Shortest Circular Paths on Planar Graphs

Dirk Farin
Eindhoven University of Technology
P.O.Box 513
5600 MB Eindhoven
The Netherlands
d.s.farin@tue.nl

Peter H. N. de With
LogicaCMG & TU/e
P.O.Box 513
5600 MB Eindhoven
The Netherlands
p.h.n.de.with@tue.nl

Abstract

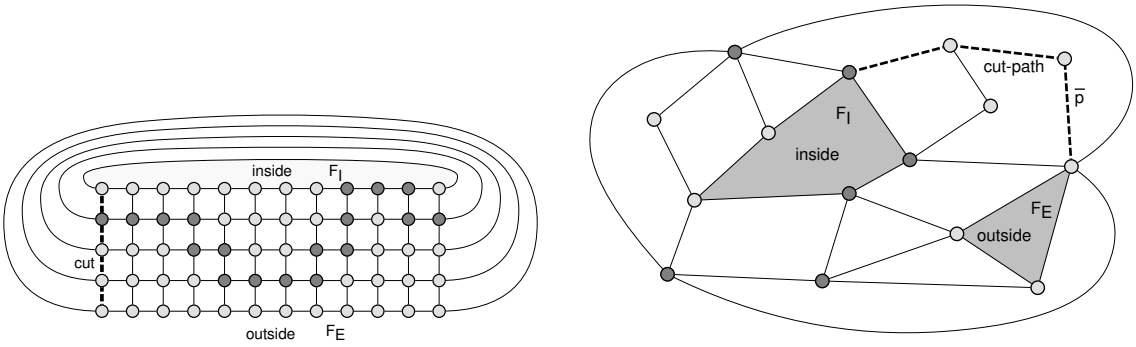
The shortest circular-path problem is similar to the ordinary shortest path problem, but instead of specifying a distinguished starting and end node, the path has to form a closed loop. This problem has several applications in optimization problems as they occur in image segmentation or shape matching. We propose a new algorithm to compute shortest circular paths on planar graphs that has a typical computation time of $O(|V| \log |V|)$ and a worst case complexity of $O(|V| \log^2 |V|)$. The fastest previously known algorithm for this problem has an average computation time of $O(|V| \log^2 |V|)$ and a worst case performance of $O(|V|^2 \log |V|)$.

1 Introduction

Computing shortest paths in graphs is a well-studied problem with numerous applications in coding (Viterbi decoding), optimization, networking, and pattern analysis. A closely related problem is the computation of shortest *circular* paths, i.e. closed paths for which no specific start node or end node is known. This problem arises in several areas like video-object segmentation [2], or shape matching [4]. Surprisingly, the problem of shortest circular paths has so far received little attention. Previously proposed algorithms include the *Multiple Search Algorithm* (MSA) or the *Image Patching Algorithm* (IPA) [5]. However, MSA has a high computational complexity of $O(|V|^{3/2} \log |V|)$, where $|V|$ is the number of nodes, and the IPA cannot guarantee to find the optimal solution. The fastest previously known algorithm for this problem is a branch-and-bound algorithm [1] with an average computation time of $O(|V| \log^2 |V|)$ and a worst case performance of $O(|V|^2 \log |V|)$.

We propose a new algorithm for computing shortest circular paths with a computation complexity of only $O(|V| \log |V|)$ in the average case and $O(|V| \log^2 |V|)$ in the worst case. Note that the average computation complexity of our algorithm is optimal since it equals the complexity of the standard shortest path problem. For graphs with nodes on a regular lattice, the computation time of our algorithm can be further reduced with dynamic-programming techniques to only $O(|V|)$ in the average case.

The paper first introduces the shortest circular-path problem in Section 2 and derives some basic properties of shortest paths in Section 3. The new algorithm is described in Section 4 and its computational complexity is analysed in Section 5.



(a) Circular-grid graph. (b) General planar graph.

Figure 1: Two graphs with examples of valid circular paths (dark nodes). In each, an example cut-path (path between inside and outside face) is indicated.

2 Shortest circular-path problem

Let $G = (V, E)$ be a planar graph with nodes V and edges E. Furthermore, the graph is assumed to be embedded in the plane with two of its faces denoted as F_I, F_E (see Fig. 1). We denote any path connecting F_I and F_E a *cut-path* $\bar{p}$. Furthermore, we define that a cyclic path $\dot{p} = v_i \rightsquigarrow v_i$ is a *circular path* on the graph G if $\dot{p}$ and any cut-path $\bar{p}$ have at least one node in common. Intuitively, this means that a circular path $\dot{p}$ separates the faces F_I (interior) and F_E (exterior).

