Journal of XXX manuscript No.
(will be inserted by the editor)

Multistart Search for the Cyclic Cutwidth Minimization Problem

Sergio Cavero · Eduardo G. Pardo ·
Abraham Duarte · Manuel Laguna

Received: date / Accepted: date

Abstract The Cyclic Cutwidth Minimization Problem (CCMP) consists of finding an embedding of the vertices of a candidate graph in a host graph in order to minimize the maximum cut of a host edge. The host graph is restricted to be a cycle. We study several properties of the CCMP and design a multistart search for this problem. We compare our procedure with the state of the art for the CCMP using sets of problem instances previously published. Statistical tests indicate the merit of our proposal.

Keywords Cyclic Cutwidth, Tabu Search, Circular Layouts, Graph embedding

1 Introduction

Some families of optimization problems can be represented as graph layout problems where the objective consists of defining a mapping of a candidate graph into a regular structure, called the host graph. In these problems, there are two mapping functions. The first one assigns each vertex in the candidate graph (candidate vertex) to a vertex in the host graph (host vertex). The second function assigns to each edge in the candidate graph (candidate edge) a path in the host graph (host path). The most common approaches found in the literature for this family of problems are those where the host graph is a line. These problems are commonly referred to as linear layout problems [34]. However, there exist mappings over more complex regular structures such as trees, grids, or cycles [8]. Regardless of the structure of the candidate and host graphs, the optimization problem may be defined over several objective function choices.

We tackle a minimization problem consisting of embedding a general candidate graph in a cycle host graph. Let $\mathcal{C} = (\mathcal{V}_\mathcal{C}, \mathcal{E}_\mathcal{C})$ be a connected, unweighted, and undirected candidate graph where $\mathcal{V}_\mathcal{C}$ and $\mathcal{E}_\mathcal{C}$ represent the sets of vertices and

Sergio Cavero, Universidad Rey Juan Carlos, E-mail: sergio.cavero@urjc.es · Eduardo G. Pardo, Universidad Rey Juan Carlos, E-mail: eduardo.pardo@urjc.es · Abraham Duarte, Universidad Rey Juan Carlos, E-mail: abraham.duarte@urjc.es · Manuel Laguna, University of Colorado Boulder, E-mail: laguna@colorado.edu

edges, respectively. Analogously, let $\mathcal{H} = (\mathcal{V}_{\mathcal{H}}, \mathcal{E}_{\mathcal{H}})$ be a host cycle graph with the following properties:

- $n = |\mathcal{V}_{\mathcal{C}}| = |\mathcal{V}_{\mathcal{H}}| = |\mathcal{E}_{\mathcal{H}}|$
- The degree of each vertex $v \in \mathcal{V}_{\mathcal{H}}$ is 2
- $\mathcal{H}$ is a Eulerian and Hamiltonian graph
- The disposition of the vertices $\mathcal{V}_{\mathcal{H}}$ in the Euclidean space is such that all adjacent vertices are placed at the same distance.

Considering these definitions, the host graph in our context is represented as a cycle. A bijective function φ assigns each vertex in the candidate graph to a single vertex in the host graph. This function is defined as $\varphi : \mathcal{V}_{\mathcal{C}} \rightarrow \mathcal{V}_{\mathcal{H}}$, where $\forall\, v \in \mathcal{V}_{\mathcal{C}}\ \exists\ w \in \mathcal{V}_{\mathcal{H}}$ such that $\varphi(v) = w$. An injective function ψ assigns candidate edges to host paths. A path is a sequence of edges that connects two vertices without repeating any edges or vertices. Let $\mathcal{P}_{\mathcal{H}}$ be the set of all possible host paths in $\mathcal{H}$. Then, for every candidate edge $(u, v) \in \mathcal{E}_{\mathcal{C}}$ there are two possible paths in $\mathcal{P}_{\mathcal{H}}$, one ending in $\varphi(u)$ and another ending in $\varphi(v)$. The ψ function is defined as the mapping of candidate edges to host paths, i.e., $\psi : \mathcal{E}_{\mathcal{C}} \rightarrow \mathcal{P}_{\mathcal{H}}$.

Figure 1 shows an example of a candidate graph ($\mathcal{C}$), a host graph ($\mathcal{H}$), a possible embedding of the candidate graph in the host graph, and the two possible host paths between a pair of adjacent vertices (A and D). In particular, Figure 1(a) shows a candidate graph with $\mathcal{V}_{\mathcal{C}} = \{\mathrm{A, B, C, D, E, F}\}$ and $\mathcal{E}_{\mathcal{C}} = \{(\mathrm{A, B}),\ (\mathrm{A, D}),\ (\mathrm{A, E}),\ (\mathrm{A, F}),\ (\mathrm{B, C}),\ (\mathrm{B, D}),\ (\mathrm{C, D}),\ (\mathrm{D, E})\}$. Therefore, the host graph must have the same number of vertices $\mathcal{V}_{\mathcal{H}'} = \{1, 2, 3, 4, 5, 6\}$ and a set of corresponding edges $\mathcal{E}_{\mathcal{H}'} = \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1)\}$, as presented in Figure 1(b). One possible mapping of the candidate graph into the host graph is showed in Figure 1(c). Each vertex in $\mathcal{V}_{\mathcal{C}}$ is assigned to a vertex in $\mathcal{V}_{\mathcal{H}}$. For instance, vertex A in $\mathcal{V}_{\mathcal{C}}$ is assigned to vertex 1 in $\mathcal{V}_{\mathcal{H}}$. The assignment is denoted by $\varphi(\mathrm{A}) = 1$. Similarly, vertex B is assigned to vertex 4 ($\varphi(\mathrm{B}) = 4$), and so forth for all other candidate vertices. Then, candidate edges are assigned to host paths. For instance, edge $(\mathrm{A, D})$ must be assigned to a host path with $\varphi(\mathrm{A}) = 1$ and $\varphi(\mathrm{D}) = 5$ as terminal vertices. The possible paths are $p_{(\mathrm{A,D})_1} = \{\varphi(\mathrm{A}), \varphi(\mathrm{E}), \varphi(\mathrm{D})\} = \{1, 6, 5\}$ and $p_{(\mathrm{A,D})_2} = \{\varphi(\mathrm{A}), \varphi(F), \varphi(\mathrm{C}), \varphi(\mathrm{B}), \varphi(\mathrm{D})\} = \{1, 2, 3, 4, 5\}$. These paths are shown in Figure 1(d). Note that for any candidate edge, there are only two possible host paths (clockwise and counterclockwise) when the host graph is a cycle. We further define ψ as a function that selects the shortest path. The length of the path is determined by the number of host edges traversed. In our example, the length of $p_{(\mathrm{A,D})_1}$ is 2 and the length of $p_{(\mathrm{A,D})_2}$ is 4. Therefore, $\psi(\mathrm{A, D}) = p_{(\mathrm{A,D})_1} = \{1, 6, 5\}$. The function must be applied to all candidate edges $\mathcal{E}_{\mathcal{C}}$. When the length of both host paths is the same, ψ selects the clockwise path.

1.1 Problem description

We study the Cyclic Cutwidth Minimization Problem (CCMP) for general candidate graphs and cycle host graphs. Our objective function for the CCMP is based on the concept of a cut of an edge in the host graph (i.e., edges in $\mathcal{E}_{\mathcal{H}}$). Given the assignments solution φ and ψ, the cut of an edge $e \in \mathcal{E}_{\mathcal{H}}$ is defined as the number of host paths assigned by ψ that traverse e. This calculation has been referred to in the literature as congestion [40].

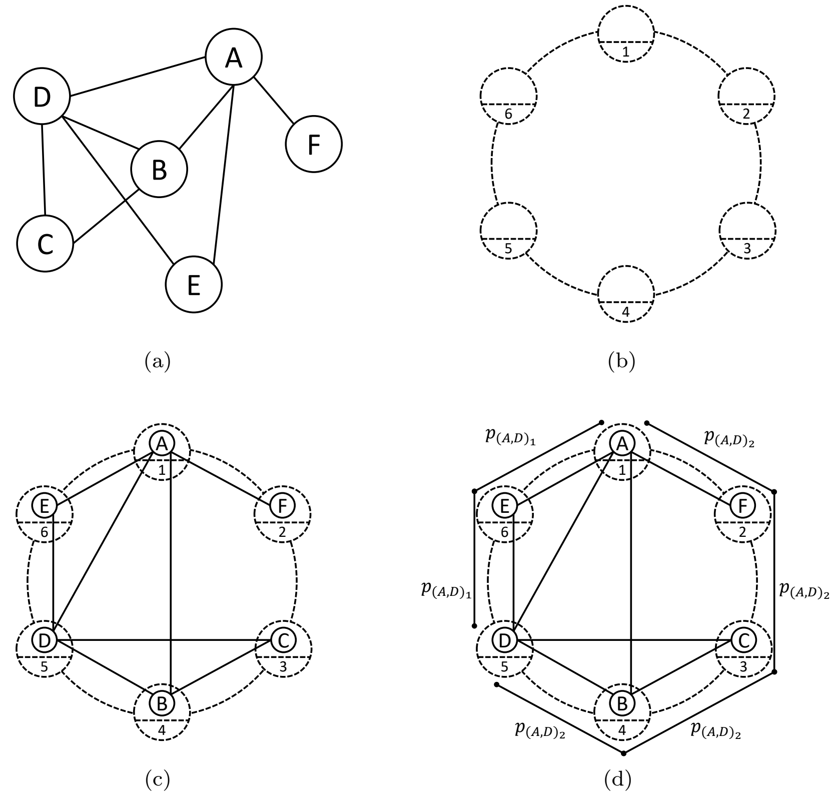


Fig. 1: (a) A candidate graph, $\mathcal{C}$. (b) A host graph, $\mathcal{H}$. (c) An embedding of $\mathcal{C}$ in $\mathcal{H}$. (d) Two possible paths for $(A, D) \in V_{\mathcal{C}}$ named as $p_{(A,D)_1}$ and $p_{(A,D)_2}$ respectively.

