Estimating the Maximum Hidden Vertex Set in Polygons

Antonio L. Bajuelos *
Department of Mathematics & CEOC
University of Aveiro
3810-193 Aveiro, Portugal
leslie@ua.pt

Santiago Canales †
Escuela Técnica Superior de Ingeniería, ICAI
Universidad Pontificia Comillas de Madrid
C/ Alberto Aguilera 21 Despacho 208
28015 Madrid, Spain
scanales@dmc.icai.upcomillas.es

Gregorio Hernández †
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo, Boadilla del Monte
28660 Madrid, Spain
gregorio@fi.upm.es

A. Mafalda Martins ‡
Department of Mathematics & CEOC
University of Aveiro
3810-193 Aveiro, Portugal
mafalda.martins@ua.pt

Abstract

It is known that the MAXIMUM HIDDEN VERTEX SET *problem on a given simple polygon is NP-hard [11], therefore we focused on the development of approximation algorithms to tackle it. We propose four strategies to solve this problem, the first two (based on greedy constructive search) are designed specifically to solve it, and the other two are based on the general metaheuristics Simulated Annealing and Genetic Algorithms. We conclude, through experimentation, that our best approximate algorithm is the one based on the Simulated Annealing metaheuristic. The solutions obtained with it are very satisfactory in the sense that they are always close to optimal (with an approximation ratio of* 1.7*, for arbitrary polygons; and with an approximation ratio of* 1.5*, for orthogonal polygons). We, also, conclude, that on average the maximum number of hidden vertices in a simple polygon (arbitrary or orthogonal) with* n *vertices is* $\frac{n}{4}$.

1. Introduction

While the traditional *art gallery problems* deal with setting the minimum number of guards (or lights) in a given *simple polygon* (simple closed polygonal curve with its interior) P, such that each point in P is seen by at least one guard [13], in this paper we deal with the inverse problems: find the maximum number of points in a given polygon, such that no two of these points see each other. This class of problems has many applications, for example in computer-games, where a player needs to find and collect or destroy as many objects as possible. Not seeing the next object while collecting an object makes the game more interesting. In the visibility problems field, these problems are known as *hiding problems* [11]. Of these, we are going to study the MAXIMUM HIDDEN VERTEX SET (MHVS) problem. This problem asks for a set S of maximum cardinality of vertices of a given simple polygon, such that no two vertices in S see each other. Given two points x and y in a simple polygon P, we say that x sees y (or y is visible to x) iff the closed segment $[xy]$ does not intersect the exterior of P. The MHVS problem is a NP-hard combinatorial problem both for arbitrary and orthogonal polygons [11]. Orthogonal polygons are interesting for they may be seen as abstractions of art galleries, for instance. In [4], Eidenbenz has proven that, this problem is, also, APX-hard.

Our contribution: Since the MHVS problem is NP-hard we propose four approximation techniques for computing an approximate solution, i.e, a large hidden vertex set for a given simple polygon P (arbitrary and orthogonal). The first two are designed specifically to solve the MHVS problem, and the other two are based on the general metaheuristics Simulated Annealing and Genetic Algorithms. We also realize a comparative study of the results obtained by them,

*Supported by CEOC through *Programa* POCTI, FACT, co-financed by EC fund FEDER

†Supported by grant MEC-HP2005-0137

‡Supported by CEOC through *Programa* POCTI, FCT, co-financed by EC fund FEDER and supported by a FCT fellowship, grant SFRH/BD/19138/2004

using for it two random polygon generators, one to generate random arbitrary simple polygons [8] and another to generate random orthogonal polygons [10]. After this comparative study, we make a statistical analysis to choose the best technique concerning the cardinality of the hidden vertex set obtained, $|H|$. As the optimal solution of the MHVS problem (i.e., the maximum number of vertices that is possible to hide in a polygon) is unknown, we have a method, as in [2], that allows us to determine an *upper bound* for it. In this way, we are able to determine the approximation ratio of our best approximation technique. Our implementation has been developed using the CGAL $2D$ Regularized Boolean Set-Operations package [1, 6] and our experiments are conducted on a large set of randomly generated simple polygons.

2. Preliminaries and some useful results

Let P be a simple polygon with n vertices, $v_0, v_1, \ldots, v_{n-1}$. As in this paper we only deal with simple polygons, we use the term polygon to refer to a simple polygon . We assume, without loss of generality, that the vertices of P are ordered in a counterclockwise direction around the interior of P. Given a set of vertices of P we say that H is a *hidden vertex set* for P if no two vertices in H see each other. As stated before, the MHVS problem on a polygon asks for a hidden vertex set, H, of maximum cardinality. In [11] Shermer has proven that for arbitrary and orthogonal polygons, with n vertices, the size of a maximum hidden vertex set can be as large as, but not exceed, $\lceil \frac{n}{2} \rceil$ and $\frac{n-2}{2}$, respectively. These tight bounds are achieved in *triangular saw* polygons (see Figure 1(a)) and in *staircase* polygons (see Figure 1(b)), respectively.

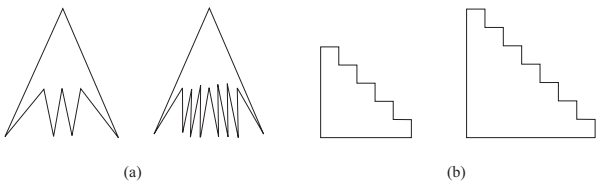


Figure 1. (a) *Triangular saw* **polygons; (b)** *Staircase* **polygons.**

Given that the MHVS problem is NP-hard, we developed approximation algorithms to tackle it. In approximation algorithms the guarantee of finding optimal solutions is sacrificed in exchange for getting good solutions in a reasonable computational time. However, we do not know the optimal solution for the MHVS problem, so we can ask: How can we expect to prove that our approximate solutions are near it? To try to answer this question, we developed a heuristic to determinate a minimum clique partition of the visibility graph of a given polygon, which gives us a method to compute an upper bound on the optimal number of hidden vertices for each instance in our experiments. Then the application of the approximation algorithms together with the heuristic to determine the upper bounds, to each instance in our experiments, gives us the performance ratio of our approximation algorithms.

The *visibility graph* of a polygon P is defined as [12]: $VG(P) = (V, E)$, where $V = \{v \mid v \text{ is a vertex of } P\}$ and $E = \{(u, v) \mid \text{the vertices } u \text{ and } v \text{ are visible in } P\}$. Figure 2 illustrates a polygon P and its visibility graph.