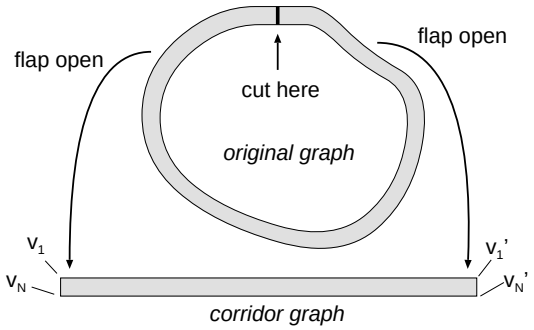
Instead of computing a shortest circular path on the original graph, it is convenient to transform it first into a *corridor graph*. The construction of the corridor graph can be imagined as cutting the original ring-shaped graph along a cut-path to obtain a lane-shaped graph. Formally, this construction can be described as follows.

Construction (corridor graph): Let $G = (V, E)$ be a planar graph and $\bar{p} = v_1 v_2 \cdots v_N$ a cut-path between F_I and F_E, consisting of the nodes $\bar{V} = \{v_1, \ldots, v_N\}$. Furthermore, we assume the following property of G: if V_n is the set of nodes adjacent to nodes in $\bar{V}$, then V_n comprises exactly two connected components V_l, V_r. If G has this property, we can transform G into a *corridor graph* $G' = (V', E')$ as follows. We set $V' = V \cup \bar{V}'$, where $\bar{V}' = \{v'_1, \ldots, v'_N\}$ forms a copy of $\bar{V}$. The edges are defined as

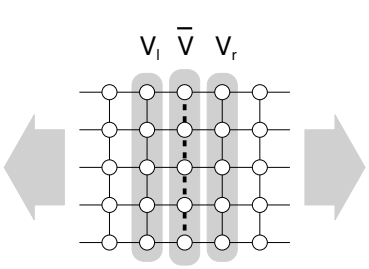
$$E' = (E \underbrace{\setminus (V_r \times \bar{V})}_{\substack{\text{cut graph between } V_r \text{ and } \bar{V}}}) \cup \{\underbrace{\{v_i \in V_r, v'_k \in \bar{V}'\}}_{\text{reconnect } V_r \text{ with } \bar{V}'} \mid \{v_i, v_k\} \in E\}. \tag{1}$$

This copies the nodes $\bar{V}$ on the cut-path to $\bar{V}'$ and reconnects the adjacent nodes V_l and V_r such that V_l connects only to $\bar{V}$, and V_r only to $\bar{V}'$. The construction is visualized in Fig. 2.

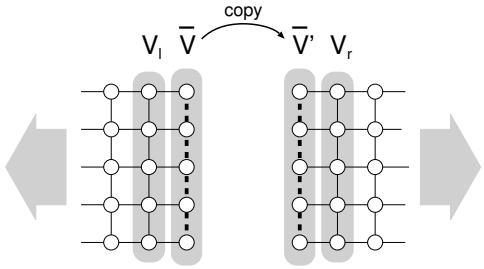
Within the corridor graph, searching for a circular path can be described easily as searching for a path from the "left side" $\bar{V}$ to the right side $\bar{V}'$ with the additional constraint that a path starting at v_i must end in v'_i. Even though a general planar graph can be transformed into a corridor graph by cutting along any arbitrary cut-path, care must be taken because the shortest circular-path algorithms operating on the corridor graph can only find paths that do not cross the cut-path more than once.



(a) The circular input graph is cut into a lane-shaped corridor graph.



(b) The nodes on the cut and the left and right neighbors.



(c) The nodes on the cut are duplicated and the graph is cut.

Figure 2: The input graph is cut along a cut-path to get a corridor graph.

The computation of a suitable cut-path is included in the main description of our algorithm.

3 Preliminary considerations about shortest paths

In the construction of our algorithm, we repeatedly make use of the following theorem about shortest paths.

