Locating Guards for Visibility Coverage of Polygons*

Yoav Amit† Joseph S. B. Mitchell‡ Eli Packer§

Abstract

We propose heuristics for visibility coverage of a polygon with the fewest point guards. This optimal coverage problem, often called the "art gallery problem", is known to be NP-hard, so most recent research has focused on heuristics and approximation methods. We evaluate our heuristics through experimentation, comparing the upper bounds on the optimal guard number given by our methods with computed lower bounds based on heuristics for placing a large number of visibility-independent "witness points". We give experimental evidence that our heuristics perform well in practice, on a large suite of input data; often the heuristics give a provably optimal result, while in other cases there is only a small gap between the computed upper and lower bounds on the optimal guard number.

1 Introduction

The art gallery problem was introduced in 1973 when Klee asked how many guards are sufficient to "guard" the interior of a simple polygon having n vertices. Although it was shown that $\lfloor \frac{n}{3} \rfloor$ guards are always sufficient and sometimes necessary [12, 14], and such a set of guards can be computed easily, such solutions are usually far from optimal in terms of minimizing the number of guards for a particular input polygon. Moreover, it was shown that determining an optimal set of guards is NP-hard, even for simple polygons [23]. Approximation algorithms with logarithmic approximation ratios are known ([13, 15, 17]) for somewhat restricted versions of the problem, e.g., requiring guards to be placed at vertices or at points of a discrete grid. Constant-factor approximations are known for guarding 1.5D terrains and monotone polygons (see [5, 22, 24]), and exact methods are known for the special case of rectangle visibility in rectilinear polygons [28]. The unrestricted version of the optimization problem remains open; no approximation algorithms are known that are better than the trivial $\lfloor \frac{n}{3} \rfloor$-approximation that comes from the classical combinatorial bound.

Our Contribution. We propose heuristics for computing a small set of point guards to cover a given polygon. While we are not able to prove good worst-case approximation bounds for our methods (indeed, each can be made "bad" in certain cases), we conduct an experimental analysis of their performance. We give methods also to compute *lower bounds* on the optimal number of guards for each instance in our experiments, using another set of implemented heuristics for determining a set of visibility-independent witness points. (The cardinality of such a set is a lower bound on the optimal number of guards.) We show that on a wide range of input polygons, our heuristics work well in practice, often matching our computed lower bounds (in which case the solution is provably optimal), and always yielding at most 2 times the lower bound (for the randomly generated instances). To our knowledge, ours is the first attempt to conduct a systematic experimentation with guard placement heuristics, together with methods to compute lower bounds that give provable performance bounds in terms of approximation ratios.

Our implementation is built on top of the CGAL arrangement package. Our experiments are conducted on a variety of polygons, including many generated "randomly" using the RPG software of [2].

Related Work. Surveys on the art gallery problem are given in [1, 25, 26, 27]. A related problem to computing a minimum guard *cover* is the problem of computing optimal *partitions* of polygons, e.g., into the fewest convex or star-shaped subpolygons. These problems are theoretically much easier than coverage problems; polynomial-time algorithms based on dynamic programming are known [3, 21]. Of course, if a polygon can be partitioned into k convex or star-shaped subpolygons, it can be guarded using at most k guards; thus, partitioning can serve as a basis for one type of heuristic for guard coverage. We note, however (see Section 5.1), that there are simple examples for which a constant number of guards are sufficient to cover, while the best partition is of linear size.

*Partially supported by grants from the National Science Foundation (ACI-0328930, CCF-0431030, CCF-0528209), Metron Aviation, and NASA Ames (NAG2-1620).

†Stony Brook University, Stony Brook, NY 11794-4400.

‡Stony Brook University, Stony Brook, NY 11794-3600. jsbm@ams.sunysb.edu.

§Stony Brook University, Stony Brook, NY 11794-4400. epacker@cs.sunysb.edu.

2 Preliminaries

The input is a (possibly multiply connected) polygon P having a total of n vertices on its boundary. We let h be the number of holes in P; if $h = 0$, P is said to be *simple*. For points $p, q \in P$, we say that p and q are *visible* to each other if the segment pq is contained in P. For $p \in P$, we let $\mathcal{V}(p)$ denote the *visibility polygon* of p, i.e., set of all points $q \in P$ that are visible to p. Clearly, $\mathcal{V}(p)$ is star-shaped and p belongs to its kernel. Similarly, for a set of points $\mathcal{P} \subseteq P$ we denote by $\mathcal{V}(\mathcal{P}) = \bigcup_{p \in \mathcal{P}} \mathcal{V}(p)$ the union of all of the visibility polygons of the points of $\mathcal{P}$. We say that a set $G \subset P$ of points is a *guarding set* of P if $\mathcal{V}(G) = P$. We let $g(P)$ be the number of guards in a minimum-cardinality guarding set of P.

We say that a finite set, $I \subset P$, of points in P is a *visibility-independent* set of *witness points* if the visibility polygons $\mathcal{V}(p) : p \in I$ are pairwise-disjoint: $\forall_{p,q \in I} \mathcal{V}(p) \cap \mathcal{V}(q) = \emptyset$. We let $i(P)$ denote the *independence number* of P, defined to be the number of witness points in a maximum-cardinality visibility-independent set. Clearly, $g(P) \geq |I|$ for any visibility-independent set I, since no single guard is able to see more than one point of I. Thus, if we find a visibility-independent set I and a guarding set G such that $|I| = |G|$, then G is an optimal guarding set (e.g., see Figure 13(e)). Not all polygons admit two such sets; indeed, there can be an $\Omega(n)$ gap between $i(P)$ and $g(P)$ (see Section 5.1). We say that a polygon for which $i(P) = g(P)$ is a *perfect polygon* (in analogy with the concept of perfect graphs).

In our experiments, our goal will be to find small guarding sets G and large visibility-independent sets I; the set G we produce approximates the optimal guard number, $g(P)$ with approximation ratio $|G|/|I|$.

3 Algorithms

3.1 Greedy Algorithms. A natural approach to placing guards is to do so greedily: Add guards one by one until coverage is achieved, choosing at each step a guard from among a set of candidates in order to maximize its contribution to the coverage (e.g., the "amount" of P that it sees that was not previously covered). Greedy methods are well known in set cover problems, as they yield logarithmic approximation bounds (which are best possible in some sense). We formulate this process as having two phases: First, we construct a set S of candidate guards that serve as a cover of P: $P = \mathcal{V}(S)$. Then, we greedily select a (much smaller) subset, $G \subset S$, of the candidates that also serves to cover P.

- Constructing a candidate set: The challenge of this step is to come up with a "good" candidate set S from which a good guarding set can be extracted.
- Criteria for greedily choosing guards $G \subset S$: From S we choose a smaller set G that also covers P. The challenge here is to derive a good heuristic for choosing guards that results in a small set G.

3.1.1 Constructing a Candidate Set. We experiment with three choices of candidate sets.

The first one, $V(P)$, is essentially the set V of vertices of P; it is easy to see that V guards all of P. For implementation purposes, we actually used points arbitrarily close to the vertices of P from inside, in order to avoid degenerate visibility on the adjacent edges – see Section 2. The exact position is on the ray that bisects the interior angle of the corresponding vertex. We also made sure that each point is actually inside P.

In our second choice of candidate set, we place the candidate points at the centers (centers of mass) of the convex polygons in a decomposition of P induced by an arrangement of certain line segments. We consider two different arrangements. One arrangement is defined by *edge extensions*, extending each edge that is incident to a reflex vertex v, extending it beyond v until it intersects the boundary of P at some other point; see Figure 1(a). The second arrangement is defined by extensions of visibility graph edges that are incident to at least one reflex vertex, v: If v sees vertex u, then we construct a segment vw that extends along the line through u and v, away from u, until it first encounters a point w on the boundary of P. See Figure 1(b). The arrangement of these *visibility extensions* also give rise to an arrangement having convex faces within P. Finally, $C(P)$ is defined to be the set of centers of mass for the convex faces in the arrangement (either of edge extensions or visibility extensions). It is easy to see that $C(P)$ guards P. Note that there are $O(n)$ edge extensions (thus, their arrangement has worst-case complexity $O(n^2)$) and there are $O(n^2)$ visibility extensions (with arrangement complexity $O(n^4)$). The intuition behind using these partition-based candidate sets is that different subpolygons will contain points with different combinatorial type with respect to visibility within P. For instance, in Figure 1(a), points in different subpolygons see different subsets of the reflex angle edges: points in the left subpolygon see the left edge incident to the reflex vertex, points in the right subpolygon see the right edge, and points in the middle subpolygon see both. It is clear in this example that the best choice is to select a guard from the "middle" subpolygon. We later try to construct criteria that reflect this intuition.

