Approximation Algorithms and Inapproximability

Ana Paula Tomás

Department of Computer Science, Faculty of Sciences
Center of Mathematics
University of Porto, Portugal

January 2021

NP Optimization Problems (the class NPO)

Π is an optimization problem. The instances I of Π are a subset of $\Sigma^\star$ (i.e., encoded as a language over Σ). $val(S, I) \in \mathbb{Q}_0^+$ is the value of the feasible solution S and $\mathcal{S}(I)$ is the set of feasible solutions. The goal is to find $OPT(I) = \min_{S \in \mathcal{S}(I)} val(S, I)$ for a minimization problem, and $OPT(I) = \max_{S \in \mathcal{S}(I)} val(S, I))$ for a maximization problem.

Π is a NP optimization problem (Π is in NPO) if:

- given $x \in \Sigma^\star$, we can check if x is an instance of Π in $poly(|x|)$ time;
- $|S|$ is $poly(|I|)$, for each I and $S \in \mathcal{S}(I)$;
- there is a poly-time decision procedure that decides if $x \in \mathcal{S}(I)$, for I and $x \in \Sigma^\star$;
- $val(I, S)$ is a poly-time computable function.

$\mathcal{L}(\Pi, B) = \{I : OPT(I) \leq B\}$, the **decision version** of a NPO minimization problem Π, belongs to NP. When the decision version is NP-hard the optimization problem is called an **NP-hard optimization problem**. If Π is an NP-hard NPO problem, its decision version $\mathcal{L}(\Pi, B)$ is NP-complete.

NP Optimization Problems (the class NPO)

Π is an optimization problem. The instances I of Π are a subset of $\Sigma^\star$ (i.e., encoded as a language over Σ). $val(S, I) \in \mathbb{Q}_0^+$ is the value of the feasible solution S and $\mathcal{S}(I)$ is the set of feasible solutions. The goal is to find $OPT(I) = \min_{S\in\mathcal{S}(I)} val(S, I)$ for a minimization problem, and $OPT(I) = \max_{S\in\mathcal{S}(I)} val(S, I))$ for a maximization problem.

Π is a NP optimization problem (Π is in NPO) if:

- given $x \in \Sigma^\star$, we can check if x is an instance of Π in $poly(|x|)$ time;
- $|S|$ is $poly(|I|)$, for each I and $S \in \mathcal{S}(I)$;
- there is a poly-time decision procedure that decides if $x \in \mathcal{S}(I)$, for I and $x \in \Sigma^\star$;
- $val(I, S)$ is a poly-time computable function.

$\mathcal{L}(\Pi, B) = \{I : OPT(I) \leq B\}$, the **decision version** of a NPO minimization problem Π, belongs to NP. When the decision version is NP-hard the optimization problem is called an **NP-hard optimization problem**. If Π is an NP-hard NPO problem, its decision version $\mathcal{L}(\Pi, B)$ is NP-complete.

NP Optimization Problems (the class NPO)

Π is an optimization problem. The instances I of Π are a subset of $\Sigma^\star$ (i.e., encoded as a language over Σ). $val(S, I) \in \mathbb{Q}_0^+$ is the value of the feasible solution S and $\mathcal{S}(I)$ is the set of feasible solutions. The goal is to find $OPT(I) = \min_{S \in \mathcal{S}(I)} val(S, I)$ for a minimization problem, and $OPT(I) = \max_{S \in \mathcal{S}(I)} val(S, I)$) for a maximization problem.

Π is a NP optimization problem (Π is in NPO) if:

- given $x \in \Sigma^\star$, we can check if x is an instance of Π in $poly(|x|)$ time;
- $|S|$ is $poly(|I|)$, for each I and $S \in \mathcal{S}(I)$;
- there is a poly-time decision procedure that decides if $x \in \mathcal{S}(I)$, for I and $x \in \Sigma^\star$;
- $val(I, S)$ is a poly-time computable function.