Formally, we define the cut of host edge $(w, z) \in \mathcal{E}_{\mathcal{H}}$ associated with φ and ψ as:

$$\underset{(w,z)\in\mathcal{E}_{\mathcal{H}}}{\mathrm{cut}((w, z), \varphi, \psi)} = |\{(u, v) \in \mathcal{E}_{\mathcal{C}} : (w, z) \in \psi(u, v)\}| \tag{1}$$

where the $\psi(u, v)$ path is defined as:

$$\psi(u, v) = \begin{cases} \{w, z\} & \text{if } \varphi(u) = w \wedge \varphi(v) = z \\ \{w, z, \ldots, \varphi(v)\} & \text{if } \varphi(u) = w \\ \{\varphi(u), \ldots, w, z\} & \text{if } \varphi(v) = z \\ \{\varphi(u), \ldots, w, z, \ldots, \varphi(v)\} & \text{otherwise} \end{cases} \tag{2}$$

The objective function (ccw) is denoted as the cyclic cutwidth and is calculated as follows:

$$ccw(\mathcal{C}, \varphi, \psi) = \max_{(w,z)\in\mathcal{E}_{\mathcal{H}}} \mathrm{cut}((w, z), \varphi, \psi) \tag{3}$$

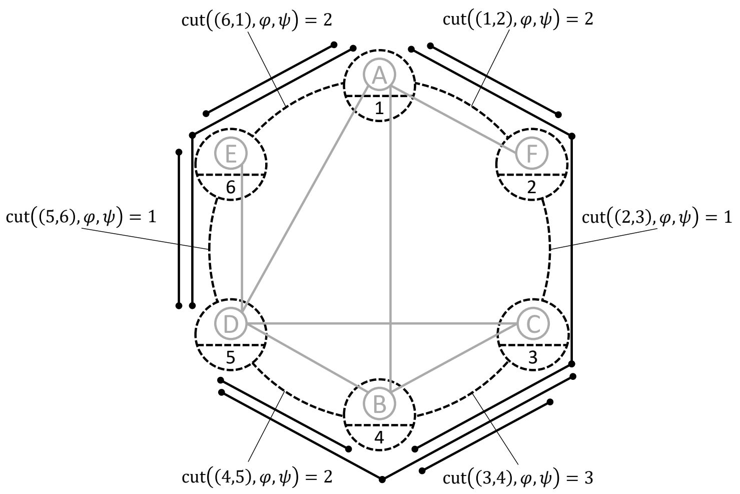


Fig. 2: Evaluation of a possible embedding of $\mathcal{C}$ in $\mathcal{H}$.

Since paths can be derived from the vertex assignments (see equation 2), a solution is fully characterized by φ. Therefore, for the purpose of the optimization problem, we can simplify the notation for ccw, by making it depend only on $\mathcal{C}$ and φ. Our minmax problem consists of finding, among all possible candidate vertex assignments $\varphi \in \Phi$, the assignment $\varphi^\star$ that minimize the cycle cutwidth:

$$\varphi^\star \leftarrow \arg\min_{\varphi \in \Phi} ccw(\mathcal{C}, \varphi) \tag{4}$$

Figure 2 depicts the evaluation of the solution represented in Figure 1, i.e., the evaluation of the assignments φ and ψ. The host graph is shown in dashed black lines and the candidate graph in gray solid lines. The host paths assigned by ψ to each candidate edge in $\mathcal{E}_\mathcal{C}$ are shown outside the cycle. Then, the cut associated with each edge of the host graph is indicated as the number of paths that traverse the edge. For example, edge $(1,2)$ is traversed by paths $p_{(\mathrm{A,F})}$ and $p_{(\mathrm{A,B})}$. Therefore, $\mathrm{cut}((1,2), \varphi', \psi') = |(\mathrm{A,F}), (\mathrm{A,B})| = 2$. Similarly, edge $(2,3)$ is traversed by only path $p_{(\mathrm{A,B})}$. Therefore, $\mathrm{cut}((2,3), \varphi, \psi) = |(\mathrm{A,B})| = 1$. The value of the objective function for the vertex assignment φ on graph $\mathcal{C}$ is $ccw(\mathcal{C}, \varphi) = \max\{2, 1, 3, 2, 2, 2\} = 3$.

1.2 Literature review

The CCMP is closely related to the Cutwidth Minimization Problem (CMP). Both problems share the same objective function, however the host graph for the CMP is a line, while the host graph for the CCMP is a cycle. The practical applications of these problems are also common, and can be found in various areas that include circuit design [7], engineering [27], and graph drawing [44].

The CMP was originally proposed in the 70s as a theoretical model in the context of circuit design [7]. The problem belongs to the $\mathcal{NP}$-hard class [12]. Due to the problem complexity, several heuristic procedures have been proposed in the literature [7,9,28,33,35,38]. The literature also includes exact procedures for the CMP on some special classes of graphs [20,40,45]. In addition, two Branch & Bound procedures have been proposed for general graphs [29,32] and mathematical formulations have been proposed in [25,26].

The bounds proposed in [23] establish a relationship of the objective function values for the CMP and the CCMP. In particular, for a candidate graph $\mathcal{C}$, if we let $lcw(\mathcal{C})$ and $ccw(\mathcal{C})$ be the optimal objective function values corresponding to the CMP and CCMP, then:

$$\frac{lcw(\mathcal{C})}{2} \leq ccw(\mathcal{C}) \leq lcw(\mathcal{C}) \tag{5}$$

The CMP and the CCMP are equivalent when $\mathcal{C}$ is a tree [5].

There are several exact algorithms for the CCMP for particular types of candidate graphs. The optimal value of the CCMP for complete graphs is known by construction [39]. For mesh graphs, it is possible to determine the optimum for grids with dimensions larger than 3×3 [6,41,42]. The optimal solution is known for three-dimensional meshes, as long as one dimension is 2 and the other two are greater than or equal to 2 [43]. Similarly, the optimal solution is know for cylindrical meshes for which one of the dimensions is greater than or equal to 2 and the other dimension is greater than or eqaul to 3 [41]. Exhaustive search procedures can be used to find optimal solutions for the CCMP on Q_3 hypercubes [1]. This result has been extended to Q_4 [22], Q_5 [3] and Q_6 [4] hypercubes. The CCMP has also been studied on general hypercubes [11] . Finally, exact algorithms exist for complete bipartite graphs [23], complete tripartite graphs [2], and n-partite graphs [2].

No exact algorithms exist for the CCMP on general graphs. Recently, the practical interest of the CCMP has motivated researchers in the optimization community to apply heuristic techniques to this problem. For instance [21] describes a Memetic Algorithm [31] for the CCMP. The authors propose six constructive heuristics to generate an initial population of solutions for their solution method. The method includes a local search procedure that attempts to move vertices from positions where the maximum cut occurs. The Memetic Algorithm was evaluated with six types of graph instances: complete splits, join of hypercubes, cones, toroidal meshes, and two random types. The procedure matched all known-optimal solutions and produced the best-known solutions for all other instances, establishing itself as the state-of-the-art for the CCMP.

1.3 Our contributions

The main contribution of this work is the development of a metaheuristic procedure that includes sound fundamental as well as advanced search strategies for the CCMP. We are able to show, through extensive computational experimentation, that our proposal is competitive with the current state-of-the-art for solving the CCMP. The method consists of a multistart search, where the starting points are generated in greedy fashion and the improvement phase is based on neighborhoods and a tabu memory structure [17]. We use a set of preliminary tests to find the best configuration of our procedure, i.e., to determine the best set of search parameter values. Through this process, we are able to determine the contribution of the various elements embedded in the proposed procedure.

After a set of tuning experiments to identify the best combination of parameter values, we employ the resulting procedure configuration for competitive testing. The test set consists of problem instances form the literature. Those instances are grouped into two main categories, graphs with known structure and random graphs. The results of these tests indicate that, in many cases, our proposed solution method is able to find solutions of better quality than the state of the art procedures, and in considerably less computational time. Furthermore, we show that these differences are statistically significant.

The description of our work is organized as follows. Section 2 describes our algorithmic proposals. Section 3 presents several advanced search strategies. Section 4 introduces the test set, describes the computational experiments, and discusses the results. Conclusions and final thoughts are in Section 5.

2 Algorithmic proposal

Our solution method has two main components, a procedure to construct solution and an improvement method based on tabu search (TS) [17]. We first provide details of a procedure to construct initial solution (Section 2.1). Then, we discuss how a local search operates on these initial solutions (see Section 2.2). This section ends with a description of the TS mechanisms that help the local search escape local optimal points.

2.1 Constructive procedure

Constructing a solution for the CCMP consists of performing two tasks: 1) assigning the vertices of the candidate graph to the vertices of the host graph (i.e., defining the domain and range of the φ function); and 2) assigning the edges of the candidate graph to a path in the host graph (i.e., defining the domain and range of the ψ function).