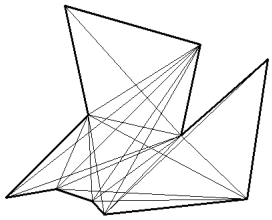


Figure 2. A polygon P, with $n = 10$, and $VG(P)$.

We say that a set C is a *clique partition* of $VG(P)$ if its elements are disjoint subsets V_i of V, where each vertex in V_i sees all vertices in V_i. Figure 3 illustrates a polygon P and a clique partition of $VG(P)$.

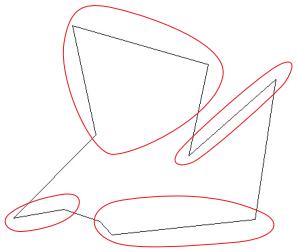


Figure 3. A clique partition (with 4 cliques) of $VG(P)$.

It is easy to see that, for each element of a C we can hide at most one vertex in P, so $|C| \geq |H|$, $\forall C, H$. Consequently, the number of hidden vertices in a maximum-cardinality of a hidden vertex set of P is *at most* the number of cliques in a minimum-cardinality clique partition of $VG(P)$. Thus, the number of cliques in a minimum-cardinality clique partition of $VG(P)$ is an upper bound on the optimal number of hidden vertices in P. But, the problem of determining this upper bound (MINIMUM CLIQUE

PARTITION problem) is NP-hard [5], so we developed, as stated above, an approximation algorithm to solve it.

In our experiments the main objective is to find large hidden vertex sets, H, and small clique partitions, C. The set H that we obtain approximates the optimal number of hidden vertices with an approximation ratio of $\frac{|C|}{|H|}$.

3. Approximation Algorithms

We developed four approximation algorithms to determine H of maximum cardinality in a given polygon P. The first two, designated by A_1 and A_2, are greedy constructive algorithms. The other two, called M_1 and M_2, are based on the general metaheuristics Simulated Annealing (SA) and Genetic Algorithm (GA), respectively. Relatively to the MINIMUM CLIQUE PARTITION problem, we developed a greedy/sequencial constructive algorithm to determine a clique partition C with a small number of cliques, which we designate by A_3. In the next sections we are going to describe these algorithms.

3.1. Greedy Strategies

A natural approach to find H is to do so in a greedy way: add hidden vertices one by one until H is achieved, selecting at each step a hidden vertex from the set of vertices of P, according to some rule. In this way, we apply two rules to select the hidden vertices, consequently we have two different greedy algorithms: A_1 and A_2.

Algorithms A_1 and A_2. The first rule is based in the *hidden region* concept. Being v_i a vertex of P, we denote by $V(v_i)$ the visibility polygon of v_i, i.e., the set of all points of P seen by v_i. We call *set of hidden regions of* v_i the set $HR_i = P \backslash V(v_i) = \{HR_i^1, \ldots, HR_i^k\}$, which is formed by all sub-polygons (*regions*) of P whose points are not seen by v_i (see, for example, Figure 4).

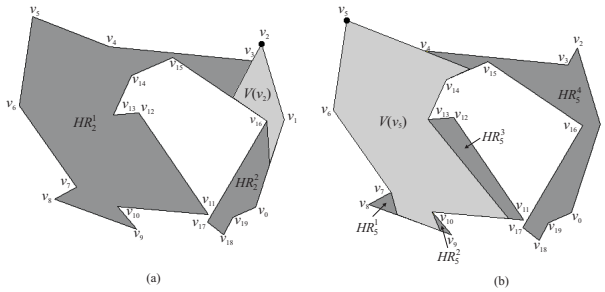


Figure 4. P with $n = 20$ and (a) $V(v_2)$ and $HR_2 = \{HR_2^1, HR_2^2\}$; (b) $V(v_5)$ and $HR_5 = \{HR_5^1, HR_5^2, HR_5^3, HR_5^4\}$.

Based on some experiments, we observe that, in most cases, the convex vertices (a vertex is called convex if its interior angle is lesser then π) are those the that has more hidden regions. Therefore, we select convex vertices one by one, according to the cardinality of *HR* and the area of its visibility polygon. In this way we build H (see Algorithm 1 for a complete description of this method).

Algorithm 1 Determining H from the hidden regions (method A_1)

Input: A polygon P with n vertices and $VG(P)$
Output: A hidden vertex set, H

1. $H \leftarrow \emptyset$
2. $V_{conv} \leftarrow \{v_i \mid v_i \text{ is convex}\}$
3. **for** each $v_i \in V_{conv}$ **do**
4. determine $V(v_i)$ and $|HR_i|$
5. **end for**
6. **while** $V_{conv} \neq \emptyset$ **do**
7. Choose v_i in V_{conv} with more hidden regions; and in the event of a tie choose the one whose $V(v_i)$'s area is smaller
8. $H \leftarrow H \cup \{v_i\}$
9. Delete v_i (and all vertices seen by v_i) from V_{conv}
10. **end while**
11. **return** H

Remark: the calculating of the $VG(P)$ is made according to the algorithm described on [7], that is based on computing the visibility polygon for each vertex of P. This calculation is made as a pre-processing.

The second rule is based in the number of vertices seen by one vertex. Thus, the algorithm A_2 is similar to A_1. The main differences are that we consider the list of all vertices of P (step 2.) and instead of selecting the convex vertex that has more hidden regions (step 5.), we choose the vertex that sees less vertices, until we have H.

3.2 Metaheuristics

As stated before we developed two techniques based on the general metaheuristics Simulated Annealing and Genetic Algorithms: M_1 and M_2. For a comprehensive survey on metaheuristics see, e.g., [3].

3.2.1 Simulated Annealing Metaheuristic

The Simulated Annealing (SA) metaheuristic tries to minimize the limitation of the *local search algorithms*, which stops as soon as they find a local extreme. For that, it is allowed to accept solutions of worse quality than the current solution with a certain *probability*. This probability is

dependent of a parameter called *temperature*, T, which decreases over the algorithm iterations according to a *decrement rule*.

To solve an optimization problem with the SA metaheuristic it is necessary to specify the following aspects:

1. **Specific Parameters** (of the problem)
   - *Solution Space (set S)*;
   - *Cost or Objective Function (f)*;
   - *Neighborhood of each solution*;
   - *Initial Solution*.
2. **Generic Parameters** (of the annealing strategy)
   - *Initial Temperature (T_0)*;
   - *Temperature decrement rule*;
   - *Number of Iterations in each temperature ($N(T)$)*;
   - *Termination Condition*.