$\mathcal{L}(\Pi, B) = \{I : OPT(I) \leq B\}$, the **decision version** of a NPO minimization problem Π, belongs to NP. When the decision version is NP-hard the optimization problem is called an **NP-hard optimization problem**. If Π is an NP-hard NPO problem, its decision version $\mathcal{L}(\Pi, B)$ is NP-complete.

Approximation Algorithms

Approximation ratio

The approximation ratio of an algorithm $\mathcal{A}$ for a **minimization problem** is $\alpha_{\mathcal{A}} = \max_I \frac{\mathcal{A}(I)}{OPT(I)}$, where $\mathcal{A}(I)$ is the value of the solution $\mathcal{A}$ returns for instance I. So, $\mathcal{A}(I) \leq \alpha_{\mathcal{A}} OPT(I)$. For a **maximization problem**, $\alpha_{\mathcal{A}} = \max_I \frac{OPT(I)}{\mathcal{A}(I)}$, so that $\mathcal{A}(I) \geq \frac{1}{\alpha_{\mathcal{A}}} OPT(I)$.

By this definition, $\alpha_{\mathcal{A}} \geq 1$ even for maximization problems. On an input of size n, the ratio $\alpha_{\mathcal{A}}$ can be a function $\alpha_{\mathcal{A}}(n)$. If the function is constant, i.e., does not depend on n, then $\mathcal{A}$ is a constant factor approximation algorithm.

The class APX and APX-hard problems

APX is the class of NPO problems for which there are constant factor *polynomial time* approximation algorithms.
An NPO problem is APX-hard if there is a constant $\epsilon > 0$ such that an approximation ratio of $1 + \epsilon$ cannot be guaranteed by any polynomial-time algorithm, unless P = NP.

Approximation Algorithms

Approximation ratio

The approximation ratio of an algorithm $\mathcal{A}$ for a **minimization problem** is $\alpha_{\mathcal{A}} = \max_I \frac{\mathcal{A}(I)}{OPT(I)}$, where $\mathcal{A}(I)$ is the value of the solution $\mathcal{A}$ returns for instance I. So, $\mathcal{A}(I) \leq \alpha_{\mathcal{A}} OPT(I)$. For a **maximization problem**, $\alpha_{\mathcal{A}} = \max_I \frac{OPT(I)}{\mathcal{A}(I)}$, so that $\mathcal{A}(I) \geq \frac{1}{\alpha_{\mathcal{A}}} OPT(I)$.

By this definition, $\alpha_{\mathcal{A}} \geq 1$ even for maximization problems. On an input of size n, the ratio $\alpha_{\mathcal{A}}$ can be a function $\alpha_{\mathcal{A}}(n)$. If the function is constant, i.e., does not depend on n, then $\mathcal{A}$ is a constant factor approximation algorithm.

The class APX and APX-hard problems

APX is the class of NPO problems for which there are constant factor *polynomial time* approximation algorithms.
An NPO problem is APX-hard if there is a constant $\epsilon > 0$ such that an approximation ratio of $1 + \epsilon$ cannot be guaranteed by any polynomial-time algorithm, unless P = NP.

Approximation Schemes

Approximation Scheme

An approximation scheme for an optimization problem Π is a family of $(1+\varepsilon)$-approximation algorithms A_ε for problem Π, over all $0 < \varepsilon < 1$.

Polynomial Time Approximation Scheme (PTAS)

A polynomial time approximation scheme for Π is an approximation scheme such that the time complexity of A_ε is polynomial in the input size, for all ε.

(the time complexity can be exponential in $1/\varepsilon$)

Fully Polynomial Time Approximation Scheme (FPTAS)

A polynomial time approximation scheme for Π is an approximation scheme such that the time complexity of A_ε is polynomial in the input size and also polynomial in $1/\varepsilon$, for all ε.

[Arora et al., FOCS'92] **If there is a PTAS for some APX-hard problem, P=NP.**

Approximation Schemes

Approximation Scheme

An approximation scheme for an optimization problem Π is a family of $(1+\varepsilon)$-approximation algorithms A_ε for problem Π, over all $0 < \varepsilon < 1$.