Initially, all candidate vertices are unassigned. We number the host vertices starting from the top vertex (i.e., the vertex that in a graphical representation would be at 12 o'clock) and continuing clockwise. At each step, we select a candidate vertex and assign it to the next available host vertex. Since the host graph is a cycle, without loss of generality, at each step of the procedure, we move sequentially in the clockwise direction. That is, the first candidate vertex is assigned to host vertex 1, the second candidate vertex is assigned to host vertex 2, and so forth. The first assignment is random. That is, a candidate vertex is randomly selected and it is assigned to host vertex 1. After the first assignment, all unassigned vertices in the candidate graph are evaluated with a greedy function to determine the most attractive vertex to assign next. The greedy function to select the next unassigned vertex from the candidate graph is inspired in previously published ideas [30]. The construction ends after all candidate vertices have been assigned to a host vertex.

The greedy selection function to select the next vertex from the candidate graph is defined as follows. Let A be the set of candidate vertices that have already been assigned and let U be the set of unassigned candidate vertices. Let $d(v)$ be the degree of an unassigned candidate vertex v ($v \in \mathrm{U}$) and let . Also, let $d_{\mathrm{A}}(v)$ be the number of assigned candidate vertices that are adjacent to v and $d_{\mathrm{U}}(v)$ be the number of unassigned vertices adjacent to v. That is,

$$d_{\mathrm{A}}(v) = |\{u \in \mathrm{A} : (v, u) \in \mathcal{E}_{\mathcal{C}}\}|$$

$$d_{\mathrm{NA}}(v) = |\{u \in \mathrm{U} : (v, u) \in \mathcal{E}_{\mathcal{C}}\}|.$$

Then, we define the greedy value g for an unassigned vertex v as:

$$g(v) = d_{\mathrm{A}}(v) - d_{\mathrm{U}}(v).$$

The g function measures the proximity of the candidate vertex under consideration to the assigned candidate vertices versus to its proximity to the unassigned candidate vertices. We would like to select the unassigned candidate vertex that, relative to other unassigned candidate vertices, is closest to the candidate vertices that have already been assigned. Therefore, the unassigned candidate vertex with the largest g value is chosen to be assigned next. The greedy function is such that it makes an unassigned candidate vertex very attractive if all vertices adjacent to it have already been assigned. Conversely, an unassigned candidate vertex is unattractive when none of its adjacent vertices has been assigned.

Algorithm 1 summarizes the greedy constructive procedure. The candidate graph $\mathcal{C}(\mathcal{V}_{\mathcal{C}}, \mathcal{E}_{\mathcal{C}})$ is the input to this procedure. Initially, all candidate vertices are unassigned (step 2). Steps 3 to 4 make the assignment of a randomly selected candidate vertex to host vertex 1. The assigned vertex is removed from the set of unassigned candidate vertices (step 5). A for-loop is then executed (steps 6 to 10) to assign all remaining candidate vertices in U. Step 7 evaluates all unassigned candidate vertices to identify the one with the largest greedy value (with ties broken arbitrarily). The chosen candidate vertex $next_{\mathrm{C}}$ is assigned to the next available host vertex $next_{\mathrm{H}}$ (step 8). The assigned candidate vertex, $next_{\mathrm{C}}$, is removed from U (step 9). Once all candidate vertices are assigned, the procedure returns φ, which contains the mapping of candidate vertices to host vertices 11.

Algorithm 1 Greedy construction

```
1: **Procedure** GreedyConstructive($\mathcal{C}(\mathcal{V}_{\mathcal{C}}, \mathcal{E}_{\mathcal{C}})$)
2: $\mathrm{U} \leftarrow \mathcal{V}_{\mathcal{C}}$
3: $next_{\mathrm{C}} \leftarrow$ rand(U)
4: $\varphi(next_{\mathrm{C}}) \leftarrow 1$
5: $\mathrm{U} \leftarrow \mathrm{U} \setminus \{next_{\mathrm{C}}\}$
6: **for all** $next_{\mathrm{H}} > 1$ **do**
7:     $next_{\mathrm{C}} \leftarrow \arg\max_{v \in U} g(v)$
8:     $\varphi(next_{\mathrm{C}}) \leftarrow next_{\mathrm{H}}$
9:     $\mathrm{U} \leftarrow \mathrm{U} \setminus \{next_{\mathrm{C}}\}$
10: **end for**
11: **return** $\varphi$
```

Figure 3 shows an example of the steps followed by the constructive procedure for the graph introduced in Figure 1(a). For this graph, the construction is completed in six iterations (i.e., one for each vertex). As shown in Figure 3(a), the procedure starts with the random selection of candidate vertex C, which is assigned to host vertex 1. In each step, we indicate which vertex is $next_{\mathrm{H}}$ and $next_{\mathrm{C}}$. The first one follows the numerical (clockwise) order. The second one is selected by computing the value of the g function. Figure 3(a) shows the g value for the vertices in U at this step (i.e., A,B,D,E, and F). For instance, $g(\mathrm{A}) = d_A(\mathrm{A}) - d_{\mathrm{NA}}(\mathrm{A}) = 0 - 4 = -4$. Similarly, $g(\mathrm{B}) = d_A(\mathrm{B}) - d_{NA}(\mathrm{B}) = 1 - 2 = -1$, and so forth. Once all the unassigned vertices have been evaluated, the greedy selection chooses the vertex with the largest g value (with ties broken arbitrarily). The number of vertices evaluated decreases by one at each step. In addition to the graphical representation of the current partial solution, Figure 3 includes tables associated with the vertex assignments, i.e, $\varphi(\mathrm{C}) = 1$, $\varphi(\mathrm{B}) = 2$, $\varphi(\mathrm{D}) = 3$, $\varphi(\mathrm{E}) = 4$, $\varphi(\mathrm{A}) = 5$, and $\varphi(\mathrm{F}) = 6$.

The ψ function can be derived from the φ mapping (see Figure 3(f)). The domain of ψ is given by the edges of the candidate graph, while the range is given by the paths in the host graph. Of the possible paths, we choose the shortest that connects the end candidate vertices. The topology of the host graph restricts the range of ψ to two possible paths per candidate edge, one clockwise and another one counterclockwise (see Figure 1(d)). If there is a tie in the length of the two possible paths, the procedure selects the clockwise path. Since ψ can be derived from the φ mapping, we consider that φ is a full characterization of the solution to the problem.

2.2 Local search

From a starting solution, a local search is an intensification strategy designed to find the local optimum in a predefined neighborhood. Our Local Search (LS) is defined in an insertion neighborhood. An insertion is a classical move in both graph layout and permutation problems. It consists of removing a candidate vertex from its current position in the host graph and inserting it in a different position. For instance, Figure 4 depicts the move of vertex A from position 5 (i.e., $\varphi(\mathrm{A}) = 5$) to position 2. We denote this operation as $\varphi' = Insert(\varphi, \mathrm{A}, 2)$, where φ' is the solution after the move. Figure 4 shows the solution before the insertion (φ) and the solution after the insertion (φ'). The figure highlights the vertices from the candidate and host graphs that are affected by the move. As customary in insertion moves, the displaced elements must be shifted. In our context, the vertices can be shifted in the clockwise or counterclockwise direction. Since the host graph is a cycle, shifting in one direction or the other results in the same solution. In our example, when candidate vertex A is moved to position 2, displaced vertices could have shifted in the clockwise direction (i.e., B moves to 3, D to 4, and E to 5). Instead, our moves are such that we always shift the displaced candidate vertices in the counterclockwise direction, as shown in Figure 4.

Considering the aforementioned insertion moves, the neighborhood associated with candidate vertex v of solution φ is defined as the solutions that can be reached by the insertions of v in all positions in the host graph that are different from its current position:

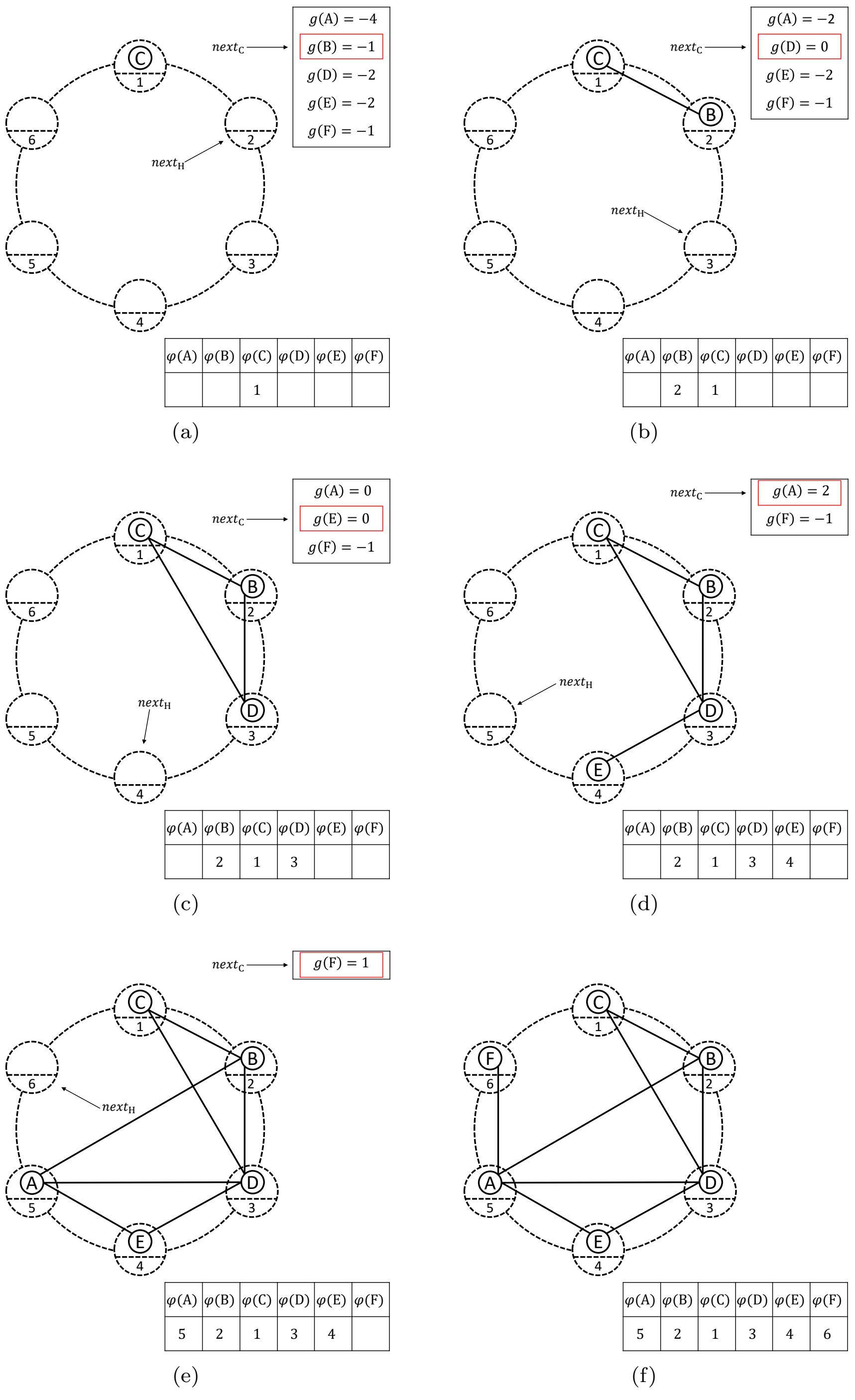

(a)
$next_C$
$g(A) = -4$
$g(B) = -1$
$g(D) = -2$
$g(E) = -2$
$g(F) = -1$
$next_H$
C
1 2 3 4 5 6
$\varphi(A)$ $\varphi(B)$ $\varphi(C)$ $\varphi(D)$ $\varphi(E)$ $\varphi(F)$
1
(b)
$next_C$
$g(A) = -2$
$g(D) = 0$
$g(E) = -2$
$g(F) = -1$
$next_H$
C B
1 2 3 4 5 6
$\varphi(A)$ $\varphi(B)$ $\varphi(C)$ $\varphi(D)$ $\varphi(E)$ $\varphi(F)$
2 1
(c)
$next_C$
$g(A) = 0$
$g(E) = 0$
$g(F) = -1$
$next_H$
C B D
1 2 3 4 5 6
$\varphi(A)$ $\varphi(B)$ $\varphi(C)$ $\varphi(D)$ $\varphi(E)$ $\varphi(F)$
2 1 3
(d)
$next_C$
$g(A) = 2$
$g(F) = -1$
$next_H$
C B D E
1 2 3 4 5 6
$\varphi(A)$ $\varphi(B)$ $\varphi(C)$ $\varphi(D)$ $\varphi(E)$ $\varphi(F)$
2 1 3 4
(e)
$next_C$
$g(F) = 1$
$next_H$
C B D E A
1 2 3 4 5 6
$\varphi(A)$ $\varphi(B)$ $\varphi(C)$ $\varphi(D)$ $\varphi(E)$ $\varphi(F)$
5 2 1 3 4
(f)
C B D E A F
1 2 3 4 5 6
$\varphi(A)$ $\varphi(B)$ $\varphi(C)$ $\varphi(D)$ $\varphi(E)$ $\varphi(F)$
5 2 1 3 4 6

Fig. 3: Example of the steps followed to construct a solution.

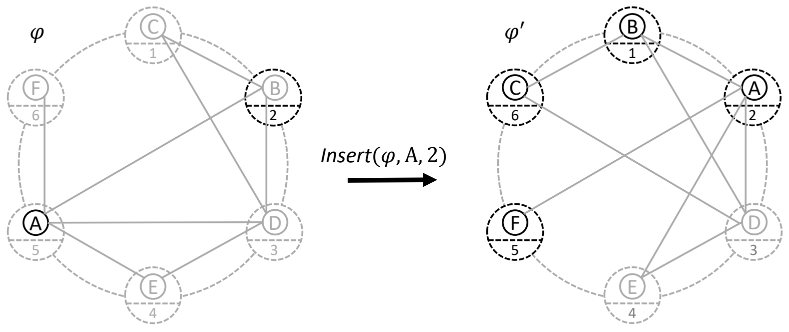


Fig. 4: Example of an embedding φ and the resultant embedding φ' obtained after the operation $Insert(\varphi, A, 2)$.

$$N_{\varphi}(v) = \{Insert(\varphi, v, u) : \forall\, u \in V_{\mathcal{H}}, i \neq \varphi(v)\}$$

For a candidate graph with n vertices, the size of the complete neighborhood (i.e., considering all candidate vertices) is $n \cdot (n-1)$.

Algorithm 2 summarizes the steps of the local search with insert moves. The input to this procedure is the candidate graph ($\mathcal{C}$) and the initial solution (φ). The procedure combines first and best improving strategies. For a candidate vertex v, it finds the best insertion in $N_{\varphi}(v)$ (step 6). However, as soon as an insertion is identified as able to improve the current solution (step 7), the search moves to the solution that results after the insertion (step 8), the improvement flag is switched to True (step 9), and the scanning of the candidate vertices starts again from this new solution (step 10). The while loop (steps 3 to 13) is repeated until no insertion of a candidate vertex is able to improve the current solution.

Algorithm 2 Local search

```
 1: Procedure LocalSearchBestImprovement ($\mathcal{C}, \varphi$)
 2: improve ← True
 3: while improve do
 4:     improve ← False
 5:     for all $v \in V_{\mathcal{C}}$ do
 6:         $\varphi' \leftarrow \arg\min_{\varphi'' \in N_{\varphi}(v)} ccw(\mathcal{C}, \varphi'')$
 7:         if $ccw(\varphi') < ccw(\varphi)$ then
 8:             $\varphi \leftarrow \varphi'$
 9:             improve ← True
10:             break
11:         end if
12:     end for
13: end while
14: return best
```

2.3 Tabu search

Tabu Search (TS) is a metaheuristic originally introduced in 1977 [13] and later formalized in [14] as a general method for solving hard optimization problems. Many ideas and extensions are discussed in [15,16,17,18]. A recent review of the strategies associated with tabu search are compiled in [24]. TS is a, so-called, single-solution neighborhood search metaheuristic methodology. TS introduces the concept of memory with the goal of making the best possible decisions based on the previous information collected throughout the search, instead resorting to randomization.

We add a simple tabu search short-term memory to the local search summarized in Algorithm 2. The TS memory consists of recording a number of recently visited solutions. A move (i.e., an insertion) is classified tabu if it transforms the current solution into a tabu solution (i.e., a solution that is currently in the short-term memory). Unlike TS designs that use memory based on attributes, in our design it is not necessary to include an aspiration criterion, since no tabu move can reach a solution that the search has not already visited. The size of the TS memory (i.e., the number of tabu solutions) is the search parameter known as $TabuTenure$.

Instead of stopping at the first local optimum (i.e., the first time that a move cannot be found to improve the current solution) as in Algorithm 2, the search is allowed to continue by selecting the nonimproving move that deteriorates the objective function the least. This move is the “best” nonimproving move. Before making a move, the current solution is added to the TS memory and the “oldest” tabu solution is deleted. The oldest tabu solution is the one added $TabuTenure$ iterations ago. The search continues after a number of iterations without improvement are executed. The number of nonimproving iterations used to stop is a search parameter ($NonImproving$). Preliminary testing revealed $TabuTenure = 0.2n$ as an effective value for this search parameter. These experiments also showed that the best results can be expected when $NonImproving$ is set to $0.1n$, with a minimum value of 6 and a maximum value of 15. When the TS stops, a new solution is generated with Algorithm 1. This process continues until a maximum time limit is reached, as long as at least 10 but no more than 30 restarts are performed.

3 Advanced search strategies

We now discuss three advanced strategies that are part of our procedure. Although these strategies were designed with the current context in mind, the ideas behind them apply to heuristic searches in other settings. The first strategy deals with landscapes where many solutions have the same objective function value. In this case, the objective function value alone does not provide enough information to find effective search directions. A secondary evaluation is able to differentiate solutions for which objective function values are the same. The second strategy explores computationally efficient ways of calculating move values. This is particularly important in large neighborhoods. The third one also tackles efficiency but from the point of view of reducing the number of moves to evaluate.

3.1 Secondary solution evaluation

In heuristic search, a flat landscape condition occurs when large fractions of the solution space have the same objective function value [36,37]. This means that structurally different solutions may be associated with the same value of the objective function. Determining search directions becomes a very difficult task when decisions are based only on the change of the objective function value produced by a move. Finding a meaningful way of differentiating solutions with the same objective function value is important because the structure of one solution may be more promising than the structure of another in terms of improving the incumbent solution later in the search.