Below, we describe the way these elements were adapted to our problem.

1. Specific Parameters

Solution space (set S). The solution space to our problem is the set of all H for P. Therefore, S is finite and we can represent it by $S = \{S_1, S_2, \ldots, S_m\}$, where $S_i = v_0^i v_1^i \ldots v_{n-1}^i$ for $i = 1, \ldots, m$. In this way, each element of S (i.e., each candidate solution), S_i, is represented by a chain with length n, where v_j^i, with $j \in \{0, \ldots, n-1\}$, represents the vertex $v_j \in P$ and can have the value 0 or 1. If $v_j^i = 1$ then the vertex v_j is marked as a hidden vertex; otherwise ($v_j^i = 0$) the vertex v_j is marked as not hidden. Besides, so that S_i is a valid solution, the vertices marked as hidden cannot see each other (see Figure 5, for an illustration).

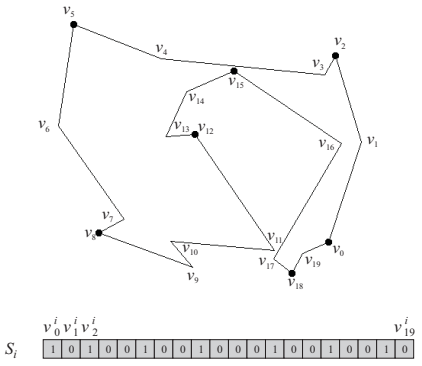


Figure 5. An element $S_i \in S$ (for a polygon with $n = 20$) and its representation. Black dots represent hidden vertices.

Objective function (f). The objective function $f : S \to \mathbb{N}$ assigns to each element of S a natural value. For each $S_i \in S$, $f(S_i)$ is the number of 1's in S_i, representing the hidden vertices.

Neighbourhood of each solution. According to the SA metaheuristic, for every $S_i \in S$, an element $S_{i+1} \in S$, called neighbour of S_i, must be obtained, which will be the element to analyze in the next iteration. In our case, given $S_i = v_0^i, \ldots, v_{n-1}^i$, we generate randomly a natural number $t \in [0, n-1]$ and then, if:

- $v_t^i = 1$ then we set $v_t^{i+1} = 0$ and we accept this new solution with some probability, since we are worsening the solution.
- $v_t^i = 0$ then we set $v_t^{i+1} = 1$. If this new solution is a valid solution then we accept it, since we are improving the solution; else we *validate* the obtained solution and accept it with some probability.
  - **solution's validation:** we mark all hidden vertices as not hidden if v_t sees them, in other words, if $v_j^{i+1} = 1$ and v_t sees v_j then we change the value of v_j^{i+1} value to 0.

  Note that, the validation always makes the solution worse.

Initial Solution. The initial solution is the first $S_i \in S$, with $i \in \{1, \ldots, m\}$ to be analyzed. In our case, it was considered $S_i = 10 \ldots 0$, i.e., the vertex v_0 is marked as hidden the remainder are labeled as not hidden.

2. Generic Parameters

Initial temperature (T_0). The literature advises that the SA metaheuristic must depart from a high initial temperature (T_0). Nevertheless, it does not seem suitable to consider, for T_0, fixed values independent from the problem. In this sense, it would be advisable to realize different analysis to choose T_0 , since its value may depend in large extent of the problem to solve. Following this idea, a comparative study has been realized taking into account two different types of T_0:

1. An initial temperature that is dependent on the number of vertices of the polygon P: $T_0 = f(n)$. In our comparative study we considered $T_0 = n$.
2. A constant initial temperature: $T_0 = 1000$.

Temperature decrement rule. The temperature decrement rule defines the value of T at each iteration k. Unfortunately, in spite of the existence of rules which guarantee the convergence to a global optimum they are not feasible in applications, because they are too slow for practical purposes. Thus, faster temperature decrement rules are adopted in applications. In this way, an analysis has been realized on three different types of rules that appear in the literature:

1. $T_{k+1} = \frac{T_0}{1+k}$ (FSA decrease);
2. $T_{k+1} = \frac{T_0}{e^k}$ (VFSA decrease);
3. $T_{k+1} = \alpha T_k$, where $0 < \alpha < 1$ (geometric decrease).

Number of iterations in each temperature ($N(T)$). The value of $N(T)$ must be large enough so that the system reaches its stationary state for each temperature T. In our algorithm the number of iterations for every temperature is equal to T, i.e., $N(T) = T$. This ensures that there are more iterations for high temperatures, which will be when the solutions are far from the optimum.

Termination Condition. Theoretically, the search should stop when a *frozen state* is achieved, i.e., when $T = 0$. Nevertheless, normally it is possible to finish with a final temperature, T_f, greater then zero without quality loss in the solution. In this sense, the termination condition chosen in our algorithm consists of finishing the search when the temperature is lesser than or equal to 0.005, i.e, $T_f \leq 0.005$. Clearly for lower temperatures the obtained solution will be closer to the optimum, but the response time of the algorithm increases considerably.

3.2.2 Genetic Algorithms Metaheuristic

Genetic Algorithms (GA) are technics that simulate the processes of the natural evolution (biological). To solve an optimization problem with the GA metaheuristic it is necessary to specify the following components:

- a genetic representation of the possible solutions, called *individuals* or *chromosomes*, to the problem (*Encoding*);
- a way of creating an initial population of possible solutions (*Initial Population*);
- a function to evaluate the individuals and make the effect of natural selection, sorting solutions according to their "strength" (*Objective or Fitness function*);
- genetic operators to alter the composition of the solutions (*Selection*, *Crossover* and *Mutation*);
- the values of various parameters used by the genetic algorithm (e.g., population's size, probability of the genetic operators, population's evaluation, population's generation, termination condition).

Next, we describe these components designed for our problem.

Encoding. In our algorithm an individual (or chromosome) I is represented by a chain of 0's and 1's, with length n, i.e., $I = g_0 g_1 \dots g_{n-1}$, where each element, g_i, is called a *gene*. Each gene represents a vertex of the polygon, i.e., the i^{th} gene represents the vertex $v_i \in P$. The value of each gene is 0 or 1. If the $g_i = 1$ then the vertex v_i is marked as a hidden vertex; otherwise ($g_i = 0$) the vertex v_i is marked as not hidden (see Figure 6).