Theorem: Let $G = (V, E)$ with $V = \{v_i\}_i$ be a graph and let $u = v_{i_1} v_{i_2} \dots v_{i_n}$ and $w = v_{k_1} v_{k_2} \dots v_{k_m}$ be two minimum-cost paths. Then we can state that if u and w have two nodes v_p and v_q in common, there is a path $w' = v_{k_1} \dots v_{k_m}$ with the same cost as w, which has a common subpath $s_u = v_p \rightsquigarrow v_q$ with path u.

Proof: If s_u and s_w have equal cost, there is nothing to show. Hence, assume that the two subpaths s_u, s_w between v_p and v_q have different cost. Then, the cost of either s_u or s_w must be lower than the other. Let us assume that the cost of s_u is lower than the cost of s_w. But then w cannot have minimum cost, because the cost can be lowered by replacing the subpath s_w with s_u. □

As a direct consequence of this theorem, we can state the following two corollaries.

Corollary 1: Minimum-cost paths cross at most once.[1]

[1]In fact, minimum-cost paths can cross more than once, but there is always a path of similar cost that does not cross more than once. Hence, when searching for a minimum-cost path, we can assume that they do not cross more than once.

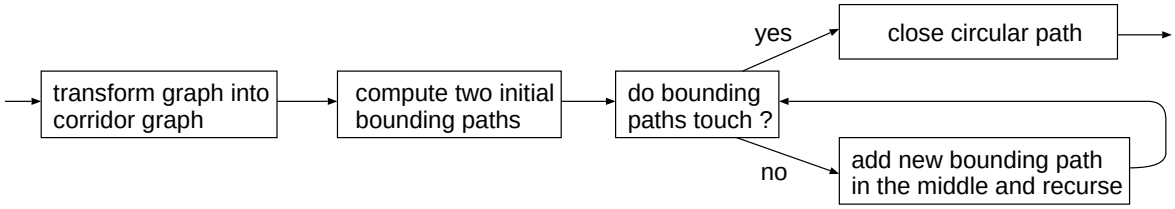


Figure 3: Overview of the shortest circular-path algorithm.

Corollary 2: If two paths share a common starting node, both paths will share a common subpath to their end node until both paths split. After the split, the paths will not cross.

Our algorithm also exploits one property of the Dijkstra algorithm, which is employed in our algorithm to compute ordinary shortest paths. We use the fact that the Dijkstra algorithm not only computes the shortest path between two nodes, but the full shortest-path tree to all nodes, rooted at the starting node.

4 Shortest Circular-Path Algorithm

Step 1: Transform graph into corridor graph

The first step of our algorithm (see also Fig. 3) is to cut the graph to create a corridor graph as introduced in Section 2. The important point is that it must be ensured that the path $\bar{p}$, along which the graph is cut open, does not cross the circular path more than once. This can be achieved by computing a shortest path between the nodes of the faces F_I and F_E. Because of Corollary 1, this path will cross the circular path exactly once.

The problem of finding a shortest circular path is now reduced to finding the shortest path $v_i \rightsquigarrow v_i'$ between the set of nodes $\bar{V}$ and $\bar{V}'$. In the following, we draw $\bar{V}$ and $\bar{V}'$ on the left and right sides of the corridor graph (Fig. 4).

Step 2: Compute two initial bounding paths

In the next step, the shortest path between v_1 and v_1', labelled (a), and the shortest path between v_N and v_N', path (b), is computed, where $N = |\bar{V}| = |\bar{V}'|$. Often, these two paths (a) and (b) will touch like shown in Fig. 4, e.g., because the edges in the center of the graph have a low cost.

Step 3/Case 1: Bounding paths do touch

For the case that the two bounding paths touch, the shortest circular path (c) can be computed immediately, because it is known that (c) also has the common subpath $v_l \rightsquigarrow v_r$. Hence, the shortest circular path can be obtained by computing the ordinary shortest path between v_r and v_l, within the area between (a) and (b), as indicated by the dashed line in Fig. 4. Note that the search for $v_r \rightsquigarrow v_l$ can be implemented more efficiently by searching backwards from $v_l \rightsquigarrow v_r$. In this case, the search can be restricted to the left area, since the shortest-path tree for the right area, rooted at v_r, is already known from the previous search.