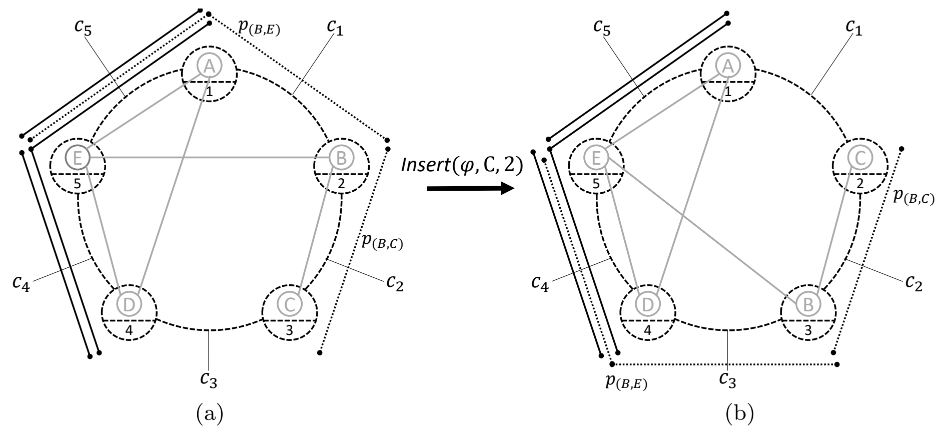
Flat landscapes are typically associated with minmax or maxmin problems [9, 35], that is, those problems where the objective is to minimize a maximum value or to maximize a minimum value. To overcome this difficulty, researchers have proposed the use of one or more secondary solution evaluations. These evaluations are only activated when solutions have the same objective function value and they are designed to indicate preferences regarding the structure of the solutions being compared [33,35].

The secondary evaluation that we employ is based on ideas presented in [33] and [35]. As our problem formulation indicates, the CCMP is an optimization problem that has the goal of finding a solution that minimizes the maximum cut produced by the assignment of candidate vertices to a host graph. It is possible for multiple solution configurations to have the same maximum cut. When comparing two solutions with the same maximum cut (i.e., the same objective function value), we are interested in knowing which one of the two has a better "improvement potential" if a move (or series of moves) could reduce the current maximum cut. The improvement potential can be defined as the difference between the maximum cut and the the second largest cut.

We use the improvement potential concept to differentiate solutions with the same objective function value. In particular, if two solutions have the same objective function value, we calculate the difference between the largest cut (i.e., the objective function value) and the second largest cut. The solution with the larger difference is deemed better (i.e., the solution has the larger improvement potential of the two solutions under consideration). If this calculation is not able to differentiate the solutions, then we calculate the difference between he second largest cut and the third largest cut. We do this until the result of the calculation is able to distinguish between the two solutions.

3.2 Efficient move calculation

The exploration of a solution neighborhood usually is the most computational intensive element in search procedures. Neighborhood searches require the evaluation of moves to determine what to do next. Developing efficient ways of calculating move values is critical in heuristic search. We propose an efficient calculation of the value of the *Insert* move that we defined in Section 2.2. The main idea consists of determining the contribution to the objective function value of the edges corresponding to candidate vertices that are reassigned by the *Insert* move. The



step	cut	c_1	c_2	c_3	c_4	c_5
0	(a)	1	1	0	2	3
1	Remove	-1	-1	0	0	-1
2	Add	0	+1	+1	+1	0
3	(b)	0	1	1	3	2

(c)

Fig. 5: (a) Solution before the move $Insert(\varphi, C, 2)$. (b) Solution after the move $Insert(\varphi, C, 2)$. (c) Efficient revaluation of the objective function.

contribution to the objective function of candidate vertices that are not reassigned does not change.

We illustrate the move evaluation with the example depicted in Figure 5. Figure 5(a) shows the solution before the move of vertex C to position 2, characterized by $Insert(\varphi, \mathrm{C}, 2)$. Figure 5(b) shows the solution after $Insert(\varphi, \mathrm{C}, 2)$. The move evaluation consists of first deleting the paths associated with the edges adjacent to vertices B and C, which are the only vertices that change positions after the move. We also delete the contribution of these paths to the objective function calculation. Then, new paths are assigned to these edges and the contribution of these paths is added to the objective function. The table in Figure 5(c) shows these calculations. Step 0 shows the cuts in the current solution. Step 1 shows the contributions that are removed and step 2 shows the contributions that are added. The cut values associated with the new solution are shown in step 3. These values are obtained by adding the corresponding values in the previous steps.

3.3 Search regions of interest

Exhaustive neighborhood searches become increasingly impractical when the size of the neighborhood grows either polynomially or exponentially with the size of the problem. In our case, the entire neighborhood of a solution is $O(n^2)$. To complement the quick move calculation described in the previous subsection, we add a strategy to focus the search on regions of interest. These regions are reached by a set of promising moves and therefore the strategy consists of identify such moves. This partial exploration of the entire neighborhood is similar to the strategy of moving to the first improving solution, which has been documented to work well in multi-start settings [19]. Our neighborhood search combines the two strategies and in addition we embed the notion of a partial exploration that focuses on regions of interest (ROI).

The ROI of a candidate vertex v, denoted by ROI(v), is the subset of host vertices such that if v were to be assigned to any of them, the current objective function value might change. That is, ROI(v) is the set of host vertices that could cause a reassignment of paths associated with the edges of v. Clearly, the idea here is to avoid changing the assignments of candidate vertices to positions that will cause no changes in the path assignments (and therefore no changes in the objective function value). The exploration of the regions of interests is then done in two steps, we first identify ROI(v) for all v, and then we evaluate all the moves associated with inserting v in all the host vertices in ROI(v). The best move is selected.

4 Experimental results

Before describing our computational experiments and discussion or results, we introduce the instances in our test set (Section 4.1). Preliminary experiments are described in Section 4.2. These experiments are devoted to adjust the parameters of our solution procedure and also to analyze the contribution of the proposed search strategies. We compare the best configuration with the state of the art in Section 4.3.

4.1 Instances

We use the following set of instances from the literature in our computational tests [21].

- **Small graphs**: the number of vertices and edges in these random graphs varies between 16 and 24, and between 18 and 49, respectively. There are 84 instances in this set.
- **Harwell-Boeing graphs**: these graphs arise from a wide variety of problems in scientific and engineering disciplines. The selected problems are a diverse subset of the original Harwell-Boeing set [10] . Particularly, we have selected the 38 graphs used in [21]. These graphs have sizes between 39 and 685 vertices and from 46 to 3720 edges.
- **Regular graphs**: these subset includes four different types of graphs (Complete Split Graph, Toroidal Mesh, Join of Hypercubes, and Cone Graph) with a predefined and well-known structure, but with an unknown optima. This set includes a total of 57 graphs with a number of vertices ranging from 12 to 1000 and a number of edges ranging from 46 to 6225.

Our experiments do no include the set of regular graphs with known optimal solutions included in [21]. These graphs are such that do not provide insightful results in comparative testing.

4.2 Preliminary experiments

Our preliminary experiments with a random sample consisting of 10% of all our problem instances (i.e., 18 graphs) have the goal of identifying the best values for the search parameters as well as assess the merit of the proposed search strategies and mechanisms.

The procedure to construct solutions described in Section 2.1 is not totally deterministic. The first candidate vertex to be assigned is chosen at random. Also, when the greedy value is the same for all unassigned candidate vertices, ties are broken arbitrarily. Therefore, it is interesting to observe the performance of the procedure when constructing more than one solution. Table 1 reports the average value of the objective function (Avg.), the deviation of the best solution in the current run with respect to the overall best within the experiment (Dev.(%)), the number of best solutions found in the experiment (#Best) and the time in seconds (CPU Time (s)) when applying the procedure to construct 1, 5, 10, 20, 30, 40, and 50 solutions. Note that we are not comparing against the best known solutions but against the best solutions found within this experiment. The results in Table 1 show that the best solutions for the 18 instances in the sample are found when the procedure is run 50 times. The improvement is significant from executing the procedure one time to executing it fifty times. However, the difference in solution quality is negligible after executing the procedure twenty times.

Iterations	1	5	10	20	30	40	50
Avg.	60.61	57.28	55.00	54.17	54.11	54.06	53.94
Dev. (%)	32.82	12.83	6.20	0.33	0.28	0.17	0.00
#Best	3	9	11	15	16	17	18
CPU Time (s)	0.004	0.010	0.015	0.024	0.032	0.041	0.048

Table 1: Performance of the constructive procedure based on the number of iterations.

We then compared the performance of our greedy construction (Greedy) and a totally random construction (Random). Table 2 reports the results of runs with a time limit of one second for each graph. As expected, the number of random constructions is larger than the number of greedy constructions. The difference is that the greedy procedure evaluates each unassigned candidate vertex before selecting it. The solution quality indicators favor the greedy constructions over a totally random approach.

The second preliminary experiment is devoted to testing the impact of the advanced strategies introduced in Section 3. We start by running the local search of Section 2.2 and reporting the results in Table 3 (LS). The table compares these results with the outcomes from running LS with the efficient move evaluation of Section 3.2 (LS+E) and the results from focusing on regions of interest of Section 3.3 (LS+E+ROI). Since the local search follows a strictly descent pattern, the secondary evaluation of Section 3.1 does not play a role. Recall that the secondary evaluation is used to distinguish between moves that result in no change of the objective function value.