Finally, the third choice of candidate set is simply the union: $V(P) \cup C(P)$.

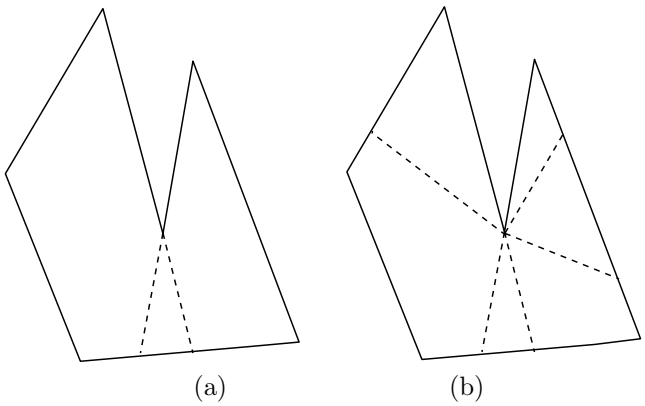


Figure 1: A polygon with (a). edge extensions, and (b). visibility extensions. Extensions are drawn with dashed lines.

3.1.2 Extracting a Small Guarding Set. We experiment with the three candidate sets, $V(P)$, $C(P)$ and $V(P) \cup C(P)$, defined in the previous subsection. In the descriptions below, we use $W(P)$ to denote the candidate set chosen. Our goal is to generate a guarding set $G \subset W(P)$ that is as small as possible; in general, we expect $|G|$ to be much less than $|W(P)|$.

We greedily add "good" candidates to the guarding set G, until the entire polygon is covered. The next guard is selected to maximize a certain measure, among all of the remaining candidates. We considered the following algorithms, labeled "A_1" through "A_{13}" (another algorithm, A_{14}, is presented in Section 3.2):

- A_1. The candidates are $W(P) = V(P) \cup C(P)$, with $C(P)$ based on the arrangement of edge extensions. The score, $\mu(c)$, for a candidate $c \in W(P)$ is the number of points of $W(P)$ that are seen from c that are not already seen by a point of G. At each iteration of the algorithm, the candidate c with the highest score $\mu(c)$ is added to G, and the scores $\mu(\cdot)$ are updated accordingly.

- A_2. A_2 is similar to A_1 with the following modification: With each candidate c added to G, the arrangement is augmented with the edges of $\mathcal{V}(c)$ that are not on the boundary of P. The idea is to enrich the set of candidates to reflect the still uncovered portion of the polygon. See Figure 2.

- A_3. A_3 is similar to A_1, but the score $\mu(c)$ of candidate c is the area seen by c that is not yet seen by G. This algorithm requires that after each candidate is added to G, we update the (unguarded) visible area for each remaining unused candidate.

- A_4. A_4 is similar to A_1, but uses a score $\mu(c)$ that weights the candidates c' that are not-yet-covered by c, by the (precomputed) area of the cell corresponding to c'.

- A_5. A_5 is the same as A_4, but the weight associated with c' is the length of the boundary of the cell in common with ∂P, instead of its area.

- A_6. A_6 is the same as A_4, but the weight associated with c' is the fraction of the perimeter of the cell that is in common with ∂P.

- A_7. A_7 is like A_1, but with candidates $W(P) = V(P)$ (vertices of P).

- A_8. A_8 is like A_1, but with candidates $W(P) = C(P)$.

- A_9. A_9 is like A_1, but with the score $\mu(c)$ defined to be the number of not-yet-covered vertices seen by c.

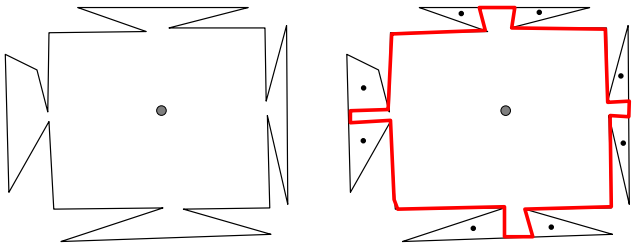
- A_{10}. A_{10} is like A_1, but with the score $\mu(c)$ defined to be the number of not-yet-covered cell centers seen by c.

- A_{11}. A_{11} is like A_1, but with the arrangement based on visibility extensions. As a result, the number of candidates becomes much larger (worst-case $O(n^4)$, versus $O(n^2)$).

- A_{12}. A_{12} is a combination of A_2 and A_{11}, using dynamically added segments, and arrangements of visibility extensions.

- A_{13}. A_{13} is a probabilistic algorithm, based on the Brönnimann-Goodrich framework [10, 13]. Each candidate is assigned a weight dynamically, proportional to its chances to be selected. Initially, each point is assigned weight 1. In each iteration, a guard is selected randomly. Then a random point p that is still unguarded is selected. We find all non-guarding candidates that see p and double their weights, improving the chances of guarding p in the next iterations. This process gives an $O(\log \Phi)$-approximation on average, where Φ is the optimal number of guards selected from $W(P)$; [13] applied this strategy for candidates defined by a grid.

All of the above heuristics may result in a set G that is not *minimal* – i.e., it may be possible to remove one or more guards while still covering P. Thus, we apply a post-processing step in which we iteratively remove redundant guards until we are left with a minimal guarding set.



(a) (b)

Figure 2: Using algorithm A_2: (a). The polygon and the first guard to be selected (shaded). (b). The visibility polygon of the guard (highlighted, in red) caused the addition of 8 new candidates (small black disks).

Remark. As reflected in our list of algorithms, we formulated several parameters that control the behavior. Together, they can be combined to yield numerous variants, making it tedious and impractical to test all of them. Instead, we defined a basic variant (A_1), which we believed would give good results and be relatively simple and time efficient. Other variants differed in one or two parameters from A_1. We were interested both in the effect of this modification and in the possibility to modify parameters further. As our experiments showed (see Section 5), the only positive influences were obtained with A_2 and A_{11}. Thus, we also tried a combination of the two, A_{12}.

3.2 Methods Based on Polygon Partition.

A very different approach is to base the guarding on *partitioning* the polygon into pieces, each with an assigned guard:

- A_{14}. Partition the polygon into star-shaped pieces, and place one guard within the kernel of each piece. While a polynomial-time algorithm is known for partitioning a simple polygon into the fewest star-shaped pieces, we opt instead to apply a simpler (and faster) heuristic, which applies also to polygons with holes: Triangulate P, and then iteratively delete diagonals (according to a heuristic priority), merging two adjacent faces as long as the resulting new face is still star-shaped (this is similar to the Hertel-Mehlhorn's heuristic that 4-approximates the minimum convex partition).

Since the results of this heuristic were relatively poor (see Section 5), we did not explore other variants of the algorithm further. Note too that for some "spike box" polygons, any partition-based algorithm will give very poor results compared with greedy coverage methods (see the discussion in Section 5.1).

3.3 Algorithms for Visibility-Independent Sets.

As we described in Section 1, a visibility-independent set I provides a lower bound on the guarding number. Since maximizing $|I|$ is NP-hard, we resort to heuristics for finding independent sets. (While there are effective combinatorial optimization methods for exactly computing maximum independent sets, we opted here to use faster heuristics.) We apply a greedy heuristic similar to our guard placement strategies: We start with an initial set S of candidates (not independent) and iteratively add visibility-independent points to a set I (initially empty), adding at each step a point that sees the least number of points in S. After adding a point to I, we remove all of the points in S that see it. We stop when S is empty.