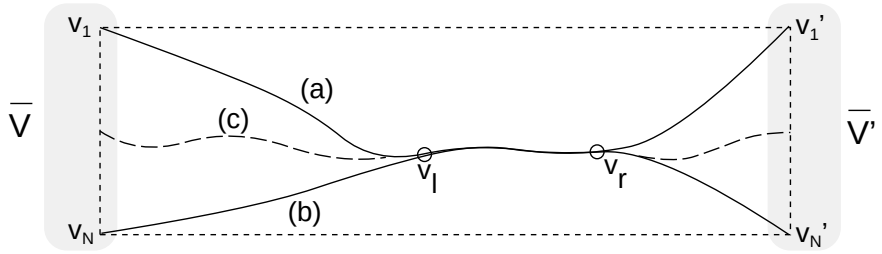


Figure 4: (Step 3/Case 1) The initial shortest paths (a) and (b) have a common subpath $v_l \rightsquigarrow v_r$. In this case, the shortest circular path (c) can be computed directly in one more step by connecting v_r with v_l through a shortest path (dashed line).

Step 3/Case 2: Bounding paths do not touch

Case 2 uses a recursive approach to continuously split the graph into smaller graphs until the shortest circular-path problem can be solved similarly as in Case 1.

The input of each recursion is an input graph which is bounded to the top and bottom by shortest paths (a) and (b). Let us denote the top bounding path (a) as $v_a \rightsquigarrow v'_a$ and the bottom bounding path (b) as $v_b \rightsquigarrow v'_b$. Note that both paths are circular paths and that initially $v_a = v_1$ and $v_b = v_N$. [2] We now compute a shortest-path tree, starting from the middle node $v_c = v_{(a+b)/2}$ at the left side of the graph (Fig. 5). In one run of the Dijkstra algorithm, we obtain all shortest paths to the nodes between v'_a and v'_b at the right side. Note that we can limit the computation to the area between the two bounding paths (a),(b).

If we consider the shortest paths between v_c and the right side of the graph, we see that the path $v_c \rightsquigarrow v'_a$ obviously joins with (a). As we consider destination nodes v'_k further down ($k > a$), there is generally a point from which onwards the shortest path does not join (a) anymore. In fact, there is a node v'_A, $A < c$ which is the last node (from top to bottom) for which the shortest path $v_c \rightsquigarrow v'_A$ touches (a). Let us denote this path as (d), like it is depicted in Fig. 5. Note that this results in a similar situation as in Case 1. Any circular path $v_i \rightsquigarrow v'_i$ with $a \leq i \leq A$ is known to include the common subpath of (a) and (d). This range is identified as $\mathbf{A} = [a; A]$. We can now compute the shortest circular path that starts and ends within range $\mathbf{A}$ in a single step, similarly to Case 1 (see Fig. 6).

A symmetrical process can be carried out to find the shortest circular path for the area $\mathbf{B}$ at the bottom of the considered graph. When the shortest circular path in each of the ranges $\mathbf{A}$ and $\mathbf{B}$ are known, we have to further consider only the remaining range in between (Fig. 7). Since we also know the path $v_c \rightsquigarrow v'_c$, denoted as (c), we can split the problem for the remaining range recursively, by first considering the graph between (f) and (c) with start nodes $v_{A+1}, \ldots, v_{c-1}$, and similarly the graph between (c) and (g) with start nodes $v_{c+1}, \ldots, v_{B-1}$. Once all shortest circular paths for all ranges of nodes along the cut are known, we simply select the shortest of them as the global solution.

[2] In later recursion steps, v_a and v_b change to the new bounding paths (for example to path (f) and (c) during recursion 1 or (c) and (g) during recursion 2 in Figure 7).

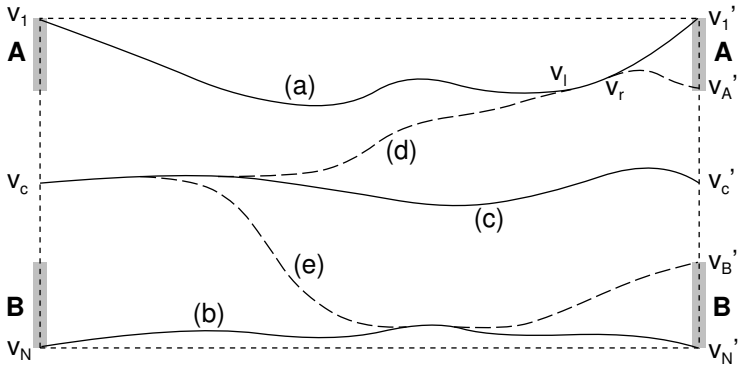


Figure 5: A shortest-path tree is computed, starting at v_c, which is the node in the middle of the left side. The bottom-most node v_i' on the right side, for which $v_c \rightsquigarrow v_i'$ joins the path (a) is denoted as v_A' and the corresponding path as (d). Similarly, we obtain v_B' as the top-most node on the right side, for which $v_c \rightsquigarrow v_B'$ (e) still joins the path (b). Furthermore, we have computed the circular path $v_c \rightsquigarrow v_c'$ for the center node v_c.