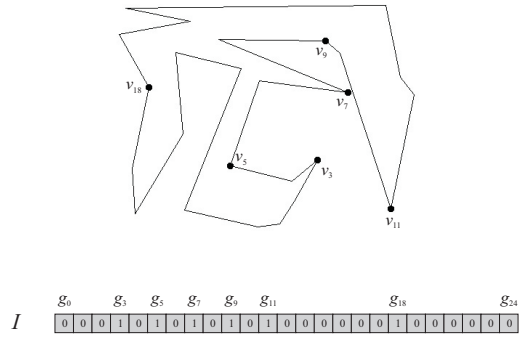


Figure 6. An individual, I, (for a polygon with $n = 25$) and its representation. Black dots represent hidden vertices.

Initial Population. The population of a given generation/iteration consists of a set of individuals. The total number of individuals in each population has to be large enough to ensure diversity, but not too much that damages the efficiency of the algorithm. In our case it has been taken as the population size the number of vertices of the polygon, linking in this way the entrance of the problem with the elements of the metaheuristic. Thus, the population for the generation t in our algorithm is represented by: $P(t) = \{I_0^t, I_1^t, \dots, I_{n-1}^t\}$, where each I_i^t represents an individual belonging to the population $P(t)$ and n is the number of vertices of the polygon P.

Remember that, an individual represents a possible solution for our problem, i.e, each individual must be a hidden vertex set. Thus, in our algorithm, to create the initial population, $P(0)$, we generate each of the n individuals in the following way: $\forall i \in \{0, \dots, n-1\}$, we mark as hidden the vertex v_i and all the vertices in P that form with v_i a hidden vertex set. This algorithm is illustrated below.

Algorithm 2 Generation of I_i^0, with $i \in \{0, \ldots, n-1\}$

1. $g_i \leftarrow 1$ and $g_j \leftarrow 0, \forall j \neq i$ (thus, $H = \{v_i\}$)
2. **for** $j = 0$ to $n-1$ **do**
3. **if** $v_j \cup H$ is a hidden vertex set **then**
4. $g_j \leftarrow 1$
5. **end if**
6. **end for**

For example, in Table 1 it is illustrated the initial population, $P(0) = \{I_0^0, I_1^0, \ldots, I_{n-1}^0\}$, of the polygon exemplified in Figure 7.

$I_0^0 = 1000000000$	$I_5^0 = 0010010100$
$I_1^0 = 0100000010$	$I_6^0 = 0010101001$
$I_2^0 = 0010101001$	$I_7^0 = 0010100100$
$I_3^0 = 0001000001$	$I_8^0 = 0100000010$
$I_4^0 = 0010101001$	$I_9^0 = 0100000001$

Table 1. Individuals of $P(0)$

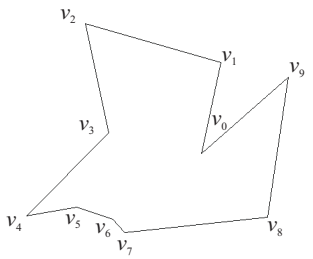


Figure 7. Polygon with $n = 10$.

Objective or Fitness Function. This function should help us to make the best selection of individuals to be reproduced, so that it will assign higher values to the solutions closer to the optimal one. In our case, given an individual I, the fitness function, $f(I)$, returns the number of 1's that exists in the chain that represents it.

Selection. The selection method should choose the best individuals to be reproduced. While there are many different types of selection, we use the most common: the *roulette wheel selection*. In this selection the individuals are given a probability of being selected that is directly proporcional to their fitness. Two individuals are then chosen randomly based on these probabilities to be parents in crossover. In our case we use this method to choose the two best individuals to be parents in crossover.

Crossover. Crossover operates on selected genes from parent individuals and creates new individuals (children). While there are many different kinds of crossover, we use the *single point crossover*, to generate *one child*. In this type of crossover, a randomly selected point (gene) on the two parents is chosen, then the parents are divided at this crossover point, and, finally, a child is created by exchanging tails (see Figure 8).

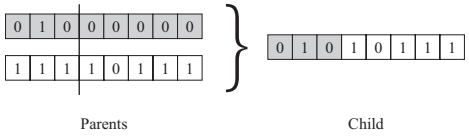


Figure 8. Single point crossover.

Crossover does not always occur, it occurs with a given probability, p_c. The value of p_c is decided on the basis of trial and error, however, p_c is generally between 70% and 95%. We use $p_c = 0.9$. Note that, the child resulting from this crossover may not be valid (i.e., it may not correspond to a hidden vertex set). Thus, we must make it valid. For that, we fix the best tail, which is the tail that has more 1's, and we validate another tail as follows. Suppose that it has m genes. We randomly generate m, different, natural numbers, $t_0, t_1, \ldots, t_{m-1}$ $(0 \leq t_i \leq m-1)$. Then, we have two cases: (1) the worst tail is the first one (2) the worst tail is the second one. In case

(1) For each t_i, if $g_{t_i} = 1$ and v_{t_i} is not seen by any vertex represented in the second tail, then the value 1 is maintained, otherwise we change its value to 0.

(2) For each t_i, if $g_{t+t_i+1} = 1$ and v_{t+t_i+1} is not seen by any vertex represented in the first tail, then the value 1 is maintained, otherwise we change its value to 0.

Figure 9 exemplifies case (2).

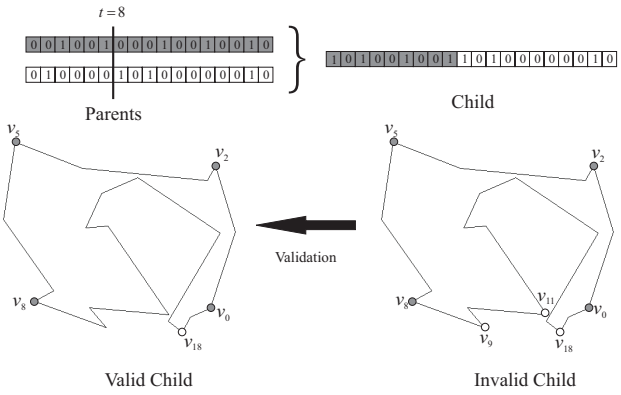


Figure 9. Single point crossover and validation.