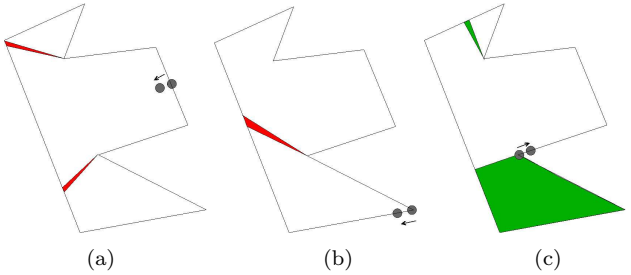
We note that if a point $p \in \partial P$ is perturbed slightly into the interior of P, its visibility polygon will usually enlarge; see Figure 3(a). Since in our greedy algorithm, we want to choose points that see as little as possible, a natural heuristic is to choose points of ∂P as candidates. Also, when perturbing a point at a convex vertex along ∂P, the visibility polygon usually enlarges; see Figure 3(b). Thus, we include convex vertices in the candidate set. For reflex vertices, however, a perturbation away from the vertex along ∂P generally causes the visibility polygon to become smaller; see Figure 3(c). This motivates using a candidate set that includes two kinds of candidates: convex vertices (denoted by I_1) and midpoints of edges incident to two reflex vertices (denoted by I_2). Let $I_3 = I_1 \cup I_2$. We experimented with independent sets built using candidates of all 3 types. Not surprisingly, it turned out that I_3 gives the best results, as it has a richer set of candidates.

Figure 4 shows a test on a 44-vertex polygon. The guarding candidates set, the guarding set, and the visibility-independent set are shown.

4 Implementation Details

4.1 Computing Extension Edges.

In all cases except vertex guard candidates, we compute a subdivision of the polygon into cells, as described in Section 3.1.1, by extending certain line segments that connect pairwise visible (possibly adjacent) vertices. If P has no holes, we can apply the algorithm of Hershberger [18], using time $O(n+k)$; otherwise, we can compute the visibility graph in time $O(k + n \log n)$ (e.g., [16]). Here, k is the output size and is the number of possible visibility extensions (at most $O(n^2)$). If we use only edge extensions, then there are $O(n)$ such extensions, and each can be determined using a ray-shooting query ($O(\log n)$ time query). The extensions, together with ∂P, constitute the input segments for the arrangement computation.



(a) (b) (c)

Figure 3: The effect of moving guards. The arrow indicates the direction, red regions are added to the visibility, while green regions are removed. (a). Moving the point inside the polygon from the boundary usually increases visibility, (b). Moving from a convex vertex along the boundary usually increases visibility. (c). Moving along the boundary from a reflex vertex usually decreases visibility.

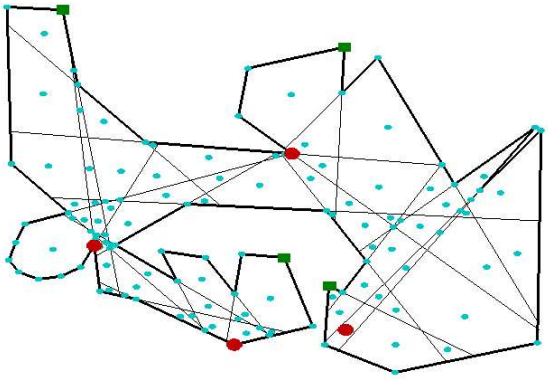
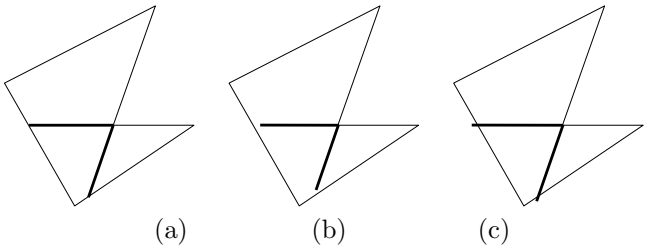


Figure 4: The result of running a test on a polygon with 44 vertices with heuristic A_1, independent candidate set I_3, and using edge extensions for the polygon subdivision. The large (red) disks are the guards, the (green) squares are the visibility-independent points, the small (cyan) disks are the remaining candidates, and the black segments inside the polygon are the edge extensions.



(a) (b) (c)

Figure 5: A polygon with edge extensions. Extensions are in bold. (a) The extensions obtained with exact precision: extensions end exactly on polygon edges. (b) Using finite precision may lead to too short extensions, resulting in erroneous cell merging. (c) Extensions are pushed slightly further to guarantee that the cells are closed and have correct topology.

4.2 Using the Arrangement to Compute a Candidate Set. The polygon boundary edges and the edge- or visibility-extensions are the segments for which we compute an arrangement, $A(P)$, of complexity $O(n^2)$ (for edge-extensions) or $O(n^4)$ (for visibility-extensions). Each (convex) cell of $A(P)$ contributes one candidate, which we take to be the center of mass of the cell.

We use the CGAL arrangement package to compute $A(P)$. We found that using exact arithmetic is extremely slow; thus, we opted to use floating point, with some small perturbations of segment endpoints to avoid robustness difficulties. Since the endpoints of extensions are imprecise, we may erroneously combine two adjacent cells if we are not careful. To avoid this error, we push the extension endpoint slightly beyond the intersection with ∂P. In this way, the interior cells will be separated and either the unbounded cell or a hole will have "notches" (see Figure 5). Although those cells are affected, they are irrelevant since they are outside of P. We did not attempt to compute the minimum perturbation to be robust; rather, we assumed that a small extension by a prespecified ϵ was sufficient for our purposes (and never had an issue arise from this assumption). We were careful, however, to check that this extra extension did not result in reentering the interior, which may happen if two edges are very close. In this case, we decrease the extra extension accordingly and still assume that it is sufficient for robust computation. However, we did not encounter polygons with this rare feature in our experiments.

After computing the full arrangement $A(P)$, we compute the candidates of each cell. Then we compute the pairwise visibility among the candidates. If the polygon is simple, one can apply the algorithm of [4] to do so in time $O(n^2 \log^2 n + k)$ or $O(n^4 \log^2 + k)$

(using space $O(n^2)$ or $O(n^4)$), for edge- and visibility-extensions, respectively. In some of our heuristics we need to extract features of the cells (see Section 3.1.2); this takes time linear on the complexity of the cells and thus linear in $|A(P)|$ (worst-case $O(n^2)$ or $O(n^4)$).

4.3 Computing a Guarding Set. We iteratively choose a guard and make the necessary updates until the polygon is covered. We next describe the different routines of this process.

4.3.1 Choosing the Next Guard. This routine depends on the algorithm we use. We use two kinds of algorithms: the greedy and the probabilistic.

Greedy. We check each candidate that has not yet been selected. According to the scoring criteria of the algorithm, we score each candidate and select the one with the highest score to be the next guard. The work for each candidate is constant, except in the case of heuristic A_3, for which we need to update the arrangement for each new guard ($O(n \log n)$ time for each).

Probabilistic. We simply select the guard chosen randomly with respect to the weights. Then, we remove its weight from the total weights.

It is worth mentioning that in both types of algorithms it is possible that all candidates are visible to the temporary list of guards, yet the polygon is not yet covered. In this special case we identify the area that is not covered (using the data structure in Section 4.3.3), compute its center of mass and find a candidate that sees this point by using point location to find in which cell of $A(P)$ it is located.

4.3.2 Updating the Data Structures. Once a candidate guard is selected, we need to update the relevant data structures before choosing the next guard. Again, it differs for each kind of algorithm we use, as we describe below.

Greedy. We update the weights of the candidates as follows. Let g be the new guard and $K(g)$ be the visible candidates from g. Among these, there may be candidates that are newly guarded and thus need not be considered any more. For each such candidate, c, we again take all of its visible candidates. For each, we update its weight by removing the effect of c. All of the information about the visibility among points was precomputed, as described in Section 4.2.

Probabilistic. We find a point that is still not guarded (if there are still such points), and double the weights of all of the candidates that see it (including the point itself).