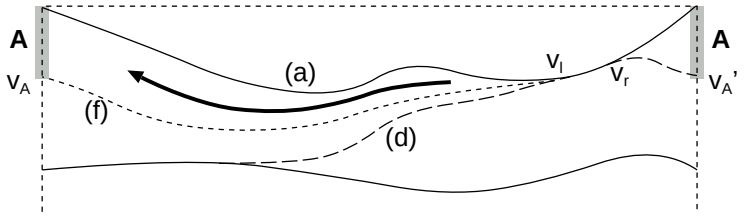


Figure 6: Given (a) and (d), we know that all shortest circular paths in range **A** include the subpath $v_l \rightsquigarrow v_r$. Hence, we search for a shortest path starting from v_l in the indicated direction to v_r. This is the shortest circular path within range **A**. Note that this search can be restricted to the area between (a) and (d). The path (f) is the shortest circular path $v_A \rightsquigarrow v_A'$, which will be used as upper-bound path to further reduce the graph size in the recursion step.

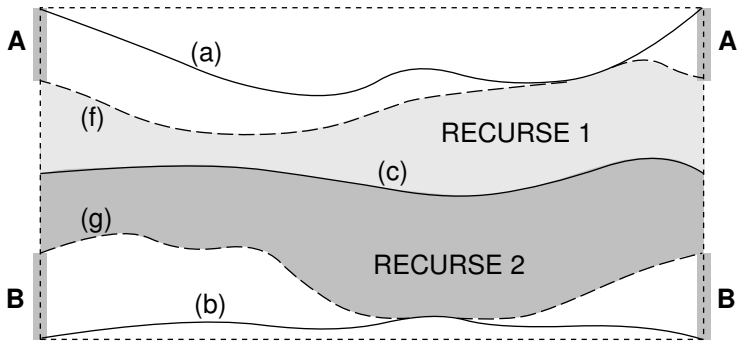


Figure 7: Regions **A** and **B** are already processed, only shortest circular paths in the range in between are unknown. This range is processed recursively, first processing the graph between (f) and (c), and then between (c) and (g).

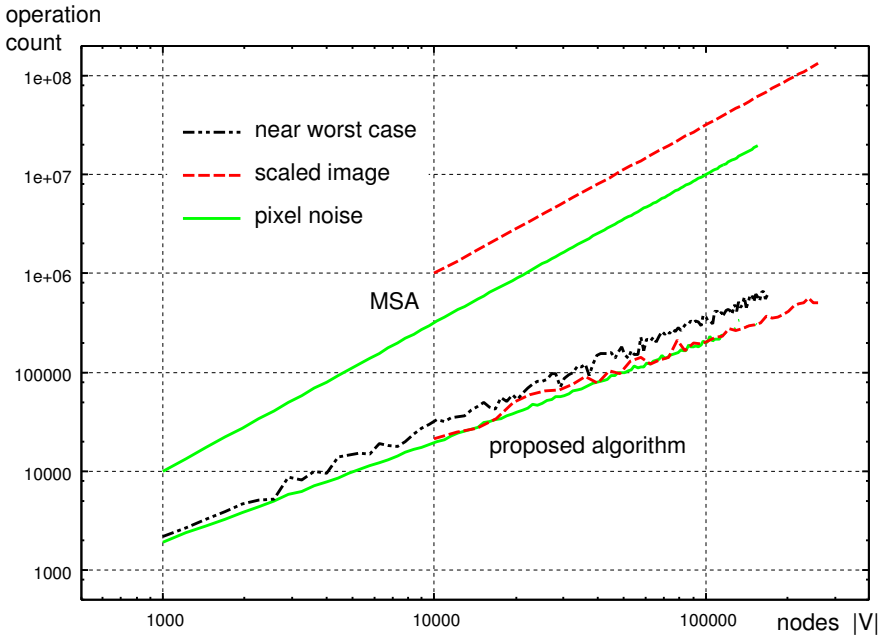


Figure 8: Computation times for input graphs for different types of complicated data. For these complex examples, the computation time of MSA is $O(|V|^{3/2})$, whereas it is only $O(|V|)$ for the proposed algorithm.