	Random	Greedy
Avg.	87.22	53.06
Dev. (%)	113.00	9.46
#Best	4	14
Avg. #Cons.	151098.83	55025.72

Table 2: Comparison between random and greedy constructions.

	LS	LS+E	LS+E+ROI
Avg.	43.61	43.61	43.61
#Best	18	18	18
CPU Time (s)	591.98	35.93	0.53
Avg. #Sol.	4312427.50	4312427.50	60463.78

Table 3: Contribution of advanced strategies to the local search.

	Greedy	LS+E+ROI	TS
Avg.	60.61	43.61	43.06
Dev. (%)	41.03	3.15	0.11
#Best	0	12	17
CPU Time (s)	0.01	0.51	2.14

Table 4: Performance differences between the procedure components and the full procedure.

Table 3 reveals that all variants are able to reach the same solution quality. Moving from LS to LS+E, we observe the decrease of one order of magnitud in computational time to explore the same number of solutions. The neighborhood reduction strategy associated with ROI is able to further reduce the computational burden (by two addional orders of magnitud). We point out that all variants were run starting from the same initial solution.

To adjust the two parameters associated with the tabu search elements in our procedure, we performed a full factorial design with values that we made dependent on the graph size. In particular, for the tabu tenure we tested 5%, 10%, 20%, 30%, and 40% of n, where n is the number of vertices in the candidate graph. We tested the same percentages for the maximum number of iterations without improvement, and limited the value to be within a minimum (6 iterations) and a maximum (15 iterations). The experiment identified $20\%n$ as the best setting for tabu tenure and $10\%n$ as the best setting for the number of iterations without improvement.

Our final preliminary experiment explores the increase in solution quality when going from simple greedy constructions (Greedy) to the local search with advanced strategies (LS+E+ROI) and to the tuned procedure that includes TS elements (TS). Table 4 reports the results. As expected, adding local search results in a noticeable improvement in solution quality. The embedding of tabu search elements results in additional quality improvement, although the difference between LS+E+ROI and TS is significantly smaller than the solution quality difference between Greedy and LS+E+ROI.

			TS	MA
Random graphs	Small Instances (84)	Avg.	3.90	3.98
		Dev. (%)	0.00	1.92
		#Best	84	78
		CPU Time (s)	0.32	15
	Harwell-Boeing (38)	Avg.	67.26	70.71
		Dev. (%)	2.40	12.09
		#Best	31	10
		CPU Time (s)	187.59	4580.61
Regular-structured graphs	Complete Split Graph (21)	Avg.	217.76	217.34
		Dev. (%)	0.18	0.00
		#Best	13	21
		CPU Time (s)	994.16	2548.43
	Join of Hypercubes (9)	Avg.	80.89	80.34
		Dev. (%)	0.27	0.00
		#Best	7	9
		CPU Time (s)	56.08	406.89
	Toroidal Mesh (17)	Avg.	30.24	30.24
		Dev. (%)	1.47	1.24
		#Best	13	16
		CPU Time (s)	142.31	1898.71
	Cone Graph (10)	Avg.	154.80	154.90
		Dev. (%)	0.10	0.13
		#Best	9	8
		CPU Time (s)	80.32	1596.80
Total		Avg.	57.24	57.94
		Dev. (%)	0.69	3.59
		#Best	157	142
		CPU Time (s)	177.42	1568.42

Table 5: Comparison with the state of the art.

4.3 Competitive Testing

In our competitive testing, we compare our tuned procedure with the one proposed in [21], the memetic algorithm that we mention in the literature review. Table 5 shows the results of this test. The results are grouped by set and graph type (i.e., random or structured).TS refers to our proposed method and MA refers to the memetic algorithm in [21]. We use the same metric as before, where average deviation form best (Dev.) is calculated considering the collective-best solutions found with either TS or MA. The ”Total” section at the bottom of the table provides averages across all graph types.

The general observations from examining this table are that:

– The TS solutions are found in about one order of magnitude less time than MA
– TS's performance is better on random graphs than on structured graphs
– For all problem types, TS solutions are on average closer to the best solutions (maximum deviation of 1.47%) than the MA solutions (maximum deviation of 12.09%)

We performed two statistical tests with the goal of identifying significant performance differences. Specifically, we use Wilcoxon's signed rank test [46] to identify difference between the objective function values of the best solutions found by

TS and MA. We apply the one-tail version of this paired test to random graphs and structured graphs separately. Our null hypothesis is that there is no difference in the median of the objective function values, while the alternative hypothesis is that the median of one set of values is smaller than the other. The first test with all 122 random graphs results in a p-value of 0.0008. Therefore, we can confidently reject the null hypothesis in favor of the alternative hypothesis that the median of the TS objective function values is less than the median of the MA values. The Wilcoxon test with all 57 structured graphs results in a p-value of 0.0069. This also indicates a strong rejection of the null hypothesis in favor of concluding that the median of the MA objective function values for structured graphs is less than the median of the TS values. We point out that while the Wilcoxon tests detect significant difference on the median values, the corresponding t-test for paired sample means result in p-values of 0.032 and 0.106 for random and structured graphs, respectively. This means that, in the case of random graphs, we could still reject the null hypothesis at a reasonable level of significance, say 5%. However, we would have to accept a Type I error of over 10% if we would like to reject the null hypothesis for the structured graphs in favor of concluding that the average MA objective function values for structured graphs is less than the TS average.

We believe that the solid performance of MA on structured graphs is due to the six different ways in which solutions are constructed to initialize the search. At least one of this constructions can be customized to exploit a particular regular structure and give the procedure the advantage of starting the search at high-quality initial point. Customization of a solution-construction procedure is not possible for graphs without a regular structure, such as as random graphs. Including various forms of constructing solutions within a single procedure is indeed a reasonable idea as long as the application can afford the price to be paid on the increased computational effort.

Appendix A includes the individual results of our competitive tests. This could help researchers to perform future comparisons.

5 Conclusions

We studied the Cyclic Cutwidth Minimization Problem consisting of embedding a candidate graph into a cycle (host) graph in order to minimize the maximum cut. This problem has been previously studied for specific classes of graphs with a regular structure. However, work on general candidate graphs is sparse. We can point to only one recent heuristic approach for general graphs. The approach in the literature is a population-based metaheuristic from the family of memetic algorithms. We took a different approach and developed a single-solution, neighborhood search. In the process of creating an effective and efficient solution method, we adapted three strategies that have general applicability:

1. Efficient move value calculation.
2. Secondary move evaluation to distinguish moves that the primary evaluation (based on the objective function) is not able to distinguish.
3. Neighborhood search space reduction via regions of interest.

Our work establishes some new benchmarks for the Cyclic Cutwidth Minimization Problem and provides validation for strategies that promise to accelerate the execution of heuristic searches without sacrificing solution quality.

Acknowledgment

This work was partially supported by the Ministerio de Ciencia, Innovación y Universidades under Grant Ref. PGC2018-095322-B-C22 and Comunidad de Madrid and European Regional Development Fund with Grant Ref. S2018/TCS-4566.

References

1. H. L. Abbott. Hamiltonian circuits and paths on the n-cube. *Canadian Mathematical Bulletin*, 9(5):557–562, 1966.
2. H. Allmond. On the cyclic cutwidth of complete tripartite and n-partite graphs. *CSUSB REU*, 2006.
3. R. Aschenbrenner. A proof for the cyclic cutwidth of q5. *REU Project, Cal State Univ., San Bernardino*, 2001.
4. C. Castillo. A proof for the cyclic cutwidth of q6. *REU Project, Cal State Univ., San Bernardino*, 2003.
5. J. D. Chavez and R. Trapp. The cyclic cutwidth of trees. *Discrete Applied Mathematics*, 87(1-3):25–32, 1998.
6. D. W. Clarke. *The cyclic cutwidth of mesh cubes.* 2002.
7. J. P. Cohoon and S. Sahni. Heuristics for backplane ordering. *Journal of VLSI and computer systems*, 2(1-2):37–60, 1987.
8. J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, September 2002.
9. A. Duarte, J. J. Pantrigo, E. G. Pardo, and J. Sánchez-Oro. Parallel variable neighbourhood search strategies for the cutwidth minimization problem. *IMA Journal of Management Mathematics*, 27(1):55, 2016.
10. I. S. Duff, R. G. Grimes, and J. G. Lewis. Users' guide for the harwell-boeing sparse matrix collection (release i), 1992.
11. J. Erbele, J. D. Chavez, and R. Trapp. The cyclic cutwidth of qn. *Manuscript, California State University, San Bernardino USA*, 2003.
12. F. Gavril. Some np-complete problems on graphs. In *Proceedings of the eleventh conference on information sciences and systems*, pages 91–95, 1977.
13. F. Glover. Heuristics for integer programming using surrogate constraints. *Decision sciences*, 8(1):156–166, 1977.
14. F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.
15. F. Glover. Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206, 1989.
16. F. Glover. Tabu search—part ii. *ORSA Journal on computing*, 2(1):4–32, 1990.
17. F. Glover. Tabu search and adaptive memory programming—advances, applications and challenges. In *Interfaces in computer science and operations research*, pages 1–75. Springer, 1997.
18. F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, USA, 1997.
19. P. Hansen and N. Mladenović. First vs. best improvement: An empirical study. *Discrete Applied Mathematics*, 154(5):802 – 817, 2006. IV ALIO/EURO Workshop on Applied Combinatorial Optimization.
20. L. H. Harper. Optimal numberings and isoperimetric problems on graphs. *Journal of Combinatorial Theory*, 1(3):385 – 393, 1966.
21. P. Jain, K. Srivastava, and G. Saran. Minimizing cyclic cutwidth of graphs using a memetic algorithm. *Journal of Heuristics*, 22(6):815–848, 2016.
22. B. James. The cyclical cutwidth of the three-dimensional and fourdimensional cubes. *Cal State Univ., San Bernardino McNair Scholar's Program Summer Research Journal*, 1996.
23. M. Johnson. The linear and cyclic cutwidth of the complete bipartite graph. *REU Project, Cal State Univ., San Bernardino*, 2003.
24. M. Laguna. *Tabu Search*, pages 741–758. Springer International Publishing, Cham, 2018.
25. M. C. López-Locés, N. Castillo-García, H. J. F. Huacuja, P. Bouvry, J. E. Pecero, R. A. P. Rangel, J. J. G Barbosa, and F. Valdez. A new integer linear programming model for the cutwidth minimization problem of a connected undirected graph. In *Recent Advances on Hybrid Approaches for Designing Intelligent Systems*, pages 509–517. Springer, 2014.

