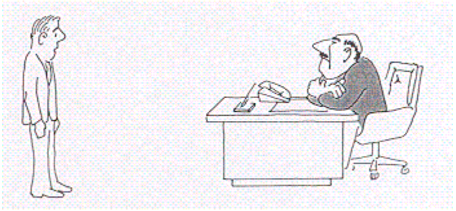
Introduction to Approximation Algorithms

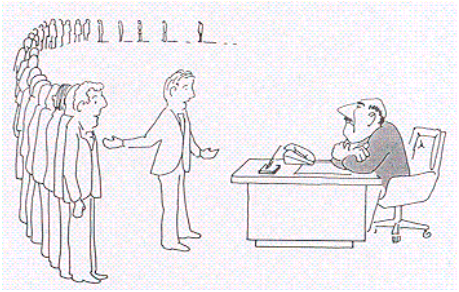
Subir Kumar Ghosh
School of Technology & Computer Science
Tata Institute of Fundamental Research
Mumbai 400005, India
ghosh@tifr.res.in

Overview



“I can’t find an efficient algorithm to the problem you have assigned to me. I guess I’m just too dumb. ”.

To avoid serious damage to your position within the company, it would be much better if you could prove that the assigned problem is inherently intractable, that no algorithm could possibly solve it quickly. Then you could stride confidently into the boss’s office and proclaim:



“I can’t find an efficient algorithm, but neither can all these famous people”.

At the very least, this would inform your boss that it would do no good to fire you and hire another expert on algorithms.

Background

- A large number of optimization problems which are required to be solved in practice are NP-hard.

Background

- A large number of optimization problems which are required to be solved in practice are NP-hard.
- For such problems, it is not possible to design algorithms that can find exactly *optimal solution* to *all instances* of the problem in time which is *polynomial* in the size of the input.

Background

- A large number of optimization problems which are required to be solved in practice are NP-hard.
- For such problems, it is not possible to design algorithms that can find exactly *optimal solution* to *all instances* of the problem in time which is *polynomial* in the size of the input.
- Of course, we assume that $P \neq NP$.

Background

- A large number of optimization problems which are required to be solved in practice are NP-hard.
- For such problems, it is not possible to design algorithms that can find exactly *optimal solution* to *all instances* of the problem in time which is *polynomial* in the size of the input.
- Of course, we assume that $P \neq NP$.
- If this stringent requirement is relaxed, the problems may be solved reasonably well.

1. M. R. Garey and D. S. Johnson, *Computers and Intractibility: A guide to the theory of NP-completeness*, W. H. Freeman, 1979.
2. R. Motwani, *Lecture Notes on Approximation Algorithms*, Volume 1, No. STAN-CS-92-1435, Stanford University, 1992.

- One possible approach of relaxation is to provide *near-optimal solution* rather than an *optimal solution* keeping the running time of the algorithm in polynomial.

- One possible approach of relaxation is to provide *near-optimal solution* rather than an *optimal solution* keeping the running time of the algorithm in polynomial.
- This gives the notion of the “approximate” solution of an optimization problem.

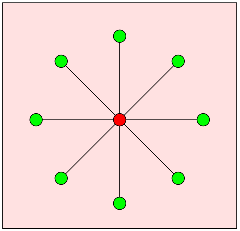
- One possible approach of relaxation is to provide *near-optimal solution* rather than an *optimal solution* keeping the running time of the algorithm in polynomial.
- This gives the notion of the “approximate” solution of an optimization problem.
- Some problems (e.g. Knapsack, Scheduling, Bin Packing, etc.) seem to be easy to approximate.

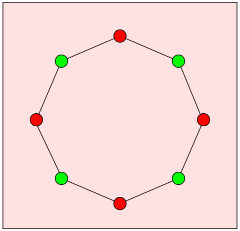
- One possible approach of relaxation is to provide *near-optimal solution* rather than an *optimal solution* keeping the running time of the algorithm in polynomial.
- This gives the notion of the “approximate” solution of an optimization problem.
- Some problems (e.g. Knapsack, Scheduling, Bin Packing, etc.) seem to be easy to approximate.
- On the other hand, there are problems (e.g. Graph Coloring, Travelling Salesman, Clique, etc.) are so hard that even finding very poor approximation can be shown to be NP-hard.

- One possible approach of relaxation is to provide *near-optimal solution* rather than an *optimal solution* keeping the running time of the algorithm in polynomial.
- This gives the notion of the "approximate" solution of an optimization problem.
- Some problems (e.g. Knapsack, Scheduling, Bin Packing, etc.) seem to be easy to approximate.
- On the other hand, there are problems (e.g. Graph Coloring, Travelling Salesman, Clique, etc.) are so hard that even finding very poor approximation can be shown to be NP-hard.
- Of course, there is a class of problems (e.g., Vertex Cover, Euclidean Travelling Salesman, Steiner Trees) which seem to be of intermediate complexity.

