al. [2020], uses the *Face Dimpling* and *Edge Dimpling* moves (see Section 2) as separate subroutines in the process of constructing feasible solutions for an instance *MWPSP*(G, w). The heuristic explores all possible seeds as input and separately applies the two moves in the process of expanding all seeds. At each iteration, a new vertex is inserted into the current subgraph, according to the highest local gain that a topological move can produce. The best result obtained among all those from the tested seeds is then returned.

The *Restricted Seeds* constructive heuristic, proposed here and denoted by $\mathcal{RS}$, works in a similar way, but considers only a restricted subset of all possible starting tetrahedra. It also uses the same topological moves used by the All Seeds method (face dimpling and edge dimpling) to develop a solution for a MWPSP instance. However, new heuristic parameters were defined to restrict the number of seeds and the choice of which topological moves to perform. In this way, the computational complexity of the method is better than that of the All Seeds heuristic, but without significant loss in the final quality of the best solution found.

The *Cubic Heuristic* algorithm (denoted here by $\mathcal{CH}$) proposed by Kataki et al. [2020], is a local search improvement algorithm based on the concepts presented in Sections 2 and 4. The idea behind this algorithm is that a maximal planar graph can be modified by selecting a particular 6-cycle and replacing the chords in that cycle by other chords. The algorithm always preserves the planarity property while traversing the neighbourhood of the current solution, that is, whenever one or more edges are replaced a planar graph is generated. This fact eliminates the need to use a planarity test. This is encouraging as, even considering its linear time complexity, planarity testing is computationally expensive when it has to be executed in every iteration of the algorithm.

The *Restricted Seeds with Cubic Heuristic* (denoted here by $\mathcal{RS} + \mathcal{CH}$), that is proposed in the present paper as a new algorithm, combines concepts used in the $\mathcal{RS}$ and $\mathcal{CH}$ heuristic algorithms to construct and improve solutions to an MWPSP instance. It works with a restricted set of seeds, which are sequentially incremented using simple topological operations, as in the Restricted Seeds heuristic. In addition, calls are periodically made to a subroutine that implements the Cubic Heuristic. In this way, as new vertices and edges are added to a current partial solution, an attempt is made to improve the solution as much as possible.

The rest of this paper is organised as follows. The next section briefly presents two known constructive procedures that build up a planar spanning subgraph step-by-step, and then discusses

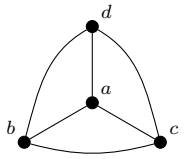
https://proceedings.science/p/157518?lang=pt-br

some improvement moves. Section 3 recalls the *Star Move* that, along with the Cubic Heuristic explained in Section 4, can be used to attempt to make improvements to the dual of the graph under study. These procedures are utilised to develop the proposed primal-dual heuristic detailed in Section 5. Computational experience with the new approach is reported in Section 6 and a summary rounds out the paper in the last section.

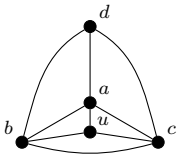
2. Some topological moves

The *Face Dimpling* (FD) constructive move consists of inserting a new vertex within a (triangular) face of a given maximal planar graph and connecting it to the three vertices of the face. This operation was originally proposed by Foulds and Robinson [1978] as a subroutine of two heuristic constructive algorithms (the author's R and S methods), which are nowadays commonly termed just the *Deltahedron Heuristic*. This move is also denoted as *Vertex Insertion* by Leung [1992]; Merker and Wäscher [1997] and as the T_2 move by Massara et al. [2017]. Let $G = (V(G), E(G))$ be a maximal planar graph, such that vertex $u \notin V(G)$ and $\langle a, b, c\rangle$ defines a face of G. This operations creates a new maximal planar graph $G' = (V(G'), E(G'))$, with $V(G') = V(G) \cup \{u\}$ and $E(G') = E(G) \cup \{ua, uc, ub\}$, as illustrated in Figures 1a and 1b.

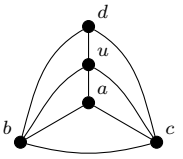
A further constructive move is the *Edge Dimpling* (ED) move. It was first proposed by Alexander [Alexander, 1930] and was termed the A move by Massara et al. [2017] in Alexander's honour. Given a maximal planar graph $G = (V(G), E(G))$, the move firstly removes an edge $ad \in E(G)$, which is common to faces abd and acd of G. It then inserts a new vertex $u \notin V(G)$ in the interior of the face $abcd$ resulting from the deletion of edge ad. Finally, it connects u to the vertices a, b, c, d. Therefore, the original faces abd, acd no longer exist and new faces bud, aub, auc and cud come into being. This move is illustrated in Figures 1a and 1c.



(a) Graph G before the topological moves.



(b) Face Dimpling move, with vertex u being inserted in face $\langle a, b, c\rangle$ of G.



(c) Edge Dimpling move, with vertex u being inserted within edge $\{a, d\}$ of G.

Figure 1: The Face Dimpling (1a $\rightarrow$ 1b) and the Edge Dimpling (1a $\rightarrow$ 1c) moves.