In these example, we can see that the generated child is not valid, since v_9 and v_{11} are seen by v_8. The worst

tail is the second one and it has $m = 11$ elements, so the following 11 numbers are randomly generated $t_0 = 3$, $t_1 = 5$, $t_2 = 7$, $t_3 = 8$, $t_4 = 6$, $t_5 = 1$, $t_6 = 0$, $t_7 = 9$, $t_8 = 4$, $t_9 = 2$ and $t_{10} = 10$. Since $g_9 = 1$ and v_9 is seen by the vertex v_8 its value is changed to 0; $g_{18} = 1$ and v_{18} is not seen by any vertex represented in the first tail, so its value remains 1; $g_{11} = 1$ and v_{11} is seen by the vertex v_8, so its value is changed to 0. Thus, the new child is now 10100100100000000010, which is a valid individual.

Mutation. In a binary representation, the action of mutation is relatively simple, it merely flips a randomly selected binary digit from zero to one or vice versa, as shown in Figure 10.

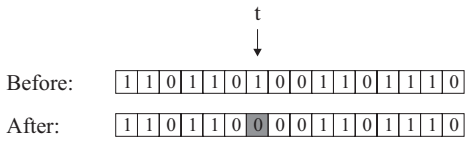


Figure 10. Mutation.

The probability of mutation, p_m, is decided on the basis of trial and error, however it is usually less than 1%. In our case we apply the mutation to the child obtained in the crossover operation, with $p_m = 0.05$, as follows: we generate randomly a natural number $t \in [0, n-1]$. If $g_t = 1$ then we change its value to 0; otherwise ($g_t = 0$), we change its value to 1 only if the resultant individual is valid (i.e., if he represents a H).

Population's Generation. To generate a new population we replace the worst individual of the population by the child obtained at the crossover.

Population's Evaluation. We consider the evaluation of a population, i.e., the fitness of a population, $F(P(t))$, as the maximum value of the objective function when applied to all individuals of the population, i.e., $F(P(t)) = \max\{f(I_0^t), \ldots, f(I_{n-1}^t)\}$.

Termination Condition. If in a sufficiently large number of generations the fitness has not changed, we can assume that we are close to optimal. Thus, we consider as the termination condition that the fitness of the population $F(P(t))$ remains unchanged for a number of generations h. In our case, has been considered $h = 5000$ (this value was chosen empirically).

3.3 Greedy/Sequencial Strategy for the MINIMUM CLIQUE PARTITION PROBLEM

As stated in section 2, we developed an approximated algorithm to determinate a minimum clique partition of the visibility graph of a given polygon, which gives us a method to compute an upper bound on the optimal number of hidden vertices for each instance in our experiments. The algorithm that we designed to tackle this problem is a greedy/sequencial constructive strategy, which we call A_3. In this strategy we first determine n clique partitions of $VG(P)$ from each of the vertices of P (see Algorithm 3). In the end, we choose as the solution the partition with fewer elements.

Algorithm 3 Algorithm to determine a clique partition from the vertex v_k

1. $C \leftarrow \emptyset$
2. $j = k$
3. **repeat**
4. Determine a clique from v_j, $C_j = \{v_j, v_{j+1}, \ldots, v_i\}$
5. $C \leftarrow C \cup C_j$
6. $j \leftarrow (i+1) \bmod n$
7. **until** $j \neq k$

4 Experiments and Results

We have implemented our approximation algorithms on a PC using the CGAL library (version 3.2.1). Our software works with Microsoft Windows XP with Microsoft C++ compiler in Visual Studio 2005. The tests were performed on a Windows XP PC with an Intel(R) Core(TM)2 CPU T5500 1.66GHz, 1.00 GB of RAM. We have performed extensive experiments with the strategies described in section 3. In this section we relate our results and conclusions from our experiments.

Our experiments were realized on a large set of randomly generated polygons, arbitrary and orthogonal. The arbitrary polygons were generated using the CGAL's function *random_polygon_2* [8], whose implementation is based on the method of eliminating self-intersections in a polygon by using the so-called "2-opt" moves method; and to generate orthogonal polygons we used the polygon generator developed by O'Rourke (for evaluation of [10]).

The following experiments were realized with four sets of arbitrary polygons, each one with 50 polygons of 50, 100, 150 and 200 vertex polygons, respectively. An analogous study is performed for orthogonal polygons and the conclusions are drawn in the end of the section 4.2.

4.1 Analysis of the SA's Parameters

According to section 3.2.1 we are going to analyze the choice of two parameters (T_0 and the function used to reduce the temperature) to find the combination of its values that fits best into our problem. The different combinations produce six cases (see Table 2).

Cases	
Case 1	$T_0 = n$ and $T_{k+1} = \frac{T_0}{1+k}$
Case 2	$T_0 = n$ and $T_{k+1} = \frac{T_0}{e^k}$
Case 3	$T_0 = n$ and $T_{k+1} = \alpha T_{k-1}$ ($\alpha = 0.9$)
Case 4	$T_0 = 1000$ and $T_{k+1} = \frac{T_0}{1+k}$
Case 5	$T_0 = 1000$ and $T_{k+1} = \frac{T_0}{e^k}$
Case 6	$T_0 = 1000$ and $T_{k+1} = \alpha T_{k-1}$ ($\alpha = 0.9$)

Table 2. The 6 different cases

We analyze these six cases by comparing the number of hidden vertices, the time spent and the number of iterations performed by each of the six cases. It should be noted that in all cases we impose a condition for the final temperature, $T_f \leq 0.005$. If we decrease this value the obtained solution is closer to the optimum, but the response times will increase because the number of evaluated candidate solutions is higher.

In Tables 3, 4 and 5 are exposed the obtained results by the first three cases. These tables present, as can be seen, the average time of pre-processing (PP) in seconds, the average number of hidden vertices, the average runtime in seconds and the average number of iterations of the algorithm.