4.3.3 Checking Coverage. We maintain the region, M, that is guarded by the current guarding set, G, as follows. We initialize M to be empty. For each new guard g we perform $M = M \cup \mathcal{V}(g)$ by performing a union of polygons. At each step we check whether $P = M$. If so, the polygon is fully covered. If the polygon is simple, then any of the above union computations takes linear time. (Note that M will never have holes.) If P has holes, then M can have holes and can have quadratic complexity; thus, each computation may take quadratic time. In order to compute visibility polygons we implemented the linear-time algorithm of Joe and Simpson [19], for visibility in simple polygons, and we use a radial sweep around the guard, for visibility in polygons with holes (time $O(n \log n)$).

4.4 Removing Redundant Guards. By using the several heuristics we described above, we decrease the guarding set significantly from the initial set of candidates. However, it may still be the case that more guards can be removed while maintaining full coverage. It may happen that after selecting a guard g, other guards are selected such that they cover the parts covered alone by g when it was chosen.

In order to remove redundant guards, we use the following routine. We maintain an arrangement of visibility polygons of the guarding set. We traverse the guards and for each guard do the following. We remove the guard's visibility polygon edges from the arrangement. Then, we check whether the new faces are covered by the remaining guards. If they are, the guard is removed from the guarding set. Otherwise, we return the removed visibility polygon edges to the arrangement. By the time the routine is completed, the guarding set is clearly minimal.

4.5 Computing Visibility-Independent Sets. In Section 3.3 we explained how we construct the independent candidate set and how we select the next independent point. We use an arrangement, A, of visibility polygon edges. For each point being selected, we insert the corresponding visibility polygon into A and check whether it intersects previous visibility polygons. If it does not, we insert the point to the independent set. Otherwise we remove its visibility polygon edges from A.

5 Experiments

We have implemented our various algorithms on a PC using openGL and CGAL (version 3.1) libraries. Our software works with Microsoft Windows XP with Visual .Net compiler. The tests were performed on a Microsoft Windows XP workstation with an Intel Pentium 4 CPU

3.20GHz, 2.00 GB of RAM.

We have performed extensive experiments with the algorithms we described in Section 3.1.2. In this section we report our results and conclusions from our experiments.

We found A_1 (the basic algorithm) to be very useful in the sense that both the guarding was efficient and the time and space required were relatively reasonable. The results obtained with algorithm A_2 were satisfactory and overall showed to be a little better than A_1. However, we paid for a much longer processing time. It was not practical for large polygons that require many guards and, thus, many iterations. In contrast, A_1 had no trouble handling even larger polygons. Although the algorithm A_3 seems useful for testing, it was extremely slow—too slow for being a candidate here. Thus, we omit its results here. Algorithms A_4, A_5 and A_6 gave reasonable results, but A_1 turned out to be at least as good or better in most tests we performed. Algorithms A_7, A_8, A_9 and A_{10} were usually worse or equal to A_1. A_7 had very bad results for the spikes box (polygon f in Figure 13). Algorithm A_{11} gave slightly better results than A_1 and was comparable to A_2. Naturally, it took more time and required more space than A_1. It was more time-efficient than A_2, but the space requirement was problematic, and it even exploded the space when the arrangement of the visibility edges was very large. The algorithm A_{12} gave the same results as A_{11}. Since it is more complicated than A_{11}, there is no reason to use it instead. The results of algorithm A_{13} were not as satisfactory as A_1. Also, the processing time was significantly higher, as the number of redundant guards was enormous, resulting in a lot of time for both computing the guarding set and removing some of them. The results of A_{14} were the worst among all algorithms we tried. We implemented and tested this algorithm only with simple polygons. Based on the results, we found it useless to implement and test this algorithm for polygons with holes.

Our tests include both interesting cases produced manually and randomly generated polygons [2]. Figure 13 shows some of the interesting cases we tested. All show the guarding and independent sets. Table 1 (see Section 5.3) shows statistics obtained with the algorithms we used on 40 input sets. As we show later, the best results were obtained with A_2 and A_{11} (ignore Algorithm A_{12} for the moment). Those results were a little better overall than A_1. The graphs in Figure 5 illustrate their results and show how close they were. While the results of A_1 are slightly worse than the other two, it takes substantially less time and space. Both (especially A_2) were substantially slower than A_1, and both (especially A_{11}) required much more space. A_2 was found to be way too slow for big polygons (randomly generated 200 vertex polygons) to be practical, while A_{11} ran out of space for one of our polygons, which had a large size of $A(P)$. The good results of A_2 and A_{11} motivated the testing of their combination, with dynamically inserted edges and visibility extensions (heuristic A_{12}). However, the results with A_{12} were almost identical to the results with A_{11}. Thus, we found no use in considering A_{12}. Based on all these results, we decided to concentrate on algorithms A_1, A_2 and A_{11}. Figure 15 shows a few 100-vertex polygons we tested, with each of the algorithms A_1, A_2 and A_{11}. As shown in this figure, the different heuristics were better for different examples. The three heuristics imposed a trade-off between guarding quality and time/space.

In order to explore their performance further and get more concrete information, we tested these three heuristics with more input sets, which strengthened our conclusions of the trade-off we described. This trade-off implies that when the input is small enough, or when time and space are not an issue, using A_2 and A_{11} would be recommended. As time and space become restricted, the use of A_2 (mainly when the time is restricted) and A_{11} (mainly when the space is restricted) becomes less desirable and motivates the use of A_1.

In order to get a quantitative measure on the quality of the guarding sets computed, we computed visibility-independent sets too. The ratio between the best guarding and the best visibility-independent set (obtained with I_3) never exceeded 2, for randomly generated polygons, and was usually close to 1. That implies that our results were always a 2-approximation to the optimal guarding, and usually were much better.

We analyze the time taken by the three algorithms. With those statistics, we want to show the correlation between the parameters and the time, and also to compare the running time of the three algorithms. Figure 5 shows the number of guards as a function of the number of vertices. It shows that there is a slight correlation between the two, but generally it shows that the number of guards is almost independent of the number of vertices. It should be clear that generally there is no correlation between the two, as any convex polygon requires only one guard, while "comb" polygons (see Figure 13(k)) requires $n/3 - O(1)$ guards. Figures 8 and 9 show the time as a function of the number of guards, extension edges and vertices. The figures show that although there is a correlation between the parameters and the time, this correlation is not strong and clearly there is a lot of noise. Thus, it would be hard to predict the time based on the parameters of the input. This observation can be explained also by the observation above about vertices vs. guards. Clearly

both sizes should affect the running time and, since they are not highly correlated, the time is not correlated with each. One thing that can be clearly observed from those figures is that A_2 took the most time, while A_1 took the least (although there were exceptions).We can also observe that for some polygons, A_2 and A_{11} took much more time than A_1, while there are cases in which the difference is not big.

5.1 Input Datasets. We tried to explore an interesting and broad variety of input datasets. Except for the randomly generated polygons, we constructed many more examples. We present some of them below.

Well. This interesting example demonstrates a case in which more than one guard "collaborates" in guarding "wells" (see Figure 13(a)). The challenge in guarding such polygons is that there is dependency among guards: if one moves, another will have to move too in order to compensate for the loss of visibility.

Spike box. This example is composed of a rectangular box with many spikes coming out of it (see Figure 13(f) and 14(h)). This example demonstrates how inefficient a partition technique can be. In Figure 13(f), two guards are sufficient to guard all spikes (and the rectangular area too). However, if we order the spikes around the rectangle perimeter and mark each spike according to the guard who sees it, we will have no consistency in this sequence. Let e be the number of guard changes along this sequence. If e is large, it is easy to see that any partition will subdivide the polygon into many pieces. In the worst case, a constant number of guards are sufficient to guard the polygon, while a linear number of guards will be necessary with partition techniques. Figure 14(h) is an example in which the upper bound was much greater than the lower bound (here, a ratio of 4 was obtained). In general, this ratio can be $\Theta(n)$.