Pesch et al. [1999] interpreted some topological improvement moves as the act of rotating a particular substructure of vertices and edges within a graph. Following this idea, Kataki et al. [2020] extended the concept and proposed some new improvement moves. These basic operations can be interpreted as the rotation of a chosen subgraph within a given cycle. A subset of these new moves extract a specific subgraph that is enclosed within a 6-cycle and then inserts it again in the cycle in a different way, yielding a distinct maximal planar graph.

As an example of the concepts discussed in the previous paragraph, let t be any combination of three edges as chords enclosed within a vertex-labelled 6-cycle $C_6 = \langle a, d, b, e, c, f\rangle$ of G. Although there are 84 possible combinations of three edges as chords enclosed within a vertex-labelled 6-cycle, only the 14 cases shown in Figure 2 define distinct enclosed planar subgraphs.

Thus, with the new rotational topological moves proposed by Kataki et al. [2020], a new maximal planar graph can be obtained by taking the structure formed by t and rotating it clockwise or counterclockwise within C_6. Depending on the initial chord structure assumed, the new rotational improvement move is termed either *face-*, *3-path-* or *claw*-rotation (cases 2a $\leftrightarrow$ 2b), 2c $\leftrightarrow$ 2d $\leftrightarrow$ 2e or 2f $\leftrightarrow$ 2g $\leftrightarrow$ 2h, and 2i $\leftrightarrow \cdots \leftrightarrow$ 2n, respectively).

https://proceedings.science/p/157518?lang=pt-br

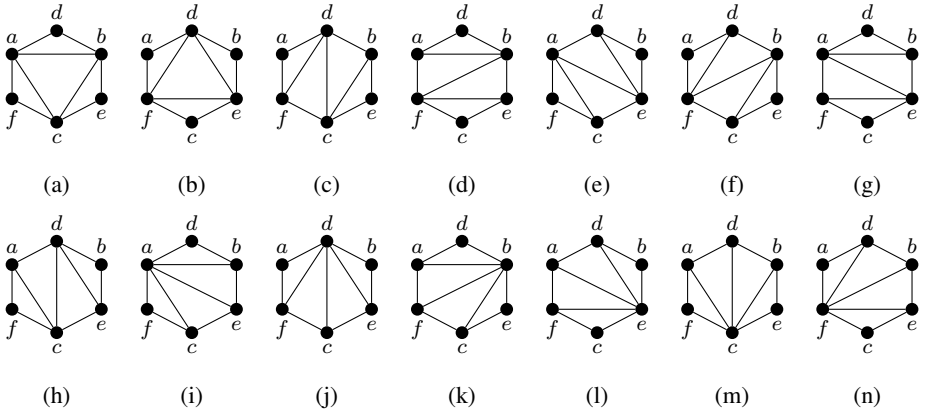


Figure 2: The planar triangulations of the vertex-labelled 6-cycle.

Furthermore, starting with any of the cases in Figure 2 and removing its combination t, of three chords from C_6 and inserting some other planar combination t' say, of three chords, always leads to one of the other cases in Figure 2. Thus, the search for a subgraph that increases the total weight of a solution to an *MWPSP*(G, w) instance can be limited to only those that can be included in 6 cycles. However, managing all 6 cycles in G and its embedded subgraphs can become an arbitrarily difficult task as the cardinality of $V(G)$ grows. Moreover, transforming a structure formed by three chords embedded in a particular 6-cycle into some other valid structure embedded in the same 6-cycle, can also entail a large number of simple topological moves before an improved feasible solution is identified.

Fortunately, recognising 6-cycles and replacing structures included in them become simple tasks, of $\mathcal{O}(1)$ time complexity, when these operations are performed on the dual graph of G. This possibility was explored by Kataki et al. [2020] who developed a topological move denoted by *Star Move*, which is described in Section 3 below.

3. The star move on a cubic graph

Let $H = (V(H), E(H))$ be a connected, cubic, planar graph. Kataki et al. [2020] defined a *Star Gadget* to be a subgraph of H that is induced by a vertex $i \in V(H)$ and three of its neighbours $j, k, \ell \in V(H)$ (i.e. a star graph, with hub i of degree 3). The authors denoted such a subgraph as $i\star jk\ell$. The *Star Move* was defined as the operation of replacing a star gadget $i\star jk\ell$ in H by another star gadget $i'\star j'k'\ell'$ (eventually $j' = j$, $k' = k$ or $\ell' = \ell$). The graph generated from H by a star move is called *valid* if it is also a connected, cubic, planar graph and *invalid* if it does not meet such requirements. A star move is said to be valid if it generates a valid graph.

When H is the dual graph of a connected maximal planar graph $G = (V(G), E(G), H$ is connected, cubic and planar [Ore, 1967]. A star move operation on graph H is equivalent to the replacement in G of at most four triangles t_1, t_2, t_3 and t_4 say, included within a 6-cycle C_6 say, by other triangles included within the same cycle C_6. In general, p edges of a triangle t in G are affected by a star move in the dual graph H, where $1 \leqslant p \leqslant 3$. The new planar subgraphs that can be obtained by this process of replacing triangles within C_6 are exactly the same as those that can be achieved by replacing one to three chords of C_6, as explained in the previous section. Vertices in H corresponding to triangles outside the C_6 in G, as well as these faces, are not affected by the star move operation. However, there can be 2, 3 or 4 faces included in C_6 that are affected.