Vertex Cover

- [Instance] Graph $G = (V, E)$.
- [Feasible Solution] A subset $C \subseteq V$ such that at least one vertex of every edge of G belongs C.
- [Value] The value of the solution is the size of the cover $|C|$, and the goal is to minimize it.





Greedy Algorithm I

1. Take any edge of $e \in E$ and choose one of its vertex (say, v)

Greedy Algorithm I

1. Take any edge of $e \in E$ and choose one of its vertex (say, v)
2. Add v to C.

Greedy Algorithm I

1. Take any edge of $e \in E$ and choose one of its vertex (say, v)
2. Add v to C.
3. Remove all edges incident on v from E.

Greedy Algorithm I

1. Take any edge of $e \in E$ and choose one of its vertex (say, v)
2. Add v to C.
3. Remove all edges incident on v from E.
4. Repeat the process till all edges are removed from E.

Greedy Algorithm I

1. Take any edge of $e \in E$ and choose one of its vertex (say, v)
2. Add v to C.
3. Remove all edges incident on v from E.
4. Repeat the process till all edges are removed from E.

The above algorithm correctly computes a vertex cover. How good is the solution?

Performance Measure

- The “measure of goodness” of an approximation algorithm must relate the solution produced by the algorithm to the optimal solution of the optimization problem.

Performance Measure

- The “measure of goodness” of an approximation algorithm must relate the solution produced by the algorithm to the optimal solution of the optimization problem.
- In absolute performance measure, some constant k for $k > 0$ is obtained such that the difference between optimal and approximate solutions is bounded by k for all instances of the optimization problem.

Performance Measure

- The “measure of goodness” of an approximation algorithm must relate the solution produced by the algorithm to the optimal solution of the optimization problem.
- In absolute performance measure, some constant k for $k > 0$ is obtained such that the difference between optimal and approximate solutions is bounded by k for all instances of the optimization problem.
- In relative performance measure, a function R is obtained such that the ratio (called *approximation ratio*) between optimal and approximate solutions is bounded by R for all instances of the optimization problem.

Performance Measure

- The "measure of goodness" of an approximation algorithm must relate the solution produced by the algorithm to the optimal solution of the optimization problem.
- In absolute performance measure, some constant k for $k > 0$ is obtained such that the difference between optimal and approximate solutions is bounded by k for all instances of the optimization problem.
- In relative performance measure, a function R is obtained such that the ratio (called *approximation ratio*) between optimal and approximate solutions is bounded by R for all instances of the optimization problem.
- In the case of a minimization problem, R is the ratio of the approximation solution by the optimal solution (i.e., $R \geq 1$).

Performance Measure

- The "measure of goodness" of an approximation algorithm must relate the solution produced by the algorithm to the optimal solution of the optimization problem.
- In absolute performance measure, some constant k for $k > 0$ is obtained such that the difference between optimal and approximate solutions is bounded by k for all instances of the optimization problem.
- In relative performance measure, a function R is obtained such that the ratio (called *approximation ratio*) between optimal and approximate solutions is bounded by R for all instances of the optimization problem.
- In the case of a minimization problem, R is the ratio of the approximation solution by the optimal solution (i.e., $R \geq 1$).
- In the case of a maximization problem, R is the ratio of the optimal solution by the approximation solution (i.e., $R \leq 1$).

Greedy Algorithm II

1. Take a vertex $v \in V$ of maximum degree in the current graph.

Greedy Algorithm II

1. Take a vertex $v \in V$ of maximum degree in the current graph.
2. Add v to C.

Greedy Algorithm II

1. Take a vertex $v \in V$ of maximum degree in the current graph.
2. Add v to C.
3. Remove all edges of E incident on v from E.

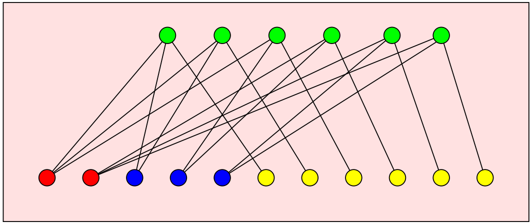
Greedy Algorithm II

1. Take a vertex $v \in V$ of maximum degree in the current graph.
2. Add v to C.
3. Remove all edges of E incident on v from E.
4. Repeat the process till all edges are removed from E.