Special art gallery variants. Figure 13(c) shows an example for simulating *line segments guarding* (a rectangular polygon with skinny rectangular holes). Other variants of interest are the guarding of the boundary of the polygon from the outside (Figures 13(h) and (i) and 14(e)) and the *prison yard problem*, which is a combination of guarding the boundary from the inside and outside (see Figure 14(d)). Note, however, that while each of these examples were constructed for instances involving guarding only of the boundary, here we produce guards that guard the entire free space. However, it would not be hard to modify the software to support boundary (surface) guarding; indeed, the problem becomes easier than area guarding. Another variant that is well researched is the case of *orthogonal polygons*, which usually have different combinatorial art

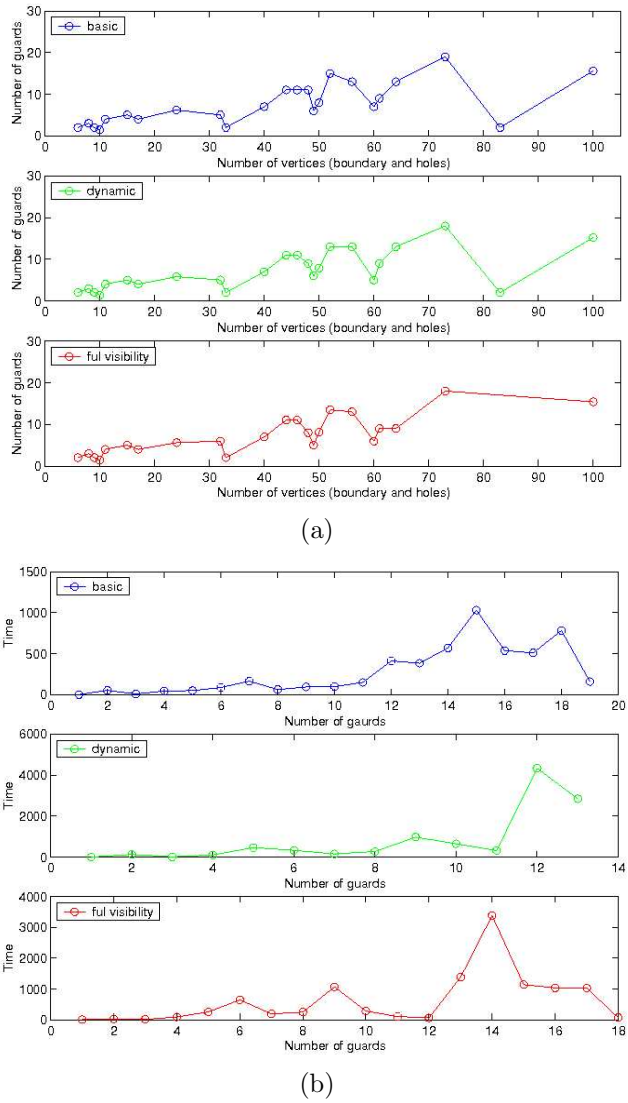
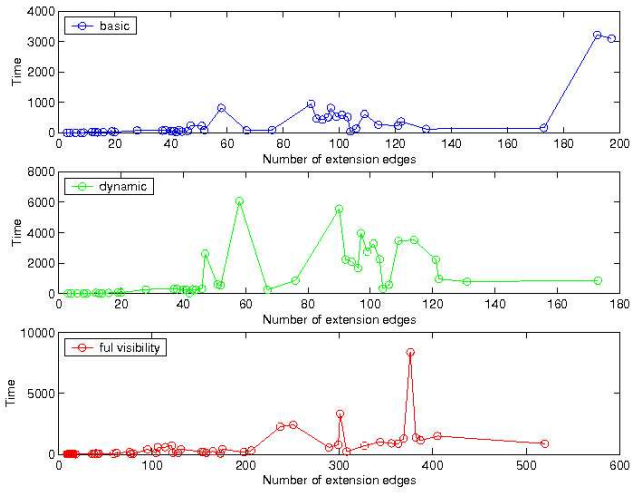
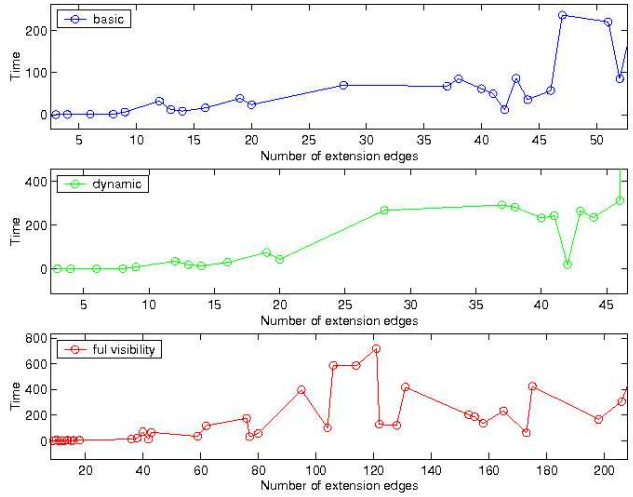


Figure 6: (a). Comparison of the guarding quality of A_1 (basic), A_2 (dynamic) and A_{11} (full visibility). The guarding number of all polygons with the same number of vertices were averaged. Circles represent the results. Note that the test with 83 vertices exploded the space with A_{11}; thus, it is not reflected in its graph. (b). Time as a function of guarding set size.



(a)



(b)

Figure 7: (a). Comparison of the running time as a function of the extensions size, comparing A_1 (basic), A_2 (dynamic) and A_{11} (full visibility). (b). A zoom-in of the plot above.

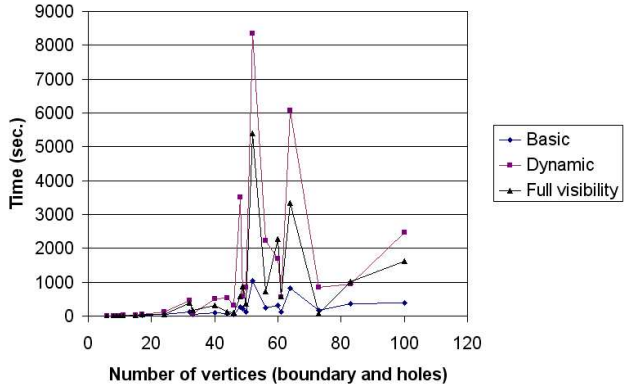


Figure 8: Time as a function of the number of vertices in the polygon.

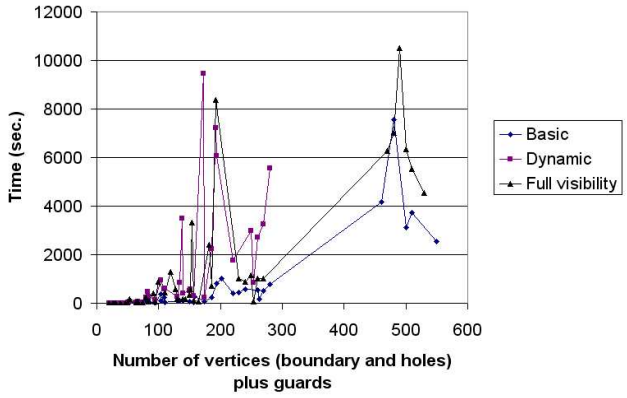


Figure 9: Time as a function of the number of vertices in the polygon, plus 10 times the number of guards. Note that A_2 (dynamic) took too long to run on the polygons with 200 vertices, so this data is not plotted.

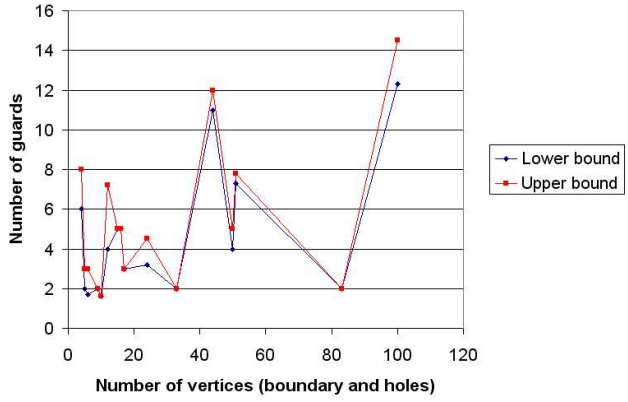


Figure 10: Comparing upper and lower bounds.