An example of the notions discussed in the previous paragraph is illustrated in Figure 3, where primal graphs G, with vertices a, b, c, d, e and f are given in black and their duals H, are

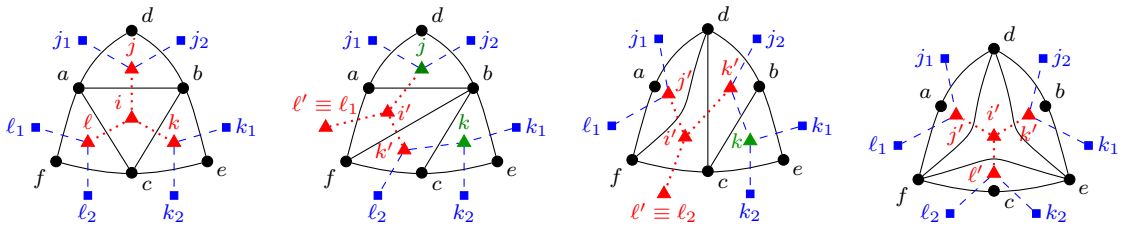
https://proceedings.science/p/157518?lang=pt-br

given in red and blue. Consider the star gadget $i\star jk\ell$ in H and the triangles $t_1 = \Delta_{abc}$, $t_2 = \Delta_{abd}$, $t_3 = \Delta_{acf}$ and $t_4 = \Delta_{bce}$ included within the 6-cycle $\langle a, d, b, e, c, f, a\rangle$ in G, as illustrated in Figure 3a. After a star move with the star gadget $i\star jk\ell$ in graph H, the result in graph G is the replacement of two to four of the faces t_1 to t_4 by new faces, depending on the choice of the new star gadget to be inserted in H, as described below:

Figure 3b: replacement of faces t_1 and t_3 with the new faces $t'_1 = \Delta_{abf}$ and $t'_3 = \Delta_{bcf}$ (new star gadget $i'\star jk'\ell_1$);

Figure 3c: replacement of faces t_1, t_2 and t_3 with the new faces $t'_1 = \Delta_{adf}$, $t'_2 = \Delta_{bcd}$ and $t'_3 = \Delta_{cdf}$ (new star gadget $i'\star j'k'\ell_2$);

Figure 3d: replacement of faces t_1 to t_4 with the new faces $t'_1 = \Delta_{adf}$, $t'_2 = \Delta_{bde}$, $t'_3 = \Delta_{cef}$ and $t'_4 = \Delta_{def}$ (new star gadget $i'\star j'k'\ell'$).



(a) (b) (c) (d)

Figure 3: Illustrations of the equivalence between triangle replacement in a primal maximal planar graph and the application of a star move in its dual.

4. The cubic heuristic

The Cubic Heuristic ($\mathcal{CH}$), using the concepts described in the previous section, works on a cubic graph H', the dual of a spanning planar subgraph G' of a graph G, part of an instance *MWPSP*(G, w). However, some other method is needed to extract from G, in a previous processing phase, such a subgraph G'. The $\mathcal{CH}$ heuristic is based on the fact that it is usually easier to identify a star gadget in H' than to identify a particular structure embedded within a 6-cycle in G'. The $\mathcal{CH}$ heuristic sequentially identifies star gadgets in H' and performs the appropriate star moves to obtain a new cubic graph H''. At the end of this process, the final maximal planar graph G'', corresponding to H'', is constructed.

The $\mathcal{CH}$ heuristic comprises two main parts: the first one controls the iterative processing of G' and the second part performs all necessary tasks regarding the dual graph H'. The pseudo-code of the first part is given as Algorithm 1. The second part, subroutine *getKNeighbours*(H', k), is the core of the $\mathcal{CH}$ heuristic, being used to construct new cubic graphs from H' (that are neighbours of H') by applying star moves. It identifies all neighbours of the current graph H' that can be generated by a single star move and returns the k dual graphs (for a given k) with the highest sum of edge weights to Algorithm 1, which decides whether any of the returned dual graphs is the best found so far. A detailed description of the operations performed by each subroutine in both parts of the $\mathcal{CH}$ heuristic, as well as its time complexity (of $\mathcal{O}(n^2)$) was provided by Kataki et al. [2020].

A high-level description of the general functioning of the heuristic is given next. Algorithm 1 receives as input a maximal planar subgraph G', builds its dual (cubic) graph H' and passes it to the subroutine *getKNeighbours*(H', k) (the main part of the $\mathcal{CH}$ heuristic), that generates the k heaviest neighbours of H' and returns them to Algorithm 1. The overall heaviest dual graph found

https://proceedings.science/p/157518?lang=pt-br

so far is recorded at each iteration and progressively updated. When a pre-specified number of iterations of Algorithm 1 have been completed, the final heaviest dual graph is converted into its primal maximal planar graph, which is returned as the final output.