Greedy Algorithm II

1. Take a vertex $v \in V$ of maximum degree in the current graph.
2. Add v to C.
3. Remove all edges of E incident on v from E.
4. Repeat the process till all edges are removed from E.

The above algorithm correctly computes a vertex cover. How good is the solution?



Optimal solution consists of green vertices.

Let $m!$ be the number of green vertices of degree m. So, there are $m!/m$ red vertices of degree m, $m!/(m-1)$ blue vertices of degree $(m-1)$, ..., $m!$ yellow vertices of degree 1.

Greedy algorithm may choose all non-green vertices. So, the total number of vertices chosen by the approximation algorithm is $m![1/m + 1/(m-1) + 1/(m-2) + \ldots + 1] \simeq m! \log(m)$

So, the approximation ratio of the greedy algorithm is $\log(m)$.

Can we design an approximation algorithm giving a constant approximation ratio?

Greedy Algorithm III

1. Take any edge $(u, v) \in E$.

Greedy Algorithm III

1. Take any edge $(u, v) \in E$.
2. Add both vertices u and v to C.

Greedy Algorithm III

1. Take any edge $(u, v) \in E$.
2. Add both vertices u and v to C.
3. Remove all edges of E incident on u or v from E.

Greedy Algorithm III

1. Take any edge $(u, v) \in E$.
2. Add both vertices u and v to C.
3. Remove all edges of E incident on u or v from E.
4. Repeat the process till all edges are removed from E.

Greedy Algorithm III

1. Take any edge $(u, v) \in E$.
2. Add both vertices u and v to C.
3. Remove all edges of E incident on u or v from E.
4. Repeat the process till all edges are removed from E.

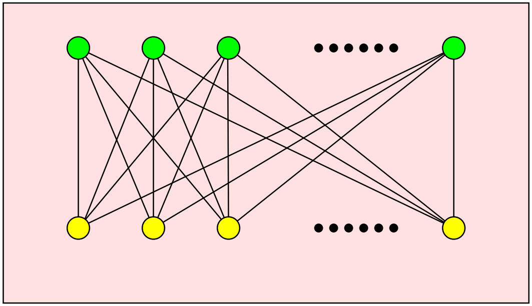
The above algorithm correctly computes a vertex cover. How good is the solution?

Observe that the set of edges chosen by the algorithm forms a maximal matching.

1. V. Vazirani, *Approximation Algorithms*, Springer, 2003.

Theorem: The approximation ratio of the Greedy Algorithm **III** is 2.

Tight Bound



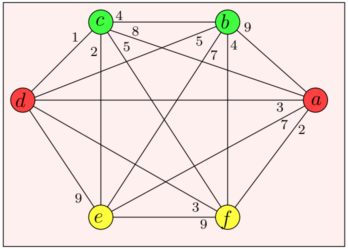
Open Problem: Design an approximation algorithm which gives a better approximation ratio.

1. G. Karakostas, *A better approximation ratio for the vertex cover problem*, ACM Transactions on Algorithms 5(4), 8 pages, 2009. (Ratio: $2 - 1/\sqrt{(}\log n))$.

Planar Graph Coloring

- The problem of coloring the vertices of a graph is to assign color to vertices such that no two adjacent vertices have the same color.
- The goal is to minimize the number of colors.
- The problem of deciding whether a planar graph is 3-colorable is NP-complete.
- The problem is to design an absolute approximation algorithm for coloring planar graphs such that the difference between an optimal solution and the solution obtained by the algorithm is at most 2.

Minumum Spanning tree



- Let $G = (V, E)$ be a connected undirected graph having non-negative weight $c(u, v)$ associated with each edge of E.
- A forest in G is the subgraph $F = (V, E' \subseteq E)$ with no cycle.
- A spanning tree in G is a forest with exactly one connected component.
- A minimum weight spanning tree T in G is a tree T whose sum of weights of the edges is minimum over all spanning trees.
- The problem is to compute the minimum weight spanning tree T in G.

Kruskal's Greedy Algorithm

1. Sort the edges of E by weights.

Kruskal's Greedy Algorithm

1. Sort the edges of E by weights.
2. Take the minimum weight edge e from the sorted list.

Kruskal's Greedy Algorithm

1. Sort the edges of E by weights.
2. Take the minimum weight edge e from the sorted list.
3. If e does not form a cycle with the existing edges of T, then add e to T.

Kruskal's Greedy Algorithm

1. Sort the edges of E by weights.
2. Take the minimum weight edge e from the sorted list.
3. If e does not form a cycle with the existing edges of T, then add e to T.
4. Otherwise, remove e form the sorted list.
5. Repeat the process till all edges in the sorted list are considered.