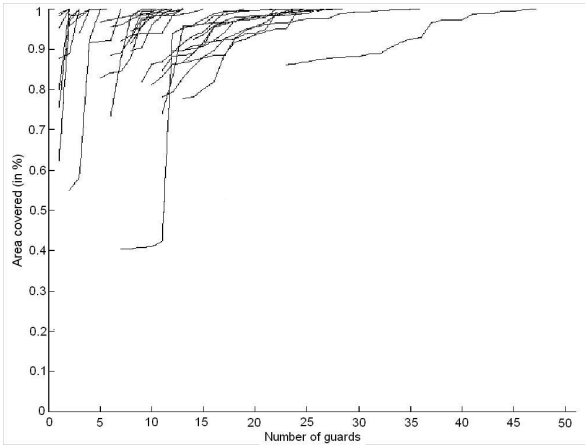


Figure 11: The area covered as a function of number of guards. The area in this graph corresponds to guarding sizes that range between the lower and the upper bounds we obtained. Each polygonal chain represents a different input. The area of the polygons was normalized (area 1 for each).

gallery theorems (see Figures 14(f) and (g)). Two more variants are *rectangular rooms* (Figure 14(i)) and *spiral polygons*, which are polygons composed of two chains – one with convex vertices and the other with reflex vertices (Figure 13(d) and 14(a)).

5.2 Covering Areas. Maximizing the area seen by a set of k guards has been studied (see, e.g.,[11]). The motivation is to guard a large portion of the polygon, while using a small and efficient guarding set. Since algorithm A_3 greedily finds the most uncovered area seen by the next guard, this algorithm may be expected to perform provably well. However, we did not explore this algorithm much as it was found to be extremely slow. Nevertheless, it would be interesting to evaluate the performance of other heuristics and see how well they cover the area, as more and more elements are added to the guarding sets.

Let l be the maximum lower bound obtained on the guard number of P; let $b > l$ be the upper bound obtained. We are interested to examine how the coverage area varies as the size of the guard set constructed by our algorithm varies from l to b. We implemented this experiment, and the results with A_1 are illustrated in Figure 11.

Let k be the size of an optimal guarding set of polygon P. Suppose we check how much area we cover with $k-1$ guards or less. The results can be arbitrarily bad, as illustrated in Figure 12. The polygon in this figure consists of a big triangle from which a branch, B, of complicated small regions spikes from one of its edges. Using A_1, we first choose guards to cover B, and might only guard the region $P \backslash B$ with the very last guard. Now suppose our bounds are not tight; then, using the first $k-1 \geq l$ guards gives us an arbitrarily bad area coverage, as B gets arbitrarily small.

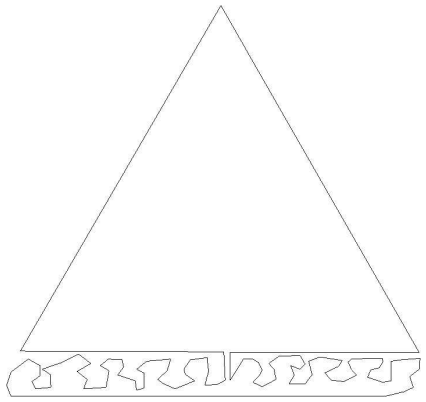


Figure 12: An example in which the guard that sees the most area is not chosen in the first place.

In our experiments, most of the time we covered at least 80% of the area of the polygon using l guards. In the worst example, only 40% was covered. Interestingly, there is a steep increase with one of the guards that is added after the lth guard. This indicates that the situation tends to be similar to what we described in the example above, where one of the "late" added guards contributes significantly to the total coverage area. We also observe that in most of the cases where the area covered was relatively small with l guards, adding a few more guards usually increased the area covered substantially. Thus, a good heuristic is to choose a number of guards that is not much more than l. According to our experiments, this method is likely to give better results. We note again, as discussed above, that the covering may be quite poor with this heuristic as well.

5.3 Main Table of Results. Let $\mathcal{A}$ denote the set of all heuristics we use, and let m be the number of input sets we tested. For each heuristic $A_i \in \mathcal{A}$, let $G_{A_i}(j)$ be the number of guards obtained on some input number j. Let $G_{min}(j)$ be the minimum number of guards obtained with all heuristics on input j. For heuristic A_i, let $K_i = \frac{\Sigma_{1 \leq j \leq m}(G_{A_i}(j) - G_{min}(j))}{|\mathcal{A}|}$, the average of number of guards minus the best guarding obtained with all heuristics, and let $M_i = \frac{\Sigma_{1 \leq j \leq m}((G_{A_i}(j) - G_{min}(j))/G_{min}(j))}{|\mathcal{A}|}$ represents the deviation from minimum in percentage. Let Q_i be the number of times the guarding obtained with A_i was the best among all heuristics (ties among the

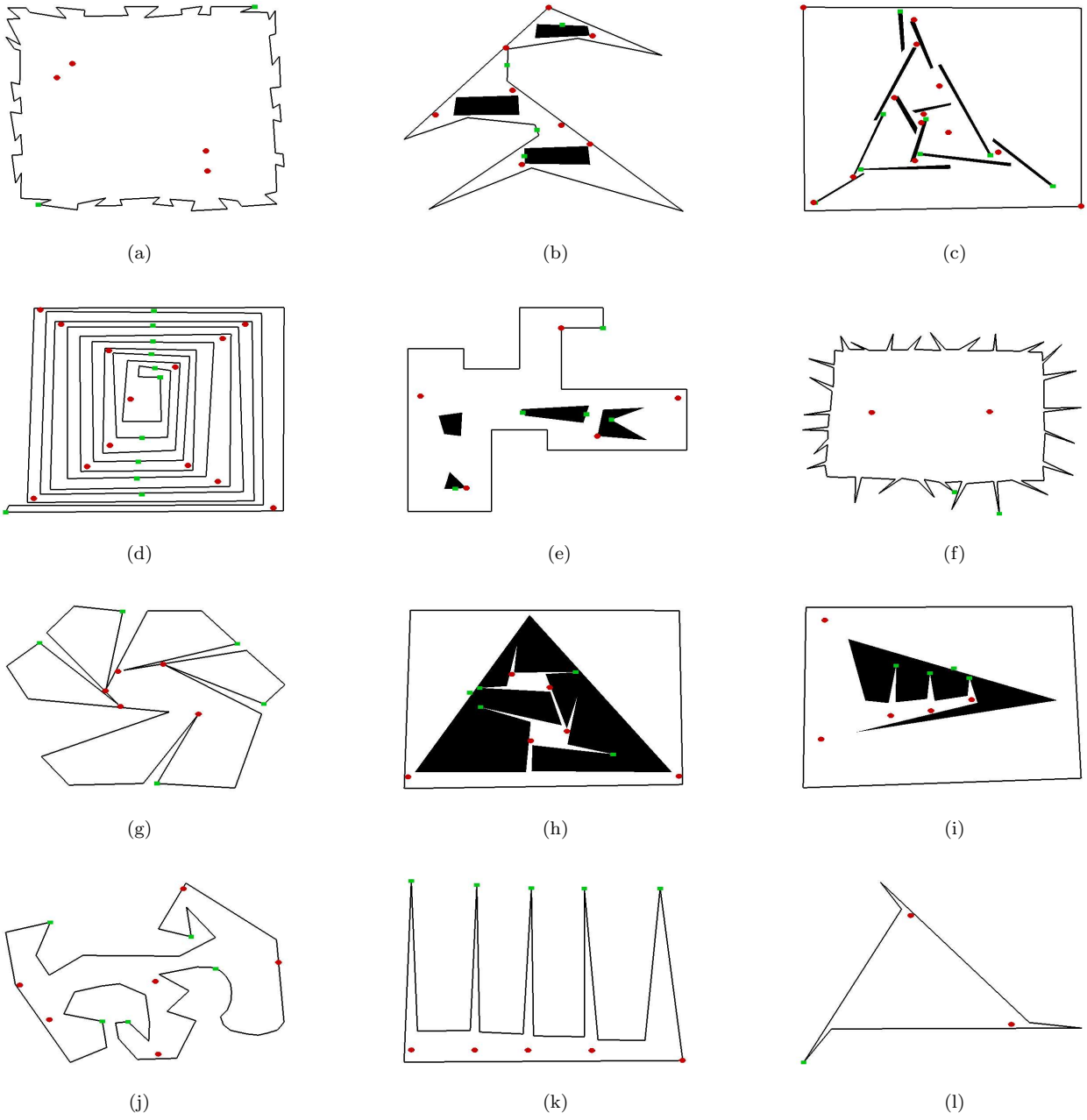


Figure 13: Experiment snapshots obtained with our software on different manually generated polygons, while using heuristic A_1. Red disks are the guards and green rectangles are the independent points. The black shapes inside some of the polygons represent holes. The sets in figures b, c, g, h and i are polygons with special features that were manually copied from [25].