Algorithm 1: CubicHeuristic $(G', k, nbIterations)$

Input: Maximal planar edge-weighted graph G'; number k of neighbours and number of iterations $nbIterations$.
Output: Maximal planar edge-weighted graph G''.

```
1  H ← buildCubicGraph( G' );
2  bestGraph ← H;
3  bagGraphs ← ∅;
4  for iteration ← 1 to nbIterations do
5      neighbours ← getKNeighbours( H, k );
6      bestNeighbour ← getBestNeighbour( neighbours, bagGraphs );
7      addNeighboursToBag( bagGraphs, neighbours );
8      if isValid(bestNeighbor) then
9          H ← bestNeighbour;
10     else
11         H ← getBestGraph( bagGraphs );
12         delete( bagGraphs, H );
13     visit( H );
14     if (getCost( H ) > getCost( bestGraph )) then
15         bestGraph ← H;
16 G'' ← buildMaximalPlanar(bestGraph);
17 return G''.
```

5. The proposed approach

In this section a new constructive heuristic for the MWPSP is proposed, which is the result of combining the Cubic Heuristic improvement algorithm described in Section 4 with another new MWPSP heuristic constructive algorithm, the Restricted Seeds algorithm, that is briefly described in Section 5.1 below. The Restricted Seeds algorithm is a variation of the All Seeds algorithm [Coelho et al., 2020]. The proposed algorithm, termed here the Return to Sender algorithm, is described in Section 5.2.

5.1. The Restricted Seeds algorithm

The *All Seeds* heuristic algorithm ($\mathcal{AS}$) [Coelho et al., 2020] uses the face dimpling and edge dimpling topological moves, described in Section 2, to greedily build a solution to an instance *MWPSP*(G, w), starting from every possible seed (all tetrahedra of G). After all seeds have been processed, only the best solution identified is returned by the heuristic. Since the number of seeds is $\mathcal{O}(n^4)$ and each is used to build a solution in $\mathcal{O}(n^2)$ topological moves, the overall time complexity of the method is $\mathcal{O}(n^6)$. As the number of seeds is often very high, it can take a relatively long time to process them all. However, it can be assumed that only a subset of the seeds actually leads to high quality solutions. Thus, the new *Restricted Seeds* ($\mathcal{RS}$) heuristic works similarly to the $\mathcal{AS}$ heuristic, but it does not process the entire set of seeds, only a strict subset of them.

The choice of this subset is the key to a good performance of the Restricted Seeds heuristic, regarding processing time and, more importantly, the quality of returned solutions. Therefore,

https://proceedings.science/p/157518?lang=pt-br

an efficient way to obtain subsets with the smallest possible number of seeds that can potentially lead to high quality solutions is proposed. An intuitive way of doing this is to consider the weight of each seed considered (the sum of its edge weights), since it is common sense that seeds with relatively high weights are usually likely to lead to good quality solutions.

The proposed method comprises a pre-processing phase concerning the complete set of all $\binom{n}{4}$ possible seeds that can be extracted from G. This phase sorts all these seeds in non-increasing order according to their weight, with a time complexity $\mathcal{O}(n^4 \cdot \log(n))$. Since this sorting phase must be performed only once before many runs of the restricted seeds heuristic, it does not affect the complexity of the main part of the heuristic, which has time complexity $\mathcal{O}((x+z)n^2)$. Thus, the complexity of the heuristic is $\mathcal{O}(n^6)$ when $x + z = \Theta(n^4)$, given the parameters x and z described below. However, the computational experiments reported in Section 6, show that $x + z = \Theta(1)$ is enough to achieve improved solutions.

Let $\mathcal{C}$ denote the subset of filtered (finally chosen) seeds. Only the first heaviest y seeds in the ordered set of seeds are considered for the filtering process. Initially, the first z (where $z \leqslant y$) heaviest seeds are added to $\mathcal{C}$. Next, x more seeds ($x + z \leqslant y$) are randomly chosen among the $(y - z)$ remaining seeds and are added to $\mathcal{C}$. Then, the *Face-Edge Dimpling* algorithm (*FED*) in Coelho et al. [2020] is applied to each seed in $\mathcal{C}$. The reason for randomly choosing some additional seeds is to improve the chances of getting out of eventual local optima that could be reached if only the heaviest z seeds were used.

The Restricted Seeds heuristic is detailed in Algorithm 2. It receives as input a graph G and the parameters $x, y, z \in \mathbb{N}$, as explained before. The set $\mathcal{C}$ of filtered seeds is constructed in Step 1. This set will be tested to construct a solution. The loop between Steps 11 and 21 is the kernel of the solution construction process. It starts with a specific seed taken at Step 4 and successively applies the face dimpling and edge dimpling moves to transform it into a solution for the given MWPSP instance. This processing is done greedily, always applying moves that locally augment the current solution by adding a new vertex into an edge or into a face, as explained in Section 2.