Kruskal's Greedy Algorithm

1. Sort the edges of E by weights.
2. Take the minimum weight edge e from the sorted list.
3. If e does not form a cycle with the existing edges of T, then add e to T.
4. Otherwise, remove e form the sorted list.
5. Repeat the process till all edges in the sorted list are considered.

The above algorithm correctly computes a vertex cover. How good is the solution?

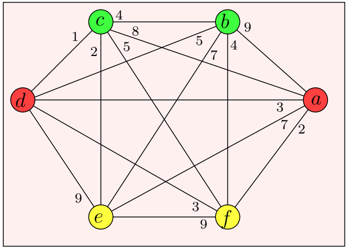
Kruskal's Greedy Algorithm

1. Sort the edges of E by weights.
2. Take the minimum weight edge e from the sorted list.
3. If e does not form a cycle with the existing edges of T, then add e to T.
4. Otherwise, remove e form the sorted list.
5. Repeat the process till all edges in the sorted list are considered.

The above algorithm correctly computes a vertex cover. How good is the solution?

Theorem: The greedy algorithm computes the minimum weight spanning tree in G in $O(n \log n)$ time.

Traveling-salesman Problem

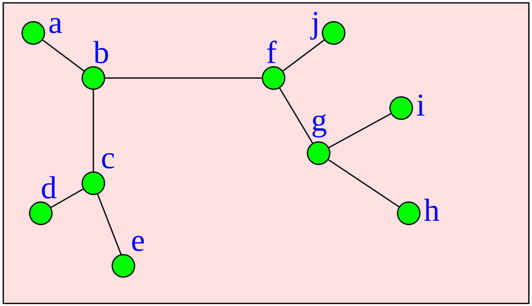


- Given a complete graph $G = (V, E)$ having a non-negative integer cost $c(u, v)$ associated with each edge of E, the problem is to find a minimum cost tour in G visiting every vertex of G exactly once.
- In other words, the problem is to find a Hamiltonian cycle in G with minimum cost.

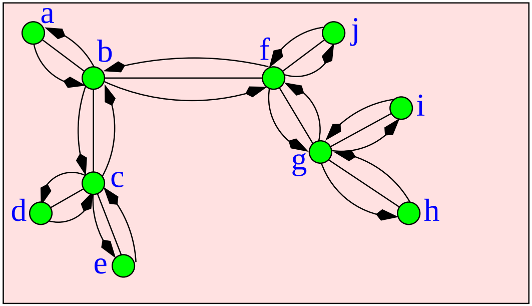
Theorem: For an polynomial time computable function $\alpha(n)$, Traveling-salesman problem cannot be approximated within a factor of $\alpha(n)$, unless $P = NP$.

Euclidean Traveling-salesman Problem

- In Euclidean traveling-salesman problem, the cost function satisfies triangle inequality i.e. $c(u, v) + c(v, w) \leq c(u, w)$.
- This restriction allows us to design approximation algorithms for Euclidean traveling-salesman problem.
- Approximation algorithm:



Step 1: Take a vertex a of G and compute a minimum cost spanning tree T for G with a as the root.

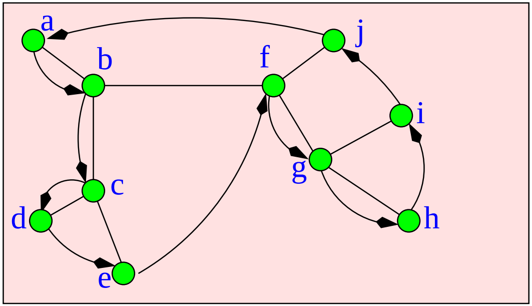


Step 2: Double every edge of T to obtain Eulerian graph.
Step 3: Find a Eulerian tour T' of T. Let S be the sequence of vertices in the Eulerian tour T'.

So, $S = [a, b, c, d, c, e, c, b, f, g, h, g, i, g, f, j, f, b, a]$

Step 4: Scan S from left to right and remove a vertex v from S if it has already occurred earlier in S.

So, the modified sequence of S is $[a, b, c, d, e, f, g, h, i, j, a]$.

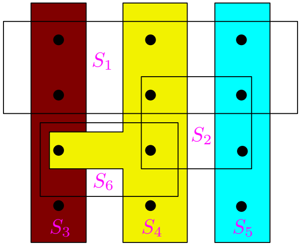


Step 5: Construct a Hamiltonian cycle from *a* to *a* by connecting consecutive vertices in the modified *S* by edges to form Euclidean traveling-salesman tour.