Vertices	PP (sec.)	$\|H\|$	Time (sec.)	Iterations
50	0.46	13.96	0.06	9999
100	2.48	27.4	0.14	19999
150	7.14	40.5	0.22	29999
200	15.76	53.84	0.38	39999

Table 3. Case 1: $T_0 = n$**,** $T_{k+1} = \frac{T_0}{1+k}$

Vertices	PP (sec.)	$\|H\|$	Time (sec.)	Iterations
50	0.46	6.86	0	10
100	2.48	11.96	0	10
150	7.14	17.56	0	11
200	15.76	22.7	0	11

Table 4. Case 2: $T_0 = n$**,** $T_{k+1} = \frac{T_0}{e^k}$

Vertices	PP (sec.)	$\|H\|$	Time (sec.)	Iterations
50	0.46	9.92	0	88
100	2.48	16.54	0	94
150	7.14	22.2	0	98
200	15.76	28.04	0	101

Table 5. Case 3: $T_0 = n$**,** $T_{k+1} = \alpha T_{k-1}$ **(**$\alpha = 0.9$**)**

As we can see, in these first 3 cases the best solution, concerning the average number of hidden vertices, is obtained in Case 1. So, the best solution corresponds to the FSA temperature reduction (which is the slowest temperature reduction), with a larger number of iterations and a greater response time.

In the following three cases, we analyze how the different types of temperature decreasing behave, being T_0 constant, we choose $T_0 = 1000$. The obtained results are illustrated in Tables 6, 7 and 8. As we can see, in these 3 cases the best solution is obtained in Case 4. Thus, the best solution corresponds, also, to the FSA temperature reduction, with a larger number of iterations and a greater response time.

Vertices	PP (sec.)	$\|H\|$	Time (sec.)	Iterations
50	0.46	13.96	0.88	199999
100	2.48	27.4	1.16	199999
150	7.14	40.48	1.46	199999
200	15.76	53.86	1.68	199999

Table 6. Case 4: $T_0 = 1000$**,** $T_{k+1} = \frac{T_0}{1+k}$

Vertices	PP (sec.)	$\|H\|$	Time (sec.)	Iterations
50	0.46	6.8	0	13
100	2.48	12.44	0	13
150	7.14	17.58	0	13
200	15.76	22.84	0	13

Table 7. Case 5: $T_0 = 1000$**,** $T_{k+1} = \frac{T_0}{e^k}$

Vertices	PP (sec.)	$\|H\|$	Time (sec.)	Iterations
50	0.46	9.66	0.12	116
100	2.78	16.24	0.04	116
150	7.14	22.64	0.06	116
200	15.76	28.04	0.08	116

Table 8. Case 6: $T_0 = 1000$**,** $T_{k+1} = \alpha T_{k-1}$ **(**$\alpha = 0.9$**)**

Comparing the 6 cases, we can notice that the obtained results, concerning the average of $|H|$, are almost the same for the Cases 1 and 4 and for the Cases 3 and 6, that is, independently of the type of T_0, the average of $|H|$ are identical for FSA and geometric decreases. It is noticed, also, that, in general, the Case 5 is slightly better than Case 2.

As a general conclusion, we can say that despite of the FSA increase of the response time of the algorithm it improves the solution obtained. So, if we are looking for a solution closer to the optimum, it is more suitable to choose a slow decreasing of the temperature, the election of the initial temperature is not influential. Note that the best solutions are those obtained in the cases 1 and 4, they are almost

equal. However, the response time and the number of iterations are higher in the case 4. As, we want have to find a compromise between the goodness of the solution obtained and the algorithm's runtime. It seems that the case 1, is the one that better answers to these conditions. Therefore, this will be the case that we will use to realize the comparison with the rest of the designed algorithms.

We must remember that we have taken $T_f \leq 0.005$. Clearly, if this temperature is lowered the solutions obtained by our heuristics improve, and in more extent those cases that are further to find the optimal, i.e., for faster temperature decreases: VFSA and geometric. Therefore, if an acceptable and rapid solution is wished it will be possible to obtain reducing T_f and using a rapid decrease.

It is important to note that all alternatives with respect to parameters of the SA metaheuristic that could be explored is almost infinite. We have attempted in this work to find general references to these parameters, noting that a more exhaustive study in future investigations might improve the obtained results.

4.2 Comparison of the four strategies

In this section we are going to analyze and to compare the results obtained with the four approximation techniques: A_1, A_2, M_1 (Case 1) and M_2. Tables 9, 10, 11 and 12 present the obtained results by the four strategies. These tables present, as can be seen, the average time of preprocessing (PP) in seconds, the average number of hidden vertices, the average runtime in seconds and the average number of iterations of the algorithm.

Vertices	PP (sec.)	$\|H\|$	Time (sec.)	Iterations
50	0.48	13.96	0.04	9999
100	2.42	27.4	0.1	19999
150	7.2	40.5	0.26	29999
200	15.58	53.86	0.36	39999

Table 9. Results obtained with M_1 (Case 1)

Vertices	PP (sec.)	$\|H\|$	Time (sec.)	Iterations
50	0.48	12.1	0.08	12.1
100	2.42	24.18	0.46	24.18
150	7.2	35.12	0.5	35.12
200	15.58	46.62	0.9	46.62

Table 10. Results obtained with A_1

Comparing the results obtained with M_1 and A_1 (Tables 9 and 10, respectively), we notice that the M_1 heuristic has, on average, much more iterations, but is faster and the average number of hidden vertices is superior.

Vertices	PP (sec.)	$\|H\|$	Time (sec.)	Iterations
50	0.48	13.58	0	13.58
100	2.42	27.12	0	27.12
150	7.2	39.88	0	39.88
200	15.58	52.68	0	52.68

Table 11. Results obtained with A_2

Contrasting, now, the data exposed in Table 11 with that exposed in Table 10, we can see that A_2 is faster and obtains better solutions. Relatively to M_1 (Table 9), we notice that although the average number of iterations of the M_1 algorithm is much higher and M_1 is slower than A_2, once more, the average of hidden vertices obtained with M_1 is higher.

Vertices	PP (sec.)	$\|H\|$	Time (sec.)	Iterations
50	0.48	13.54	0.06	5226.4
100	2.42	26.28	0.08	5680.0
150	7.2	38.3	0.26	6666.7
200	15.58	50.34	0.5	7160.2

Table 12. Results obtained with M_2

We can also conclude, from Tables 12 and 10, that the average number of iterations of the M_2 algorithm is very superior to the A_1, but it is faster and the solutions obtained are better. From Tables 12 and 11 we can end that the algorithm M_2 is slower than the algorithm A_2 and that accrue worst solutions. Finally, the data exposed in Table 12 with those exposed in Table 9, we notice that the M_2 has on average less iterations, but M_1 is slightly faster and obtains better solutions.