The block of steps enclosed in Step 3 of Algorithm 2 runs $|\mathcal{C}| = x + z$ times. From Step 4 to Step 9 (that are of $\mathcal{O}(1)$ time complexity), the sets $\mathcal{E}$ of edges and F of faces are extracted from the chosen seed. The set S of vertices (not yet belonging to the solution) and the heaps $\mathcal{H}_e$ and $\mathcal{H}_f$ are also constructed. These heaps store the face and edge dimpling moves that can be performed on the current solution, respectively. The loop in Step 11 is executed $(n - 4)$ times. Step 12 takes $\mathcal{O}(\log n)$ time. Both steps in the conditions between 12 and 15 are performed in $\mathcal{O}(1)$ time, and this is where the face and edge moves occur, adding a new vertex, and hence creating new edges and faces. Step 16 can be done in $\mathcal{O}(\log n)$ time. Steps 17 and 18 are the most expensive in this loop, and are where the new possibilities for face and edge moves on the heaps arise. For that purpose, it is necessary to consider the junction of each new face and edge added in the topological move just performed with each vertex not yet present in the solution. The number of faces and edges added at each step, is of order $\mathcal{O}(1)$, and the number of vertices that are not yet added is represented by $|S|$. As $|S| = \mathcal{O}(n)$, Steps 17 and 18 have complexity $O(n \cdot \log n)$ and, as shown by Coelho et al. [2020], this can be improved to $\mathcal{O}(n)$. Steps 20 and 21 can be done in $\mathcal{O}(\log n)$. The final step in this loop, just updates the current weight. This weight represents the edge weight sum of the current solution, and can be calculated using the added and removed edges, and thus, takes $\mathcal{O}(1)$ time. Hence, the final complexity of this internal loop is $O(n^2)$. As the external loop in Step 3 runs $(x + z)$ times, the overall complexity is $\mathcal{O}((x + z)n^2)$.

https://proceedings.science/p/157518?lang=pt-br

Algorithm 2: RestrictedSeeds(G, x, y, z)

Input: Graph $G = (V, E)$ and parameters $x, y, z \in \mathbb{N}$ that reduce the search space of viable solutions

Output: Maximal planar subgraph G', filtered from G.

```
/* Seed weight is the choice criterion adopted, according
   to the heuristic strategy described in Section 5.1.   */
```

$\mathcal{C} \leftarrow FilteredSeeds(G, x, y, z)$;
$MaxWeight \leftarrow 0$;
foreach $k \leftarrow 1$ **to** $|\mathcal{C}|$ **do**
  $\mathcal{C}_k \leftarrow \{v_1^k, v_2^k, v_3^k, v_4^k\} \in \mathcal{C}$;
  $\mathcal{S} \leftarrow V - \{\mathcal{C}_k\}$;
  $\mathcal{E} \leftarrow \{\{v_1^k, v_2^k\}, \{v_1^k, v_3^k\}, \{v_1^k, v_4^k\}, \{v_2^k, v_3^k\}, \{v_2^k, v_4^k\}, \{v_3^k, v_4^k\}\}$;
  $F \leftarrow \{\{v_1^k, v_2^k, v_3^k\}, \{v_1^k, v_2^k, v_4^k\}, \{v_1^k, v_3^k, v_4^k\}, \{v_2^k, v_3^k, v_4^k\}\}$;
  $\mathcal{H}_f \leftarrow \{$maximum gain with $\{s, f\}$, $\forall\, f \in F$ e $s \in \mathcal{S}\}$;
  $\mathcal{H}_e \leftarrow \{$maximum gain with$\{s, e\}$, $\forall\, e \in E$ e $s \in \mathcal{S}\}$;
  $TmpMax \leftarrow \sum_{e \in \mathcal{E}} w(e)$;
  while $(\mathcal{S} \neq \varnothing)$ **do**
    /* A new vertex is inserted into the current subgraph, according to the highest local gain that a topological move can produce. */
    best_pair $\leftarrow \max\{\max\{\mathcal{H}_f\}, \max\{\mathcal{H}_e\}\}$; **if** $(\textit{best_pair} = \max\{\mathcal{H}_f\})$ **then**
      Call *Face Dimpling* routine;
    else
      Call *Edge Dimpling* routine;
    Remove the added vertex from $\mathcal{S}$;
    $\mathcal{H}_f \leftarrow \mathcal{H}_f \cup \{$Gain of s and each added face, $\forall\, s \in \mathcal{S}\}$;
    $\mathcal{H}_e \leftarrow \mathcal{H}_e \cup \{$Gain of s and each added edge, $\forall\, s \in \mathcal{S}\}$;
    Remove *best_pair* from its respective heap;
    Update F accordingly;
    Update $\mathcal{E}$ accordingly;
  if $(TmpMax \geqslant MaxWeight)$ **then**
    $MaxWeight \leftarrow TmpMax$;
    $\mathcal{M}' \leftarrow \mathcal{E}$;
return $G' = (V, \mathcal{M}')$.

5.2. Restricted Seeds with Cubic Heuristic

The *Restricted Seeds with Cubic Heuristic* algorithm ($\mathcal{RS} + \mathcal{CH}$) combines the $\mathcal{RS}$ heuristic with the $\mathcal{CH}$ heuristic, seeking to take advantage of the best features of both heuristics. On the one hand, it uses face and edge dimpling moves and on the other hand, it uses the fact that $\mathcal{CH}$ is an improvement algorithm. Thus, dimpling routines are used in the same fashion as they are used in the $\mathcal{RS}$, but with periodic calls to a subroutine that implements $\mathcal{CH}$.