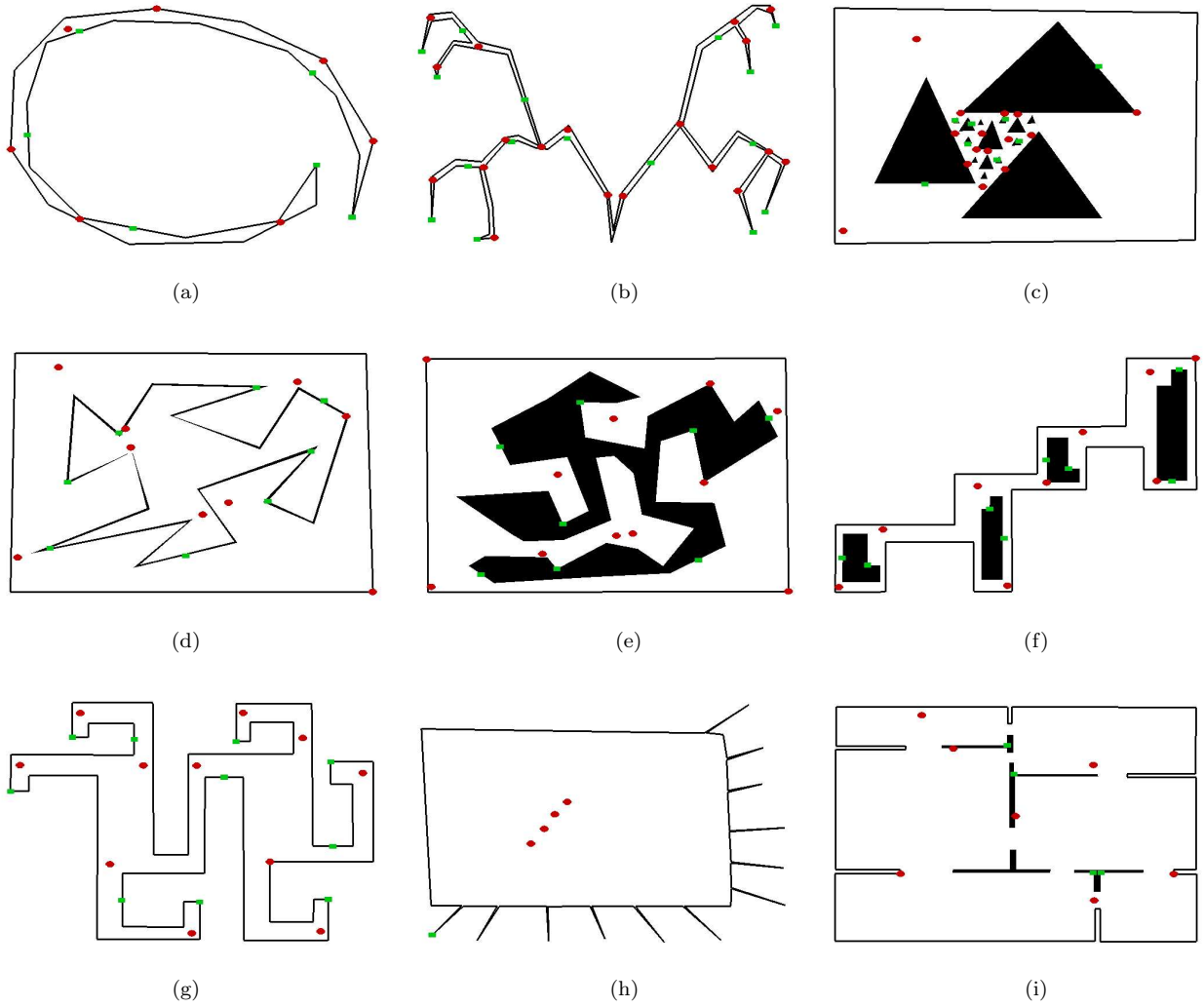
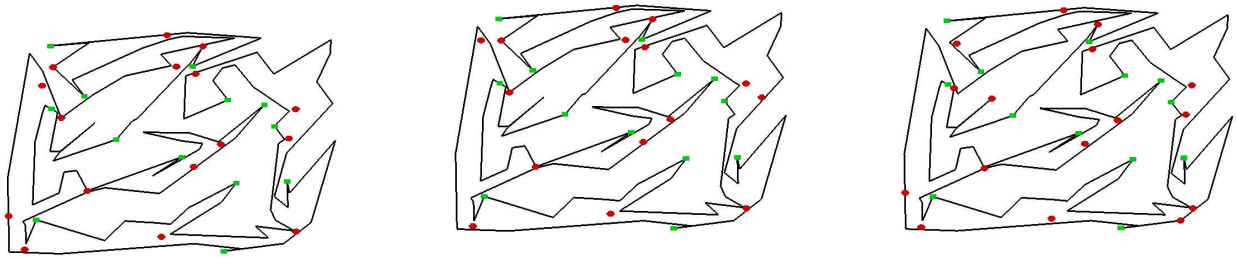
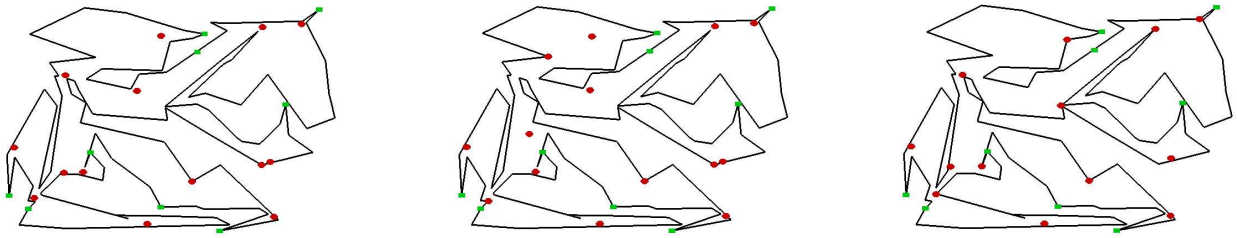


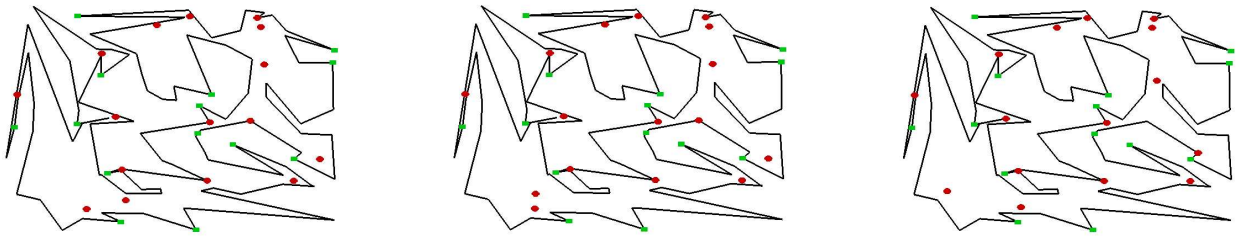
Figure 14: Experiment snapshots obtained with our software on different manually generated polygons, while using heuristic A_1. Red disks are the guards and green rectangles are the independent points. The black shapes inside some of the polygons represent holes. The sets in figures b, c, f, g and i are polygons with special features that were manually copied from [25].