The above algorithm correctly computes a Euclidean traveling-salesman tour. How good is the solution?

Theorem: The approximation ratio of the Greedy Algorithm is 2.

Set-covering Problem



Set-covering problem: Given a finite family C of sets $S_1, \ldots, S_n$, the problem is to determine the minimum cardinality $A \subseteq C$ such that $\bigcup_{i \in A} S_i = \bigcup_{j=1}^{n} S_j$.

1. D. S. Johnson, *Approximation algorithms for combinatorial problems*, Journal of Computer and System Sciences, 9 (1974), 256-278.

Greedy Algorithm

1. Initialize the current family of sets by $S_1, \ldots, S_n$.

Greedy Algorithm

1. Initialize the current family of sets by $S_1, \ldots, S_n$.
2. Find a set S_i in the current family of sets which has the maximum number of elements.

Greedy Algorithm

1. Initialize the current family of sets by $S_1, \ldots, S_n$.
2. Find a set S_i in the current family of sets which has the maximum number of elements.
3. Remove the elements of S_i from the existing sets in the family.

Greedy Algorithm

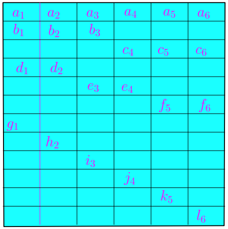
1. Initialize the current family of sets by $S_1, \ldots, S_n$.
2. Find a set S_i in the current family of sets which has the maximum number of elements.
3. Remove the elements of S_i from the existing sets in the family.
4. Remove S_i and all empty sets from the current family of sets.

Greedy Algorithm

1. Initialize the current family of sets by $S_1, \dots, S_n$.
2. Find a set S_i in the current family of sets which has the maximum number of elements.
3. Remove the elements of S_i from the existing sets in the family.
4. Remove S_i and all empty sets from the current family of sets.
5. Repeat this process till the current family of sets is empty.

The above algorithm correctly computes a set cover. How good is the solution?

Upper bound on approximation ratio



Sets corresponding to rows are $\{a_1, a_2, a_3, a_4, a_5, a_6\}$, $\{b_1, b_2, b_3\}$, $\{c_4, c_5, c_6\}$, $\{d_1, d_2\}$, $\{e_3, e_4\}$, $\{f_5, f_6\}$, $\{g_1\}$, $\{h_2\}$, $\{i_3\}$, $\{j_4\}$, $\{k_5\}$, $\{l_6\}$. Sets corresponding to columns are $\{a_1, b_1, d_1, g_1\}$, $\{a_2, b_2, d_2, h_2\}$, $\{a_3, b_3, e_3, i_3\}$, $\{a_4, c_4, e_4, j_4\}$, $\{a_5, c_5, f_5, k_5\}$, $\{a_6, c_6, f_6, l_6\}$.

Optimal cover chooses 6 sets corresponding to 6 columns (say, $k = 6$) but the greedy algorithm chooses sets corresponding to rows (i.e., $\frac{k}{6} + \frac{k}{3} + \frac{k}{2} + \frac{k}{1} \simeq k \log k$).

Theorem: Approximation ratio of the Greedy Algorithm is $O(\log(n))$.

Minimum number of guards

Let P be a polygon with or without holes. What is the minimum number of guards required for guarding a polygon P with or without holes?

Suppose, a positive integer k is given. Can it be decided in polynomial time whether k guards are sufficient to guard P?

The problem is NP-complete.

Theorem: The minimum vertex, point and edge guard problems for polygons with or without holes (including orthogonal polygons) are NP-hard.

Theorem: The minimum vertex and point guard problems for orthogonal polygons with or without holes are NP-hard.

1. D. T. Lee and A. K. Lin, *Computational Complexity of Art Gallery Problems*, IEEE Transactions on Information Theory, IT–32 (1986), 276–282.
2. J. O'Rourke and K. Supowit, *Some NP-hard polygon decomposition problems*, IEEE Transactions on Information Theory, IT-29 (1983), 181-190.
3. D. Schuchardt and H. Hecker, *Two NP-hard art-gallery problems for ortho-polygons*, Mathematical Logic Quarterly, 41 (1995), 261-267.
4. B.C. Liaw, N.F. Huang, R.C.T. Lee, *The minimum cooperative guards problem on k-spiral polygons*, Proceedings of the Canadian Conference on Computational Geometry, pp. 97–101, 1993.
5. M. Katz and G. Roisman, *On guarding the vertices of rectilinear domains*, Computational Geometry: Theory and Applications, 39 (2008), 219-228.