The first step of $\mathcal{RS} + \mathcal{CH}$ is to choose what seed to apply the face and edge dimpling moves to, along with calls to $\mathcal{CH}$. The chosen seed is the one that leads to the best solution returned by the Restricted Seeds heuristic. The procedure that will be applied to this specific seed, is explained in more detail in the pseudocode of Algorithm 3. It receives as input: an edge-weighted

graph G, a seed C and parameters $p, nbIter \in \mathbb{N}$ that distribute the calls to $\mathcal{CH}$ and limit the number of its iterations at each call, respectively, as will be further explained later. Steps 1 to 17 are virtually the same as in the internal loop of Algorithm 2, i.e. Steps 5 to 21. The main difference between Algorithms 2 and 3 is in Step 18 of the latter, where the calls to $\mathcal{CH}$ occur. As explained in Section 4, this tends to improve the current solution without adding any vertices, but changes only the edges of the current graph, while still preserving planarity.

Algorithm 3: ReturnToSender($G, C, p, nbIter$)

Input: Graph $G = (V, E)$, seed C, and parameters $p, nbIter \in \mathbb{N}$.
Output: Maximal planar subgraph G', filtered from G.

1 $\mathcal{S} \leftarrow V - \{C\}$;
2 $\mathcal{E} \leftarrow \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_3\}, \{v_2, v_4\}, \{v_3, v_4\}\}$;
3 $F \leftarrow \{\{v_1, v_2, v_3\}, \{v_1, v_2, v_4\}, \{v_1, v_3, v_4\}, \{v_2, v_3, v_4\}\}$;
4 $\mathcal{H}_f \leftarrow$ {maximum gain with $\{s, f\}$, $\forall\, f \in F$ e $s \in \mathcal{S}$};
5 $\mathcal{H}_e \leftarrow$ {maximum gain with$\{s, e\}$, $\forall\, e \in E$ e $s \in \mathcal{S}$};
6 **while** $(\mathcal{S} \neq \varnothing)$ **do**
7 *best_pair* $\leftarrow \max\{\max\{\mathcal{H}_f\}, \max\{\mathcal{H}_e\}\}$;
8 **if** (*best_pair* $= \max\{\mathcal{H}_f\}$) **then**
9 Call Face Dimpling routine;
10 **else**
11 Call Edge Dimpling routine;
12 Remove the added vertex from $\mathcal{S}$;
13 $\mathcal{H}_f \leftarrow \mathcal{H}_f \cup$ {Gain of s and each added face, $\forall\, s \in \mathcal{S}$};
14 $\mathcal{H}_e \leftarrow \mathcal{H}_e \cup$ {Gain of s and each added edge, $\forall\, s \in \mathcal{S}$};
15 Remove *best_pair* from its respective heap;
16 Update F accordingly;
17 Update $\mathcal{E}$ accordingly;
18 **if** $|V \setminus S|$ *is divisible by* p **then**
19 $CubicHeuristic(\langle V \setminus S, \mathcal{E}\rangle, k = 50, nbIter)$
20 **return** $G' = (V, \mathcal{E})$.

As the other steps of this algorithm are the same as those previously explained, we shall focus only on this added step to explain how it works. The parameter p is added to this algorithm, as mentioned before, to distribute the calls for $\mathcal{CH}$. This is done by checking the size of the current solution in terms of its number of vertices. This is done in Step 18, where the term $(|V \setminus S|)$ is calculated as the number of such vertices, with S being defined as the set of vertices not yet in the solution. The effect of this is that $\mathcal{CH}$ will be called every p iterations. That is, the first call will occur when the current solution has p vertices, the second when it has $2p$ vertices, the third with $3p$, and so on. When $\mathcal{CH}$ is called, the current maximal planar graph is $\langle V \setminus S, \mathcal{E}\rangle$, where the current vertex set is $V \setminus S$, and the edge set is $\mathcal{E}$. This graph, along with the two other parameters k and $nbIterations$, are passed as input to $\mathcal{CH}$. For the present purpose, the value of the $\mathcal{CH}$ parameter k was fixed at 50, as this value led to good results. The complexity of $\mathcal{RS}$ depends on the number of times that $\mathcal{CH}$ is called, which is n/p, with each call taking $\mathcal{O}(n^2)$ time, as explained in Section 4. This implies that the overall time complexity of $\mathcal{RS}$ is $\mathcal{O}((n/p)n^2) = \mathcal{O}(n^3/p)$.

6. Computational experience

The experimental work was carried out on a computer running Ubuntu 18.04 (64 bits), with an Intel i5 2.80 GHz processor and 16GB of ECC RAM. The code was compiled with GCC

https://proceedings.science/p/157518?lang=pt-br

7.5.0, with the code targeting the native architecture and the O3 optimisation flag set. The computer was dedicated to running tests with the chosen data instances.