As a conclusion we can say that, M_1 seems to be the best technique, since the obtained average of hidden vertices is the best, and in spite that average of the number of iterations is the biggest, the only method that is faster than it, is the A_2. In terms of the obtained solution the best approximation technique is the M_1, the second best is the A_2, followed by M_2, and finally the worst is the A_1 (see Figure 11).

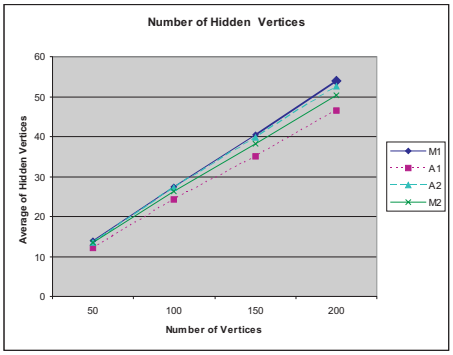


Figure 11. Solutions obtained by our strategies for arbitrary polygons.

However, the comparison between the results obtained by the different algorithms only makes sense, if they are significantly different, i.e., if we can ensure that the distributions of these results are different. To check this situation we apply the ANOVA test (using the Statistics Toolbox (Version 6.1) of the MATLAB [9]) to the results that has been used to obtain the averages exposed in the previous tables, concerning the average number of hidden vertices. We declare that a result is significantly different if the p-value is less than 0.05. The p-values returned by the ANOVA test are $9.88098e^{-14}$ for the data obtained with the polygons with $n = 50$ and 0 for the data obtained with the polygons with $n = 100, 150$ and 200. So we can conclude that at least one sample average is significantly different than the other sample averages, for any case. Then we performed a multiple comparison test to determine which pairs of averages are significantly different, and which are not (using the MATLAB's multcompare function). The answers provided by the realized tests are presented in the following tables, with the $+$ sign indicating whether the average is significantly different and $-$ if it is not.

Algorithm	M_1	A_1	A_2	M_2
M_1	•	+	-	-
A_1	+	•	+	+
A_2	-	+	•	-
M_2	-	+	-	•

Table 13. ANOVA Test for $n = 50$

Algorithm	M_1	A_1	A_2	M_2
M_1	•	+	-	+
A_1	+	•	+	+
A_2	-	+	•	+
M_2	+	+	+	•

Table 14. ANOVA Tests for $n = 100$, $n = 150$ and $n = 200$

As we can see, for $n = 50$, M_1 is not significantly different from A_2 and M_2; and A_2 is not significantly different from M_2. For $n = 100$, $n = 150$ and $n = 200$, A_2 and M_1 are not significantly different. So, from the multiple comparison tests realized we can conclude that A_2 and M_1 do not obtain significantly different results, in spite that we observe that M_1 gets better results in the average of the number of hidden vertices. The rest of the algorithms generate significantly different results. However, as we said before, a likewise study was made with orthogonal polygons. For those polygons the conclusions are similar, with the difference that the strategies M_1 and A_2 obtain significantly different results (concerning the average number of hidden vertices). So we proceed our study considering that the M_1 is the algorithm that obtains the best solutions.

Now, to conclude about the average of the maximum number of vertices that we can hide in an arbitrary polygon with n vertices, we applied M_1 (which is the one that obtains better solutions) to nine sets of arbitrary polygons, each one with 50 polygons of 20, 40, 60, 80, 100, 120, 140, 150 and 200 vertex polygons, respectively. The average of the obtained results, concerning $|H|$, are exposed in Table 15.

Vertices	20	40	60	80	100	120	140	150	200
$\|H\|$	5.7	10.98	16.3	21.58	26.88	32.08	37.2	39.7	53.22

Table 15.

Then, using the least squares method, we obtained the following linear adjustment (see Figure 12):

$$f(x) = 0.2667x + 0.6182 \approx \frac{x}{3.7} + 0.6182 \approx \frac{x}{4}$$

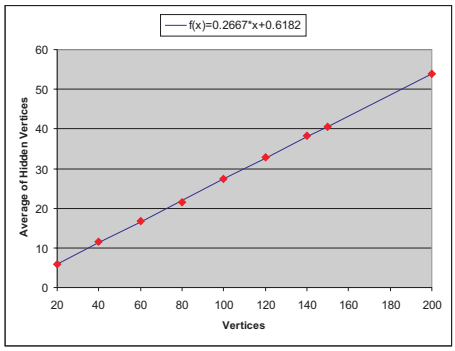


Figure 12. Least Squares Method.

Thus, we can conclude that on average the maximum number of hidden vertices in a arbitrary polygon P with n vertices is, at least, $\frac{n}{4}$. In order to get a quantitative measure on the quality of the calculated $|H|$, we computed the minimum clique partitions, to our instances (the nine sets of polygons described above). The ratio between the larger H (obtained with M_1) and minimum clique partition, C, obtained with A_3 (see section 3.3) never exceeded 1.7. That implies that our algorithm has an approximation ratio of 1.7.

A similar study was made with randomly generated orthogonal polygons, and we concluded that:

1. The best strategy is the M_1 (case 1), with significantly different solutions from the other methods. In Figure 13, that was obtained in similar way to the Figure 11, we can see that M_1 stands out among the other strategies, mainly for $n \geq 100$.

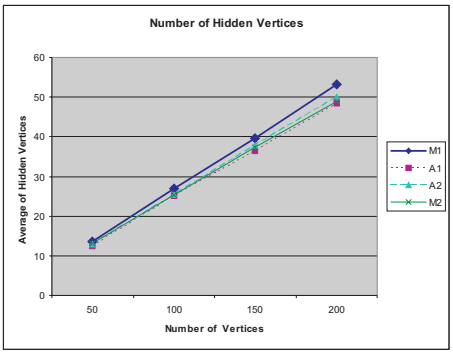
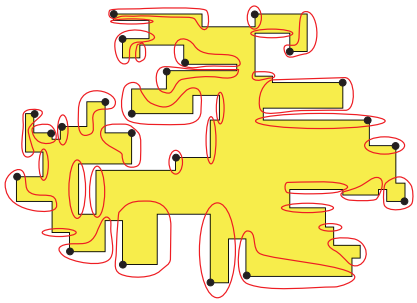
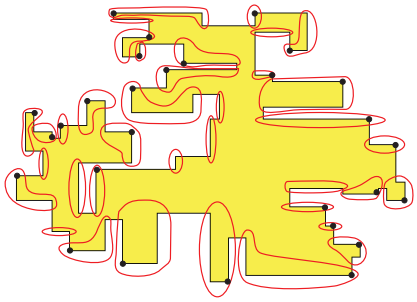


Figure 13. Solutions obtained by our strategies for orthogonal polygons.