Approximation algorithms for minimum guard problems

1. S. K. Ghosh, *Approximation algorithms for art gallery problems*, Technical report no. JHU/EECS-86/15, Department of Electrical Engineering and Computer Science, The Johns Hopkins University, August 1986. Also in Proceedings of the Canadian Information Processing Society Congress, pp. 429-434, 1987. Running time: $O(n^5 \log n)$ time. Approximation ratio: $O(\log n)$.
2. S. K. Ghosh, *Approximation algorithms for art gallery problems in polygons*, Discrete Applied Mathematics, 158 (2010), 718-722. Recently, the running time has been improved to $O(n^4)$ for simple polygons and $O(n^5)$ for polygons with holes, keeping the approximation ratio same.
3. A. Efrat and S. Har-Peled, *Guarding galleries and terrains*, Information Processing Letters, 100 (2006), 238-245. Running time and the approximation ratio: (i) For simple polygons, $O(nc_{opt}^2 \log^4 n)$ expected time, and $O(\log c_{opt})$ approximation ratio, where c_{opt} is the number of vertices in the optimal solution. (ii) For polygons with h holes, $O(nhc_{opt}^3\ polylog\ n)$

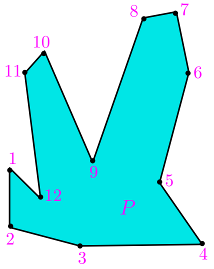
4. B. J. Nilsson, *Approximate Guarding of Monotone and Rectilinear Polygons*, Proceedings of the 32nd International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, Springer-Verlag, no. 3580, pp. 1362-1373, 2005. The paper gives polynomial time approximation algorithms (i) for monotone polygons and (ii) for simple orthogonal polygons. Approximation ratios are 12 and 96 respectively.
5. A. Deshpande, T. Kim, E. D. Demaine1 and S. E. Sarma, *A pseudopolynomial time O(log n)-approximation algorithm for art gallery problems* , Proceedings of the 10th International Workshop on Algorithms and Data Structures, LNCS, Springer-Verlag, no. 4619, pp. 163-174, 2007. Running time: Polynomial in n, the number of walls and the spread, where the spread can be exponential. Approximation ratio: $O(\log c_{opt})$.

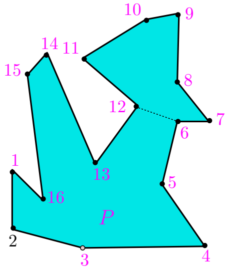
Heuristics for Stationary Guard Problems

- Recently, efforts are being made to design heuristics to solve stationary guard problems where the efficiency of these heuristics are evaluated by experimentation.
- These heuristics essentially follow Ghosh's method of first discretizing the entire region of a polygon and then using the minimum set-cover solution.
- However, these heuristics use different criteria for discretization or in choosing candidate sets.

1. Y. Amit and J. S. B. Mitchell and E. Packer, *Locating Guards for Visibility Coverage of Polygons*, Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX'07), SIAM, pp. 120-134, 2007.
2. M. C. Couto and P. J. de Rezende and C. C. de Souza, *An exact and efficient algorithm for the orthogonal art gallery problem*, Proceedings of the 20th Brazilian Symposium on Computer Graphics and Image Processing, pp. 87-94, 2007.
3. M. C. Couto and P. J. de Rezende and C. C. de Souza, *Experimental evaluation of an exact algorithm for the orthogonal art gallery problem*, Proceedings of the 7th International Workshop on Experimental Algorithms, LNCS, vol. 5038, pp. 101-113, Springer, Heidelberg, 2008.
4. M. C. Couto and P. J. de Rezende and C. C. de Souza, An IP solution to the art gallery problem, Proceedings of the 25th Annual ACM Symposium on Computational Geometry, pp. 88-89, 2009.

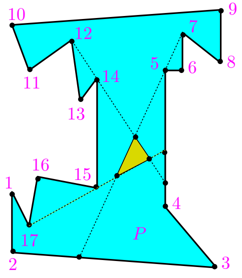
Vertex-guard problem

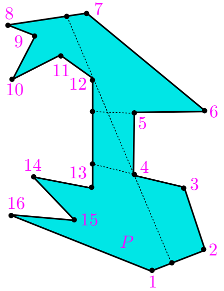




A simple polygon is called a *fan* if there exists a vertex that is visible from all points in the interior of the polygon.

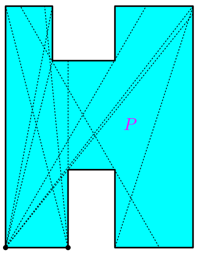
The vertex guard problem can be treated as a polygon decomposition problem in which the decomposition pieces are fans.