26. J. Luttamaguzi, M. Pelsmajer, Z. Shen, and B. Yang. Integer programming solutions for several optimization problems in graph theory. Technical report, Center for Discrete Mathematics and Theoretical Computer Science, DIMACS, 2005.
27. F. Makedon and I. H. Sudborough. On minimizing width in linear layouts. *Discrete Applied Mathematics*, 23(3):243–265, 1989.
28. V. G. Martins Santos and A. M. Moreira de Carvalho. Tailored heuristics in adaptive large neighborhood search applied to the cutwidth minimization problem. *European Journal of Operational Research*, 2019.
29. R. Martí, J. J. Pantrigo, A. Duarte, and E. G. Pardo. Branch and bound for the cutwidth minimization problem. *Computers & Operations Research*, 40(1):137 – 149, 2013.
30. A. J. Mcallister. A new heuristic algorithm for the linear arrangement problem. 1999.
31. P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989, 1989.
32. G. Palubeckis. A branch-and-bound algorithm for the single-row equidistant facility layout problem. *OR Spectrum*, 34(1):1–21, Jan 2012.
33. J. J. Pantrigo, R. Martí, A. Duarte, and E. G. Pardo. Scatter search for the cutwidth minimization problem. *Annals of Operations Research*, 199(1):285–304, 2012.
34. E. G. Pardo, R. Martí, and A. Duarte. *Linear Layout Problems*. Handbook of Heuristics. Springer International Publishing. ISBN: 978-3-319-07153-4, 2016.
35. E. G. Pardo, N. Mladenović, J. J. Pantrigo, and A. Duarte. Variable formulation search for the cutwidth minimization problem. *Appl. Soft Comput.*, 13(5):2242–2252, May 2013.
36. E. Pinana, I. Plana, V. Campos, and R. Martı. Grasp and path relinking for the matrix bandwidth minimization. *European Journal of Operational Research*, 153(1):200–210, 2004.
37. M. G. C. Resende, R. Martí, M. Gallego, and A. Duarte. Grasp and path relinking for the max–min diversity problem. *Computers & Operations Research*, 37(3):498–508, 2010.
38. M. G. C. Resende and Andrade D. V. In *Method and system for network migration scheduling. United States Patent Application Publication. US2009/0168665*, 2009.
39. F.R. Rios. Complete graphs as a first step toward finding the cyclic cutwidth of the n-cube. *Cal State Univ., San Bernardino McNair Scholar's Program Summer Research Journal*, 1996.
40. J. Rolim, O. Sỳkora, and I. Vrt'o. Optimal cutwidths and bisection widths of 2-and 3-dimensional meshes. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 252–264. Springer, 1995.
41. H. Schröder, O. Sỳkora, and I. Vrt'o. Cyclic cutwidths of the two-dimensional ordinary and cylindrical meshes. *Discrete applied mathematics*, 143(1):123–129, 2004.
42. H. Schröder, O. Sỳykoa, and I. Vrt'o. Cyclic cutwidth of the mesh. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 449–458. Springer, 1999.
43. V. Sciortino, J. D. Chavez, and R. Trapp. The cyclic cutwidth of a p2× p2× pn mesh. *REU Project, Cal State Univ., San Bernardino*, 2002.
44. F. Shahrokhi, O. Sỳkora, L. A. Székely, and I. Vrt'o. On bipartite drawings and the linear arrangement problem. *SIAM Journal on Computing*, 30(6):1773–1789, 2001.
45. D. M Thilikos, M. Serna, and H. L. Bodlaender. Cutwidth ii: Algorithms for partial w-trees of bounded degree. *Journal of Algorithms*, 56(1):25–49, 2005.
46. F. Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.

Appendix A Individual results

These are the results of our competitive testing. These values were used to calculate the summary presented in Table 5.

Small Instances

	TS			MA		
Instance	*ccw*	**CPUt (s)**	**Dev. (%)**	*ccw*	**CPUt (s)**	**Dev. (%)**
p17_16_24	5	0.27	0.00	5	<15	0.00
p18_16_21	4	0.20	0.00	4	<15	0.00
p19_16_19	3	0.26	0.00	3	<15	0.00
p20_16_18	4	0.20	0.00	4	<15	0.00
p21_17_20	3	0.12	0.00	3	<15	0.00
p22_17_19	3	0.24	0.00	3	<15	0.00
p23_17_23	4	0.39	0.00	4	<15	0.00
p24_17_29	5	0.39	0.00	6	<15	20.00
p25_17_20	3	0.20	0.00	3	<15	0.00
p26_17_19	3	0.16	0.00	3	<15	0.00
p27_17_19	3	0.30	0.00	3	<15	0.00
p28_17_18	3	0.18	0.00	3	<15	0.00
p29_17_18	3	0.16	0.00	3	<15	0.00
p30_17_19	3	0.20	0.00	3	<15	0.00
p31_18_21	3	0.22	0.00	3	<15	0.00
p32_18_20	3	0.32	0.00	3	<15	0.00
p33_18_21	3	0.17	0.00	3	<15	0.00
p34_18_21	3	0.32	0.00	3	<15	0.00
p35_18_19	3	0.09	0.00	3	<15	0.00
p36_18_20	3	0.24	0.00	3	<15	0.00
p37_18_20	4	0.14	0.00	4	<15	0.00
p38_18_19	3	0.15	0.00	3	<15	0.00
p39_18_19	3	0.26	0.00	3	<15	0.00
p40_18_32	6	0.68	0.00	6	<15	0.00
p41_19_20	3	0.16	0.00	3	<15	0.00
p42_19_24	4	0.18	0.00	4	<15	0.00
p43_19_22	3	0.22	0.00	3	<15	0.00
p44_19_25	4	0.35	0.00	4	<15	0.00
p45_19_25	4	0.23	0.00	4	<15	0.00
p46_19_20	3	0.16	0.00	3	<15	0.00
p47_19_21	3	0.19	0.00	3	<15	0.00
p48_19_21	3	0.14	0.00	3	<15	0.00
p49_19_22	3	0.21	0.00	3	<15	0.00
p50_19_25	3	0.29	0.00	3	<15	0.00
p51_20_28	4	0.56	0.00	5	<15	25.00
p52_20_27	4	0.34	0.00	4	<15	0.00
p53_20_22	3	0.27	0.00	3	<15	0.00
p54_20_28	5	0.32	0.00	5	<15	0.00
p55_20_24	3	0.14	0.00	3	<15	0.00
p56_20_23	3	0.30	0.00	3	<15	0.00
p57_20_24	4	0.33	0.00	4	<15	0.00
p58_20_21	3	0.07	0.00	3	<15	0.00
p59_20_23	4	0.37	0.00	4	<15	0.00
p60_20_22	3	0.63	0.00	4	<15	33.33
p61_21_22	3	0.27	0.00	3	<15	0.00
p62_21_30	5	0.34	0.00	5	<15	0.00
p63_21_42	7	0.62	0.00	7	<15	0.00
p64_21_22	3	0.17	0.00	3	<15	0.00
p65_21_24	3	0.39	0.00	3	<15	0.00