2. On average the maximum number of hidden vertices in an orthogonal polygon with n vertices is, also $\frac{n}{4}$;
3. The approximation ratio is 1.5 , for all randomly generated instances



(a)



(b)

Figure 14. Example of the minimum clique partition (using the method described in section 3.3) of the orthogonal polygon with $n = 100$ vertices. The maximum H; (a) using A_1 (b) using M_1

Figure 14 shows one of the orthogonal polygons, with $n = 100$, that we tested, with our best and worst algorithms, M_1 and A_1, respectively. In this Figure, are illustrated the minimum clique partition obtained by A_3 (see section 3.3), $|C| = 37$, the solution obtained by A_1, $|H| = 23$ (Figure 14(a)) and the solution obtained by M_1, $|H| = 28$ (Figure 14(b)). The black dots represent the obtained hidden vertices.

Figure 15 shows one of the arbitrary polygons, with $n = 100$, that we tested, with our best and worst algorithms, M_1 and A_1, respectively. In this Figure, are illustrated the minimum clique partition obtained by A_3, $|C| = 35$, the solution obtained by A_1, $|H| = 23$ (Figure 15(a)) and the solution obtained by M_1, $|H| = 27$ (Figure 15(b)). The black dots represent the obtained hidden vertices.

(a)

(b)

Figure 15. Example of the minimum clique partition (using the method described in section 3.3) of the orthogonal polygon with $n = 100$ vertices. The maximum H; (a) using A_1 (b) using M_1

Figure 16 (a) shows a saw polygon, with $n = 20$, that we tested with M_1, and Figure 16 (b) shows a staircase polygon, with $n = 20$, that we tested with M_1. In this Figure, are illustrated the minimum clique partition obtained by

A_3, $|C| = 10$ and the solution obtained by M_1, $|H| = 10$ in both cases. Note that, in these examples, $\frac{|C|}{|H|} = 1$. The black dots represent the obtained hidden vertices.

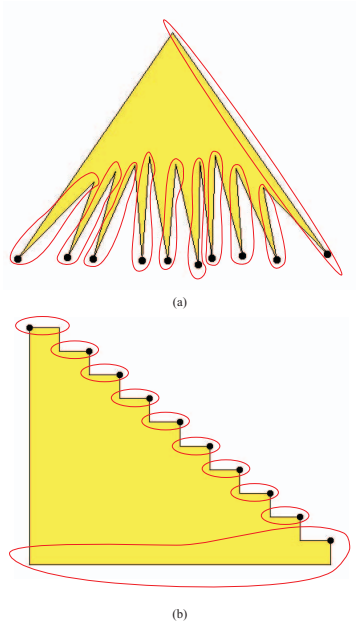


Figure 16. The C and H sets; (a) in saw polygon, with $n = 20$, obtained with A_3 and M_1 and (b) in staircase polygon, with $n = 20$, obtained with A_3 and M_1.

5 Conclusions and Future Work

We designed and implemented four approximation algorithms for solving the MAXIMUM HIDDEN VERTEX SET problem in polygons. The first two, A_1 and A_2, are greedy constructive strategies, and the other two, M_1 and M_2, are based on the general metaheuristics Simulated Annealing and Genetic Algorithms, respectively. We also give a method to compute the minimum clique partitions of the visibility graph of a polygon, allowing us to obtain provable bounds on how close our results are to optimal. From our statistical analysis of these approximation algorithms, we concluded that that the best one, concerning the number of hidden vertices, is the M_1. The hidden vertex sets obtained with it were very satisfactory in the sense that they were always close to optimal. We, also, concluded, that on average the maximum number of hidden vertices in a polygon (arbitrary or orthogonal) with n vertices is $\frac{n}{4}$.

Since the metaheuristics have proven to behave very well in solving this class of NP-hard problem, there are several directions for further research. We intend to use different parameterizations of the Genetic Algorithms metaheuristic (e.g., use different types of genetic operators), to adapt and implement other metaheuristics (e.g., the Ant Colony and Tabu Search metaheuristics) and to develop *hybrid metaheuristics* to solve, not only this problem, but also other NP-hard visibility problems.

References

[1] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

[2] Y. Amit, J. S. B. Mitchell, and E. Packer. Locating guards for visibility coverage of polygons. In *Proceedings of the Workshop on Algorithm Engineering and Experiments*, pages 1–15, 2007.

[3] C. Blum and R. Andrea. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, September 2003.

[4] S. Eidenbenz. *(In-)Approximability of Visibility Problems on Polygons and Terrains*. PhD thesis, Institute for Theoretical Computer Science, ETH, Zurich, 2000.

[5] S. Eidenbenz and C. Stamm. Maximum clique and minimum clique partition in visibility graphs. In *TCS '00: Proceedings of the International Conference IFIP on Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics*, pages 200–212, London, UK, 2000. Springer-Verlag.

[6] E. Fogel, R. Wein, B. Zukerman, and D. Halperin. 2d regularized boolean set-operations. In C. E. Board, editor, *CGAL User and Reference Manual*. 3.2.1 edition, 2006.

[7] S. Ghosh. *Visibility Algorithms in the Plane*. Cambridge University Press, New York, NY, USA, 2007.

[8] S. Hert, M. Hoffmann, L. Kettner, , and S. Schönherr. Geometric object generators. In C. E. Board, editor, *CGAL User and Reference Manual*. 3.2.1 edition, 2006.

[9] W. L. Martinez and A. R. Martinez. *Computational Statistics Handbook with MATLAB[R]*. Chapman & Hall/CRC, Boca Raton, London, New York, 2002.

[10] J. O'Rourke, I. Pashchenko, and G. Tewari. Partitioning orthogonal polygons into fat rectangles. In *CCCG*, pages 133–136, 2001.

[11] T. Shermer. Hiding people in polygons. *Computing*, 42(2-3):109–131, 1989.

[12] T. C. Shermer. Recent results in art galleries. In *Proceedings of the IEEE*, volume 80, pages 1384–1399, sep 1992.

[13] J. Urrutia. Art gallery and illumination problems. In J.-R. Sack and J. Urrutia, editors, *Handbook of computational geometry*, pages 973–1027. Elsevier, 2000.