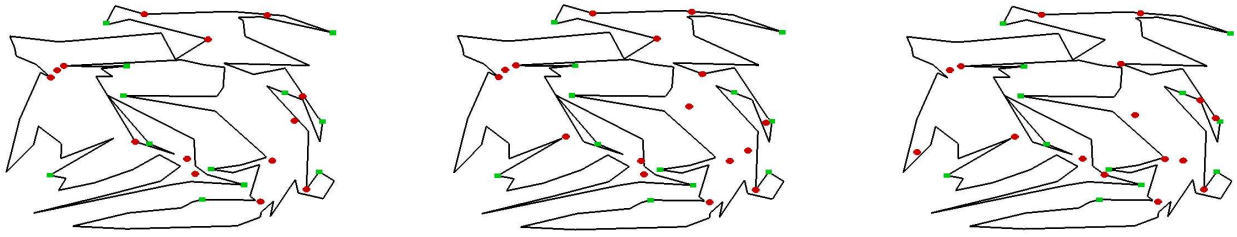
(a) 16 guards (b) 15 guards (c) 16 guards



(d) 14 guards (e) 14 guards (f) 13 guards



(g) 16 guards (h) 16 guards (i) 15 guards



(j) 14 guards (k) 16 guards (l) 16 guards

Figure 15: Experiment snapshots of guarding sets obtained with our software on different 100-vertex polygons (each row is dedicated to one input), with heuristics A_1 (first column), A_2 (middle column) and A_{11} (right column).

	A_1	A_2	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}
K	0.7	0.47	1.47	1.6	1.3	1.22	0.9	0.83	1.75	0.48	0.5	1.64	3.33
M	0.10	0.06	0.22	0.22	0.16	0.29	0.13	0.14	0.41	0.08	0.09	0.27	0.69
Q	16	17	11	11	10	10	11	12	8	15	15	9	8
B	40	40	40	40	40	40	40	30	29	39	38	39	30

Table 1: Results obtained with our heuristics on 40 input sets.

best guarding were counted for each heuristic). Let B_i be the number of tests that were actually completed, not counting those that either exploded the memory or whose candidate scores became zero values (and thus caused the algorithm to select candidates arbitrarily, until all points of the polygon are seen, thereby taking an unusually long time to complete). We use these notations (without subscripts) in Table 1.

6 Conclusions

We have conducted the first experimental analysis of a broad class of heuristics for locating guards to cover polygons. We designed and implemented several heuristics for guarding polygons, possibly with holes. We also computed visibility-independent sets, allowing us to obtain provable bounds on how close our results are to optimal. We concluded that there are three recommended heuristics, and they trade off guarding quality, space and time. The guarding sets obtained with them were very satisfactory in the sense that they were always either optimal or close to optimal (within factor 2 for all randomly generated instances) in all of the cases we encountered.

Most of our methods extend naturally to practical variants of the guarding and sensor coverage problem in which there are constraints on the visibility, e.g., view distance, good view angles of walls to be observed, robustness of coverage, etc [17, 20]. The methods also can be generalized to three dimensions, though the implementation would be substantially more involved.

There are several directions for further research. Since our results were very promising and the general art gallery problem is NP-hard, our main theoretical objective is to obtain provable heuristics.

We note that in recent and forthcoming work of Bottino and Laurentini [6, 7, 8, 9], experimental results are obtained similar to our own: They perform experiments on many input polygons, showing near optimality of guarding sets computed using their proposed heuristic (iterative) algorithms based on partitioning and covering.

References

[1] A. Aggarwal. The art gallery theorem: Its variations, applications and algorithmic aspects. PhD thesis, John Hopkins University, 1984.

[2] T. Auer and M. Held. Heuristics for the generation of random polygons. In *Proc. 8th Canad. Conf. Computat. Geometry*, pages 38–43, 1996.

[3] D. Avis and G.Toussaint. An efficient algorithm to decompose a polygon into star-shaped pieces. *Pattern Recognition*, 13:295–298, 1981.

[4] B. Ben-Moshe, O. Hall-Holt, M. J. Katz, and J. S. B. Mitchell. Computing the visibility graph of points within a polygon. In *Symposium on Computational Geometry*, pages 27–35, 2004.

[5] B. Ben-Moshe, M. J. Katz, and J. S. B. Mitchell. A constant-factor approximation algorithm for optimal terrain guarding. In *Proc. 16th ACM-SIAM Symposium on Discrete Algorithms*, pages 515–524, 2005.

[6] A. Bottino and A. Laurentini. Optimal positioning of sensors in 2D. In *Proc. CIARP*, vol. 3287 of *Lecture Notes Comput. Sci.*, pages 53–58, Springer-Verlag, 2004.

[7] A. Bottino and A. Laurentini. A practical iterative algorithm for sensor positioning. In *Proc. 10th IEEE Conf.Emerging Technologies and Factory Automation*, Sept., 2005.

[8] A. Bottino and A. Laurentini. A new art gallery algorithm for sensor location. In *Proc. ICINCO*, pages 242–249, Barcelona, 2005.

[9] A. Bottino and A. Laurentini. Experimental results show near-optimality of a sensor location algorithm. To appear, *Proc. IEEE Internat. Conf. Robotics and Biomimetics (ROBIO 2006)*, Kunming, China, December 17-20, 2006

[10] H. Brönnimann and M. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete and Computational Geometry*, 14:263–279, 1995.

[11] O. Cheong, A. Efrat, and S. Har-Peled. On finding a guard that sees most and a shop that sells most. In *Proc. 15th ACM-SIAM Sympos. Discrete Algorithms*, pages 1098–1107, 2004.

[12] V. Chvatal. A combinatorial theorem in plane geometry. *J. Combin. Theory Ser. B*, 18:39–41, 1975.

[13] A. Efrat and S. Har-Peled. Guarding galleries and terrains. *Information Processing Letters*, to appear, 2006.

[14] S. Fisk. A short proof of Chvátal's watchman theorem. *J. Combin. Theory Ser. B*, 24:374, 1978.

[15] S. K. Ghosh. Approximation algorithms for art gallery problems. *Proc. of the Canadian Information Processing Society Congress*, pages 429–434, 1987.

[16] S. K. Ghosh and D. M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM J. Comput.*, 20(5):888–910, 1991.

[17] H. González-Banos and and J-C. Latombe. A randomized art-gallery algorithm for sensor placement. In *Proc. 17th Annu. ACM Sympos. Comput. Geom.*, pages 232–240, 2001.

[18] J. Hershberger. An optimal visibility graph algorithm for triangulated simple polygons. *Algorithmica*, 4(1):141–155, 1989.

[19] B. Joe and R. B. Simpson. Correction to Lee's visibility polygon algorithm. *BIT*, 27:458–473, 1987.

[20] G. D. Kazakakis and A. A. Argyros. Fast positioning of limited visibility guards for inspection of 2D workspaces. In *Proc. IEEE/RSJ Internat. Conf. Intelligent Robots and Systems*, pages 2843–2848.

[21] J. M. Keil. Decomposing a polygon into simpler components. *SIAM J. Comput.*, 14:799–817, 1985.

[22] J. King. A 4-approximation algorithm for guarding 1.5-dimensional terrains. In *Proc. 7th Latin American Sympos. on Theoretical Informatics*, vol. 3887 of *Lecture Notes Comput. Sci.*, pages 629–640, Springer-Verlag, 2006.

[23] D. T. Lee and A. K. Lin. Computational complexity of art gallery problems. *IEEE Trans. Inform. Theory*, 32(2):276–282, 1986.

[24] B. J. Nilsson. Approximate guarding of monotone and rectilinear polygons. In *Proc. 32nd Internat. Colloq. Automata Lang. Prog.*, vol. 3580 of *Lecture Notes Comput. Sci.*, pages 1362–1373, Springer-Verlag, 2005.

[25] J. O'Rourke. *Art gallery theorems and algorithms.* Oxford University Press, Oxford, 1987.

[26] T. Shermer. Recent results in art galleries. *Proc. of the IEEE*, 80(9):1384–1399, 1992.

[27] J. Urrutia. Art gallery and illumination problems. In J. Sack and J. Urrutia, eds, *Handbook of Computational Geometry*, pages 973–1027. Elsevier Science Publishers, Amsterdam, 2000.

[28] C. Worman and M. Keil. Polygon decomposition and the orthogonal art gallery problem. *Internat. J. Comput. Geom. Appl.*, to appear, 2006.