	TS			**MA**		
Instance	*ccw*	**CPUt (s)**	**Dev. (%)**	*ccw*	**CPUt (s)**	**Dev. (%)**
p66_21_28	5	0.42	0.00	5	<15	0.00
p67_21_22	3	0.07	0.00	3	<15	0.00
p68_21_27	4	0.26	0.00	4	<15	0.00
p69_21_23	3	0.19	0.00	4	<15	33.33
p70_21_25	4	0.26	0.00	4	<15	0.00
p71_22_29	4	0.38	0.00	4	<15	0.00
p72_22_49	9	1.26	0.00	9	<15	0.00
p73_22_29	4	0.38	0.00	4	<15	0.00
p74_22_30	5	0.51	0.00	5	<15	0.00
p75_22_25	4	0.32	0.00	4	<15	0.00
p76_22_30	4	0.84	0.00	5	<15	25.00
p77_22_37	6	0.56	0.00	6	<15	0.00
p78_22_31	5	0.37	0.00	5	<15	0.00
p79_22_29	5	0.39	0.00	5	<15	0.00
p80_22_30	5	0.38	0.00	5	<15	0.00
p81_23_46	8	0.45	0.00	8	<15	0.00
p82_23_24	4	0.25	0.00	4	<15	0.00
p83_23_24	4	0.17	0.00	4	<15	0.00
p84_23_26	4	0.27	0.00	4	<15	0.00
p85_23_26	4	0.32	0.00	4	<15	0.00
p86_23_24	3	0.22	0.00	3	<15	0.00
p87_23_30	4	0.28	0.00	4	<15	0.00
p88_23_26	4	0.16	0.00	4	<15	0.00
p89_23_27	4	0.33	0.00	4	<15	0.00
p90_23_35	5	0.41	0.00	5	<15	0.00
p91_24_33	5	0.33	0.00	5	<15	0.00
p92_24_26	4	0.40	0.00	4	<15	0.00
p93_24_27	4	0.40	0.00	4	<15	0.00
p94_24_31	4	0.49	0.00	5	<15	25.00
p95_24_27	4	0.44	0.00	4	<15	0.00
p96_24_27	3	0.38	0.00	3	<15	0.00
p97_24_26	4	0.36	0.00	4	<15	0.00
p98_24_29	4	0.66	0.00	4	<15	0.00
p99_24_27	4	0.14	0.00	4	<15	0.00
p100_24_34	5	0.41	0.00	5	<15	0.00
Avg.	3.90	0.32	0.00	3.98	<15	1.92

Harwell-Boeing

	TS			**MA**		
Instance	*ccw*	**CPUt (s)**	**Dev. (%)**	*ccw*	**CPUt (s)**	**Dev. (%)**
494_bus	25	288.88	25.00	20	3934	0.00
662_bus	31	306.09	0.00	38	18010	22.58
685_bus	37	304.98	0.00	50	18007	35.14
arc130	122	302.89	0.00	143	2841	17.21
ash292	34	309.38	0.00	47	5102	38.24
ash85	14	13.37	0.00	16	295	14.29
bcspwr01	4	0.45	0.00	5	80	25.00
bcspwr02	5	1.46	0.00	5	88	0.00
bcspwr03	10	18.36	0.00	11	512	10.00
bcspwr04	36	306.27	2.86	35	6755	0.00
bcspwr05	25	82.35	0.00	26	4429	4.00
bcsstk01	25	12.03	0.00	27	100	8.00
bcsstk02	561	8.42	2.75	546	164	0.00

	TS			**MA**		
Instance	*ccw*	**CPUt (s)**	**Dev. (%)**	*ccw*	**CPUt (s)**	**Dev. (%)**
bcsstk04	302	410.32	0.00	311	12814	2.98
bcsstk05	113	165.78	0.00	114	2988	0.88
bcsstk06	254	2125.71	18.14	215	22232	0.00
bcsstk22	10	43.15	0.00	11	550	10.00
can__144	25	53.82	25.00	20	1472	0.00
can__161	45	84.09	0.00	48	656	6.67
can__292	67	309.42	0.00	105	10625	56.72
curtis54	11	6.20	0.00	12	75	9.09
dwt__209	47	147.80	0.00	55	6368	17.02
dwt__221	27	90.53	0.00	28	9260	3.70
dwt__234	12	8.40	0.00	14	1080	16.67
dwt__245	32	140.78	0.00	38	3575	18.75
fs_183_1	113	303.52	0.00	136	7774	20.35
gent113	70	187.25	0.00	80	1932	14.29
gre__115	25	33.82	0.00	28	1028	12.00
gre__185	46	79.24	9.52	42	3554	0.00
ibm32	15	3.40	0.00	15	60	0.00
impcol_b	44	15.69	0.00	47	278	6.82
impcol_c	33	34.09	0.00	41	1494	24.24
lns__131	24	18.29	0.00	25	748	4.17
lund_a	107	221.78	8.08	99	3918	0.00
lund_b	100	300.58	0.00	101	3048	1.00
saylr3	46	302.45	0.00	60	18000	30.43
west0132	48	84.28	0.00	62	117	29.17
will57	11	2.96	0.00	11	100	0.00
Avg.	67.26	187.59	2.40	70.71	4580.61	12.09

Complete Split

	TS			**MA**		
Instance	*ccw*	**CPUt (s)**	**Dev. (%)**	*ccw*	**CPUt (s)**	**Dev. (%)**
CompleteSplit_K4_K10	13	0.92	0.00	13	29	0.00
CompleteSplit_K4_K15	17	0.48	0.00	17	50	0.00
CompleteSplit_K4_K20	23	2.63	0.00	23	68	0.00
CompleteSplit_K4_K30	33	5.27	0.00	33	122	0.00
CompleteSplit_K4_K50	53	16.71	0.00	53	278	0.00
CompleteSplit_K4_K100	103	98.15	0.00	103	103	0.00
CompleteSplit_K5_K5	10	0.93	0.00	10	20	0.00
CompleteSplit_K5_K10	16	1.20	0.00	16	37	0.00
CompleteSplit_K5_K20	29	3.26	0.00	29	89	0.00
CompleteSplit_K5_K50	68	15.71	0.00	68	309	0.00
CompleteSplit_K5_K100	133	117.84	0.00	133	879	0.00
CompleteSplit_K6_K15	27	1.81	0.00	27	80	0.00
CompleteSplit_K6_K50	81	30.88	1.25	80	477	0.00
CompleteSplit_K6_K100	156	145.45	0.65	155	1446	0.00
CompleteSplit_K10_K15	50	5.09	0.00	50	156	0.00
CompleteSplit_K10_K50	139	91.92	0.78	138	787	0.00
CompleteSplit_K10_K100	264	310.32	0.38	263	2604	0.00
CompleteSplit_K20_K50	302	310.33	0.33	301	2342	0.00
CompleteSplit_K20_K100	552	925.71	0.18	551	7523	0.00
CompleteSplit_K50_K50	940	2705.05	0.21	938	18059	0.00
CompleteSplit_K50_K100	1564	16087.62	0.06	1563	18059	0.00
Avg.	217.76	994.16	0.18	217.33	2548.43	0.00

Torodial Mesh

	TS			**MA**		
Instance	*ccw*	**CPUt (s)**	**Dev. (%)**	*ccw*	**CPUt (s)**	**Dev. (%)**
ToroidalMC3xC3xC3	14	0.52	0.00	14	58	0.00
ToroidalMC3xC3xC4	17	1.22	0.00	17	81	0.00
ToroidalMC3xC3xC5	17	1.73	0.00	17	108	0.00
ToroidalMC3xC3xC10	17	20.20	0.00	17	296	0.00
ToroidalMC3xC4xC4	20	2.43	0.00	20	118	0.00
ToroidalMC3xC4xC5	20	7.30	0.00	20	167	0.00
ToroidalMC3xC4xC10	22	56.54	10.00	20	536	0.00
ToroidalMC3xC5xC5	23	7.38	0.00	23	308	0.00
ToroidalMC3xC5xC10	23	89.24	0.00	23	951	0.00
ToroidalMC3xC10xC10	38	300.66	0.00	46	6310	21.05
ToroidalMC4xC4xC4	26	5.82	0.00	26	215	0.00
ToroidalMC4xC4xC5	26	18.04	0.00	26	432	0.00
ToroidalMC4xC4xC10	26	115.00	0.00	26	1198	0.00
ToroidalMC4xC5xC5	32	31.46	6.67	30	992	0.00
ToroidalMC4xC5xC10	32	260.36	6.67	30	1693	0.00
ToroidalMC5xC5xC5	37	50.13	0.00	37	668	0.00
ToroidalMC10xC10xC10	124	1451.31	1.64	122	18147	0.00
Avg.	30.24	142.31	1.47	30.24	1898.71	1.24

Join of Hypercubes

	TS			**MA**		
Instance	*ccw*	**CPUt (s)**	**Dev. (%)**	*ccw*	**CPUt (s)**	**Dev. (%)**
Hypercube_Q2+Q3	12	1.02	0.00	12	29	0.00
Hypercube_Q2+Q4	23	2.25	0.00	23	63	0.00
Hypercube_Q2+Q5	46	10.77	0.00	46	204	0.00
Hypercube_Q3+Q3	21	2.92	0.00	21	55	0.00
Hypercube_Q3+Q4	40	7.11	0.00	40	118	0.00
Hypercube_Q3+Q5	79	34.47	0.00	79	375	0.00
Hypercube_Q4+Q4	76	11.67	0.00	76	260	0.00
Hypercube_Q4+Q5	147	132.50	1.38	145	752	0.00
Hypercube_Q5+Q5	284	302.05	1.07	281	1806	0.00
Avg.	80.89	56.08	0.27	80.33	406.89	0.00

Cones

	TS			**MA**		
Instance	*ccw*	**CPUt (s)**	**Dev. (%)**	*ccw*	**CPUt (s)**	**Dev. (%)**
Cone10_10	26	1.58	0.00	26	76	0.00
Cone10_15	39	4.64	0.00	39	195	0.00
Cone10_20	51	3.67	0.00	51	207	0.00
Cone10_50	126	46.89	0.00	127	868	0.79
Cone15_15	58	17.40	0.00	58	197	0.00
Cone15_20	77	18.98	0.00	77	477	0.00
Cone15_50	190	162.52	0.00	191	1924	0.53
Cone20_20	102	9.65	0.99	101	599	0.00
Cone20_50	252	226.61	0.00	252	2810	0.00
Cone50_50	627	311.24	0.00	627	8615	0.00
Avg.	154.80	80.32	0.10	154.90	1596.80	0.13