To evaluate the performance of the $\mathcal{RS} + \mathcal{CH}$ heuristic, the solutions found are compared with the solutions generated by the combination of the $\mathcal{AS}$ and the $\mathcal{CH}$ heuristics, which currently seems to be among the best performing heuristic methods for the MWPSP. The experiments used synthetically generated instances (*MWPSP*(K_n, w), $n = 20, 25, \ldots, 100$), originally proposed by Coelho et al. [2020], with weights $w(e), e \in E(K_n)$ randomly generated in the range $[0, 199]$.

	Upper	$\mathcal{AS} + \mathcal{CH}$		$\mathcal{RS} + \mathcal{CH}$						
n	Bound	Solution	Gap (%)	Solution	Gap (%)	y	z	x	p	$nbIter$
20	**8434**	8434	0.000	8434	0.000	4845	95	95	20	5000
25	**11594**	11594	0.000	11594	0.000	7590	80	160	5	2000
30	**13770**	13744	0.189	13761	0.065	16446	80	160	2	2000
35	**17177**	17128	0.285	17138	0.227	52360	95	95	1	5000
40	**19862**	19605	1.294	19685	0.891	31416	80	160	20	2500
45	22384	21755	2.810	21822	2.511	148995	95	95	1	5000
50	25763	25162	2.333	25263	1.941	230300	95	95	4	5000
55	28536	27937	2.099	27964	2.004	341055	95	95	1	5000
60	31272	30576	2.226	30579	2.216	487635	95	95	1	5000
65	34226	33190	3.027	33321	2.644	677040	95	95	1	5000
70	37493	36569	2.464	36718	2.067	341348	100	100	1	3200
75	40130	39022	2.761	39267	2.151	1215450	130	95	3	5000
80	43058	41900	2.689	42022	2.406	948948	160	320	2	3200
85	45701	44407	2.831	44493	2.643	24161	100	200	2	4000
90	48828	47313	3.103	47533	2.652	1533114	100	200	5	4000
95	51626	50019	3.113	50295	2.578	1910130	100	200	5	4000
100	54875	53388	2.710	53748	2.054	3921225	95	95	1	5000
Average gaps (%)			**1.996**		**1.709**					

Table 1: A summary of the experimental results.

Table 1 presents some of the results obtained from the experiments. The values in the second column are upper bounds on the values of the optimal solutions, obtained by rounding down the values of the solutions calculated by using an exact algorithm for the MWPSP and some bounding techniques proposed by Ahmadi-Javid et al. [2015]. The numbers in bold represent the values of optimal solutions found by the exact algorithm proposed by Kataki [2021]. The columns labelled $\mathcal{AS} + \mathcal{CH}$ display the values of the best solutions to the instances returned by the Cubic heuristic, when it was run with the best solution found by the All Seeds heuristic as the initial solution (initial planar subgraph G') of the search process, and their *gap* percentages of the upper bounds, obtained by Kataki et al. [2020]. The next two columns are relative to the solutions obtained by the proposed $\mathcal{RS} + \mathcal{CH}$ heuristic.

The $\mathcal{RS} + \mathcal{CH}$ heuristic was run with each instance for different parameter value combinations (the last five columns of Table 1), which were mainly set as follows: y was chosen in the range of 30% to 100% of the total number of seeds (except that the instance $n = 85$ was tested with just 1.2% of the total number of seeds); z was chosen in the range [60,160], x was chosen in the range [40,320], p was in the range either [1,10] or [20,40] and $nbIter$ was chosen in the range

[2000, 5000]. Most experiments focused on testing variations of the $\mathcal{RS} + \mathcal{CH}$ heuristic parameters p and $nbIter$. It can be observed that even considering a small amount of seeds in relation to total possible (with $z + x$ decreasing from less than 2% in the instance $n = 25$ to less than 0.008% in the instance $n = 100$), the heuristic managed to achieve good results, with small variations in the parameter p. The gaps between the upper bounds on the optimal solutions and the solutions constructed by the $\mathcal{RS} + \mathcal{CH}$ heuristic are all smaller than the gaps previously reached by the combination of the $\mathcal{AS}$ and $\mathcal{CH}$ heuristics, with the overall average gap being reduced from 1.996% to 1.709%.

The performance of the $\mathcal{RS} + \mathcal{CH}$ heuristic is highly dependent on its parameters x, y and z. In order for it to work as efficiently as possible, it is necessary to find the best the values of each parameter. The tuning of parameters for meta-heuristics has already been considered by many authors and several methods have been proposed for this (for instance, Huang et al. [2020] provides a survey of automatic parameter tuning methods for meta-heuristics). However, so far in the present research, such methods have not yet been considered. Thus, the proposed algorithm was executed on each instance for various combinations of parameter values, and the results shown in Table 1 are the best obtained, using the values shown in the columns labelled x, y, z and p.

Method	**Solution value**	**Gap (%)**
Upper-Bound	121.835	—
$\mathcal{RS} + \mathcal{CH}$	117.072	3.909
FED $+ \mathcal{CH}$	116.964	3.998
FED	116.897	4.053
FD $+ \mathcal{CH}$	116.623	4.278
FD	116.553	4.335
TMFG	116.224	4.605
PMFG	115.568	5.144

Table 2: Solution values for the 100-vertex instance [Tumminello et al., 2005].