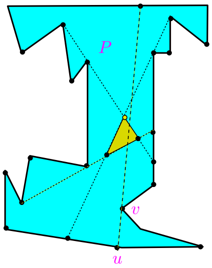




Vertices 7, 12 and 17 together can see the entire boundary of the polygon but the shaded region is not visible from any of these vertices.

Three fans (vertices 1, 4 and 7) are necessary to cover the polygon if only edge extensions are allowed, whereas two fans (vertices 1 and 7) suffice if we allow the boundary of convex components to be bounded by segments that contains any two vertices of the polygon.





The polygonal region of P is decomposed into convex components where every component is bounded by segments that contains any two vertices of the polygon.

Every convex component must lie in at least one of the fans chosen by the approximation algorithm.

Lemma: Every convex component is either totally visible or totally invisible from a vertex of P.

The problem of finding the minimum number of fans to cover P is same as the set-covering problem, where every fan is a set and convex components are elements of the set.

Vertex-guard algorithm

Step 1: Draw lines through every pair of vertices of P and compute all convex components $c_1, c_2, \dots, c_m$ of P. Let $C = (c_1, c_2, \dots, c_m)$, $N = (1, 2, \dots, n)$ and $Q = \emptyset$.

Vertex-guard algorithm

Step 1: Draw lines through every pair of vertices of P and compute all convex components $c_1, c_2, \ldots, c_m$ of P. Let $C = (c_1, c_2, \ldots, c_m)$, $N = (1, 2, \ldots, n)$ and $Q = \emptyset$.
Step 2: *For* $1 \leq j \leq n$, construct the set F_j by adding those convex components of P that are totally visible from the vertex v_j.

Vertex-guard algorithm

Step 1: Draw lines through every pair of vertices of P and compute all convex components $c_1, c_2, \ldots, c_m$ of P. Let $C = (c_1, c_2, \ldots, c_m)$, $N = (1, 2, \ldots, n)$ and $Q = \emptyset$.
Step 2: *For* $1 \leq j \leq n$, construct the set F_j by adding those convex components of P that are totally visible from the vertex v_j.
Step 3: Find $i \in N$ such that $|F_i| \geq |F_j|$ *for all* $j \in N$ and $i \neq j$.

Vertex-guard algorithm

Step 1: Draw lines through every pair of vertices of P and compute all convex components $c_1, c_2, \ldots, c_m$ of P. Let $C = (c_1, c_2, \ldots, c_m)$, $N = (1, 2, \ldots, n)$ and $Q = \emptyset$.
Step 2: *For* $1 \leq j \leq n$, construct the set F_j by adding those convex components of P that are totally visible from the vertex v_j.
Step 3: Find $i \in N$ such that $|F_i| \geq |F_j|$ *for all* $j \in N$ and $i \neq j$.
Step 4: Add i to Q and delete i from N.

Vertex-guard algorithm

Step 1: Draw lines through every pair of vertices of P and compute all convex components $c_1, c_2, \ldots, c_m$ of P. Let $C = (c_1, c_2, \ldots, c_m)$, $N = (1, 2, \ldots, n)$ and $Q = \emptyset$.
Step 2: *For* $1 \leq j \leq n$, construct the set F_j by adding those convex components of P that are totally visible from the vertex v_j.
Step 3: Find $i \in N$ such that $|F_i| \geq |F_j|$ *for all* $j \in N$ and $i \neq j$.
Step 4: Add i to Q and delete i from N.
Step 5: *For all* $j \in N$, $F_j := F_j - F_i$, and $C := C - F_i$.

Vertex-guard algorithm

Step 1: Draw lines through every pair of vertices of P and compute all convex components $c_1, c_2, \ldots, c_m$ of P. Let $C = (c_1, c_2, \ldots, c_m)$, $N = (1, 2, \ldots, n)$ and $Q = \emptyset$.
Step 2: *For* $1 \leq j \leq n$, construct the set F_j by adding those convex components of P that are totally visible from the vertex v_j.
Step 3: Find $i \in N$ such that $|F_i| \geq |F_j|$ *for all* $j \in N$ and $i \neq j$.
Step 4: Add i to Q and delete i from N.
Step 5: *For all* $j \in N$, $F_j := F_j - F_i$, and $C := C - F_i$.
Step 6: *If* $|C| \neq \emptyset$ *then goto* Step 3.