5 Computational complexity

In most practical cases, the problem can be processed with Step 3/Case 1, such that only three ordinary shortest-path searches have to be conducted. Using the Dijkstra algorithm with a heap implementation, each search takes $O(|V| \log |V|)$ time, since the graph is planar. Hence, the total running time is also just $O(|V| \log |V|)$.

The computation time for Step 3/Case 2 is more complicated to determine, because it is directly depending on the input data. In the worst-case, every shortest circular path crossing the cut is independent and must be computed. However, the size of the graph in the recursion is only about half the size of the input graph. This gives a total number of nodes to be processed of approximately

$$\sum_{i=0}^{\log |\bar{V}|} 2^i \frac{|V| \log |V|}{2^i} = |V| \cdot \log |V| \cdot (\log |\bar{V}| + 1). \tag{2}$$

Furthermore, nodes from the ranges **A** and **B** can be excluded, so that the actual computation time is usually lower. Note also that the cut-path is determined such that it is short, which makes $|\bar{V}|$ small.

A special, but important case is formed by trellis-shaped graphs with directed edges. For this type of graph, the shortest-path search using the Dijkstra algorithm can be replaced by a more simple dynamic programming approach, which processes the nodes column by column. This reduces the computation time for an ordinary shortest-path search to $O(|V|)$ instead of $O(|V| \log |V|)$ for the Dijkstra algorithm on general planar graphs. In total, we then obtain a practical computation time of only $O(|V|)$ for the shortest circular-path search and a worst case of $O(|V| \log |V|)$.

Experimental verification

To justify the estimation of computation time, we conducted experiments to compute the shortest circular path through a rectangular image, folded to a cylinder. In the first experiment, we took a square-shaped natural image with costs taken from the pixel luminance. The shortest circular path horizontally crossing the image was computed for different sizes of the input image. In the second experiment, we used rectangular grid graphs that are 10 times longer than wide with edge weights set to random values. In a final experiment, we synthesized an input with alternating high-cost and low-cost rows, which results in a problem close to the worst-case. In all experiments, we used directed edges arranged in a regular way, such that a fast shortest-path search with dynamic programming can be applied.

The measured computation times for both algorithms are depicted in Figure 8. It is clearly visible that the MSA algorithm has a complexity of $O(|V|^{3/2})$, while the proposed algorithm usually runs in only $O(|V|)$ time. A computation time that comes close to the worst case could only be reached with the synthetically constructed input.

6 Conclusions

This paper proposed a new algorithm for computing shortest circular paths in planar graphs. In practical cases, the described algorithm has the same computational complexity as the ordinary shortest-path algorithm employed within the shortest circular path algorithm. Hence, when using the Dijkstra algorithm, a practical computation time of $O(|V| \log |V|)$ is achieved, while $O(|V|)$ can be achieved using dynamic programming or specialized algorithms for shortest paths in planar graphs [3]. The worst-case performance is a factor $\log |V|$ larger than in the practically occuring case, but even for difficult inputs, the computation time is close to the typical case.

References

[1] B. Appleton and C. Sun. Circular shortest paths by branch and bound. *Pattern Recognition*, 36(11):2513–2520, November 2003.

[2] Dirk Farin, Magnus Pfeffer, Peter H. N. de With, and Wolfgang Effelsberg. Corridor scissors: A semi-automatic segmentation tool employing minimum-cost circular paths. In *International Conference on Image Processing (ICIP)*, pages 1177–1180, October 2004.

[3] Philip Klein, Satish Rao, Monika Rauch, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 27–37, New York, NY, USA, 1994. ACM Press.

[4] Herbert Suesse and Wolfgang Ortmann. Robust matching of affinely transformed objects. In *IEEE International Conference on Image Processing (ICIP)*, volume 2, pages 375–378, September 2003.

[5] C. Sun and S. Pallottino. Circular shortest path in images. *Pattern Recognition*, 36(3):711–721, March 2003.