A second experiment was performed on a 100-vertex instance proposed by Tumminello et al. [2005], which is based on an empirical study of practical financial data derived from the US stock market, with vertices and edges representing stocks and given correlation coefficients between them, respectively. Table 2 shows the solution values to the instance that were returned by the application of the heuristics mentioned earlier. The top row of the table shows an upper bound on the optimal solution value of the instance, obtained by Kataki [2021] using the bounding techniques proposed by Ahmadi-Javid et al. [2015]; the second row shows the value of the solution obtained with the proposed $\mathcal{RS} + \mathcal{CH}$ heuristic; and the third to the eighth rows display the results obtained with the methods: "*FED* $+ \mathcal{CH}$" and "*FD* $+ \mathcal{CH}$", due to Kataki et al. [2020] (where the plus sign means "$\mathcal{CH}$ starts with output from ..."); *FED* and *FD*, due to Coelho et al. [2020]; *TMFG*, due to Massara et al. [2017]; and *PMFG*, due to Tumminello et al. [2005]. The solution values produced by the listed methods are shown in decreasing value, with $\mathcal{RS} + \mathcal{CH}$ being the best performing method and *PMFG* the worst performing in terms of solution quality.

7. Summary

A new polynomial-time constructive and improvement heuristic is proposed for the maximum edge-weight planar subgraph problem. The algorithm combines the features of two other heuristics for the problem, (namely the new Restricted Seeds heuristic and the existing and well-established Cubic Heuristic), seeking to take advantage of the best features of both heuristics. The

https://proceedings.science/p/157518?lang=pt-br

proposed algorithm was applied to several synthetically created numerical instances and to an instance of 100 vertices from the literature. The solutions obtained from the synthetically created instances are on average within 1.80% of optimality and at most 2.88% in the worst case. The best solution found so far for the literature instance has been improved, reducing its optimality gap to less than 3.91%. For the instances solved, in terms of solution quality, the proposed algorithm outperformed the combination of the All Seeds and Cubic heuristics for the problem, with the Cubic heuristic being executed with the best solution found by the All Seeds heuristic as the initial solution of the search process. It was also shown that the proposed algorithm has asymptotic time complexity lower than the total complexity of the All Seeds and Cubic heuristics combination that is, $\mathcal{O}(n^3/p)$ of the former compared to $\mathcal{O}(n^6) + \mathcal{O}(n^2) = \mathcal{O}(n^6)$ of the latter.

References

Ahmadi-Javid, A., Ardestani-Jaafari, A., Foulds, L. R., Hojabri, H., and Farahani, R. Z. (2015). An algorithm and upper bounds for the weighted maximal planar graph problem. *Journal of the Operational Research Society*, 66(8):1399–1412.

Alexander, J. W. (1930). The combinatorial theory of complexes. *Annals of Mathematics*, 31: 292–320.

Coelho, V. S., Foulds, L. R., Longo, H. J., and Martins, W. S. (2020). Parallelised heuristics for the maximum-weight planar subgraph problem. Unpublished paper.

Foulds, L. R. and Robinson, D. F. (1978). Graph theoretic heuristics for the plant layout problem. *The International Journal of Production Research*, 16(1):27–37.

Huang, C., Li, Y., and Yao, X. (2020). A Survey of Automatic Parameter Tuning Methods for Metaheuristics. *IEEE Transactions on Evolutionary Computation*, 24(2):201–216.

Kataki, P. A. G. (2021). Novos algoritmos heurístico e exato para o MWPSP. Master's thesis, Instituto de Informática, Universidade Federal de Goiás, Goiânia-GO, Brazil.

Kataki, P. A. G., Cappelle, M. R., Foulds, L. R., and Longo, H. J. (2020). A new algorithm for the Maximum-weight Planar Subgraph Problem. In *Anais do LII Simpósio Brasileiro de Pesquisa Operacional*, p. 1–12, João Pessoa, PB, Brazil.

Leung, J. (1992). A New Graph-Theoretic Heuristic for Facility Layout. *Management Science*, 38 (4):594–605.

Massara, G. P., Di Matteo, T., and Aste, T. (2017). Network Filtering for Big Data: Triangulated Maximally Filtered Graph. *Journal of Complex Networks*, 5(2):161–178.

Merker, J. and Wäscher, G. (1997). Two new heuristic algorithms for the maximal planar layout problem. *OR Spektrum*, 19(2):131–137.

Ore, Ø. (1967). *The four-color problem*. Pure and applied mathematics. Academic Press.

Pesch, E., Glover, F., Bartsch, T., Salewski, F., and Osman, I. (1999). Efficient facility layout planning in a maximally planar graph model. *International Journal of Production Research*, 37 (2):263–283.

Tumminello, M., Aste, T., Di Matteo, T., and Mantegna, R. N. (2005). A tool for filtering information in complex systems. *Proceedings of the National Academy of Sciences*, 102(30):10421–10426.

https://proceedings.science/p/157518?lang=pt-br