Vertex-guard algorithm

Step 1: Draw lines through every pair of vertices of P and compute all convex components $c_1, c_2, \ldots, c_m$ of P. Let $C = (c_1, c_2, \ldots, c_m)$, $N = (1, 2, \ldots, n)$ and $Q = \emptyset$.
Step 2: *For* $1 \leq j \leq n$, construct the set F_j by adding those convex components of P that are totally visible from the vertex v_j.
Step 3: Find $i \in N$ such that $|F_i| \geq |F_j|$ *for all* $j \in N$ and $i \neq j$.
Step 4: Add i to Q and delete i from N.
Step 5: *For all* $j \in N$, $F_j := F_j - F_i$, and $C := C - F_i$.
Step 6: *If* $|C| \neq \emptyset$ *then goto* Step 3.
Step 7: Output the set Q and Stop.

Vertex-guard algorithm

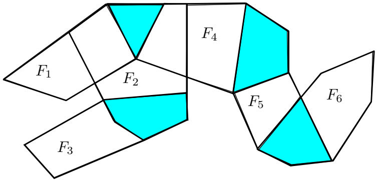
Step 1: Draw lines through every pair of vertices of P and compute all convex components $c_1, c_2, \ldots, c_m$ of P. Let $C = (c_1, c_2, \ldots, c_m)$, $N = (1, 2, \ldots, n)$ and $Q = \emptyset$.
Step 2: *For* $1 \leq j \leq n$, construct the set F_j by adding those convex components of P that are totally visible from the vertex v_j.
Step 3: Find $i \in N$ such that $|F_i| \geq |F_j|$ *for all* $j \in N$ and $i \neq j$.
Step 4: Add i to Q and delete i from N.
Step 5: *For all* $j \in N$, $F_j := F_j - F_i$, and $C := C - F_i$.
Step 6: *If* $|C| \neq \emptyset$ *then goto* Step 3.
Step 7: Output the set Q and Stop.

Theorem: The approximation algorithm for the minimum vertex guard problem in a polygon P of n vertices computes solutions that are at most $O(\log n)$ times the optimal. If P is a simple polygon, the approximation algorithm runs in $O(n^4)$ time. If P is a polygon with holes, the approximation algorithm runs in $O(n^5)$ time.

Edge-guard algorithm

Step 1: Draw lines through every pair of vertices of P and compute all convex components $c_1, c_2, \ldots, c_m$ of P. Let $C = (c_1, c_2, \ldots, c_m)$, $N = (1, 2, \ldots, n)$ and $Q = \emptyset$.
Step 2: *For* $1 \leq j \leq n$, construct the set E_j by adding those convex components of P that are totally visible from the edge e_j of P.
Step 3: Find $i \in N$ such that $|E_i| \geq |E_j|$ *for all* $j \in N$ and $i \neq j$.
Step 4: Add i to Q and delete i from N.
Step 5: *For all* $j \in N$, $E_j := E_j - E_i$, and $C := C - E_i$.
Step 6: *If* $|C| \neq \emptyset$ *then goto* Step 3.
Step 7: Output the set Q and Stop.
Theorem: For the minimum edge guard problem in an n-sided polygon P, an approximate solution can be computed which is at most $O(\log n)$ times the optimal. If P is a simple polygon, the approximation algorithm runs in $O(n^4)$ time. If P is a polygon with holes, the approximation algorithm runs in $O(n^5)$ time.

Any set consisting of arbitrary chosen convex components may not form a fan as every fan consists of contiguous convex components. Therefore, constructing any example where the greedy algorithm takes $O(\log n)$ times optimal does not seem to be possible.



Conjecture: (Ghosh 1986) Approximation algorithms are expected to yield solutions within a constant factor of the optimal.

Lower bound on approximation ratio

Regarding the lower bound on the approximation ratio for the problems of minimum vertex, point and edge guards in simple polygons, it has been shown that these problems are APX-hard using gap-preserving reductions from 5-OCCURRENCE-MAX-3-SAT.

This means that for each of these problems, there exists a constant $\epsilon > 0$ such that an approximation ratio of $1 + \epsilon$ cannot be guaranteed by any polynomial time approximation algorithm unless $P = NP$.

The above statement implies that there may be approximation algorithms for these problems whose approximation ratios are not small constants.

1. S. Eidenbenz C. Stamm and P. Widmayer, *Inapproximability Results for Guarding Polygons and Terrains*, Algorithmica, 31 (2000), 79-113.

On the other hand, for polygons with holes, these problems cannot be approximated by a polynomial time algorithm with ratio $((1-\epsilon)/12)(\mathit{ln}\ n)$ for any $(\epsilon > 0)$, unless $NP \subseteq TIME(n^{O(loglogn)})$. The results are obtained by using gap-preserving reductions from the SET COVER problem.

Open problems

Design approximation algorithms for vertex, edge and point guards problems in simple polygons which yield solutions within a constant factor of the optimal.