Polynomial Time Approximation Scheme (PTAS)

A polynomial time approximation scheme for Π is an approximation scheme such that the time complexity of A_ε is polynomial in the input size, for all ε.

(the time complexity can be exponential in $1/\varepsilon$)

Fully Polynomial Time Approximation Scheme (FPTAS)

A polynomial time approximation scheme for Π is an approximation scheme such that the time complexity of A_ε is polynomial in the input size and also polynomial in $1/\varepsilon$, for all ε.

[Arora et al., FOCS'92] **If there is a PTAS for some APX-hard problem, P=NP.**

Approximation Schemes

Approximation Scheme

An approximation scheme for an optimization problem Π is a family of $(1+\varepsilon)$-approximation algorithms A_ε for problem Π, over all $0 < \varepsilon < 1$.

Polynomial Time Approximation Scheme (PTAS)

A polynomial time approximation scheme for Π is an approximation scheme such that the time complexity of A_ε is polynomial in the input size, for all ε.

(the time complexity can be exponential in $1/\varepsilon$)

Fully Polynomial Time Approximation Scheme (FPTAS)

A polynomial time approximation scheme for Π is an approximation scheme such that the time complexity of A_ε is polynomial in the input size and also polynomial in $1/\varepsilon$, for all ε.

[Arora et al., FOCS'92] **If there is a PTAS for some APX-hard problem, P=NP.**

2-approximation algorithm for VERTEX COVER

VERTEXCOVER asks for a *minimum cardinality vertex-cover* of a given undirected graph $G = (V, E)$. A **vertex-cover** of G is a subset $S \subseteq V$ such that for each edge $(u, v) \in E$, either $u \in S$ or $v \in S$, or both.

- Given G and $k \in \mathbb{N}$ as input, deciding if G has a vertex-cover S of size $|S| \leq k$ is a well-known **NP-complete** problem.
- VERTEXCOVER is **APX-complete** (i.e., APX-hard and belongs to APX).

2-approximation for VERTEXCOVER

$S := \emptyset$
while $(E \neq \emptyset)$ **do**
  remove an edge $e = (u, v)$ *from* E
  remove all edges incident to u *or* v
  $S := S \cup \{u, v\}$
return S

Proof: The algorithm yields a vertex-cover S in poly-time. The selected edges e do not share endpoints (they form a matching M of G). If $S^\star$ is an optimal vertex-cover, $|S^\star| \geq |M| = |S|/2$, since each edge in M is covered by a distinct vertex of $S^\star$. Thus, $|S| \leq 2|S^\star|$. □

2-approximation algorithm for VERTEX COVER

VERTEXCOVER asks for a *minimum cardinality vertex-cover* of a given undirected graph $G = (V, E)$. A **vertex-cover** of G is a subset $S \subseteq V$ such that for each edge $(u, v) \in E$, either $u \in S$ or $v \in S$, or both.

- Given G and $k \in \mathbb{N}$ as input, deciding if G has a vertex-cover S of size $|S| \leq k$ is a well-known **NP-complete** problem.
- VERTEXCOVER is **APX-complete** (i.e., APX-hard and belongs to APX).

2-approximation for VERTEXCOVER

$S := \emptyset$
while *($E \neq \emptyset$)* **do**
  remove an edge $e = (u, v)$ from E
  remove all edges incident to u or v
  $S := S \cup \{u, v\}$
return S

Proof: The algorithm yields a vertex-cover S in poly-time. The selected edges e do not share endpoints (they form a matching M of G). If $S^\star$ is an optimal vertex-cover, $|S^\star| \geq |M| = |S|/2$, since each edge in M is covered by a distinct vertex of $S^\star$. Thus, $|S| \leq 2|S^\star|$. □

2-approximation for VERTEX COVER (by LP rounding)

Consider VERTEXCOVER as a *boolean linear programming problem*.

$$\text{minimize } \sum_{v\in V} x_v$$
$$\begin{cases} x_u + x_v \geq 1, \text{ for all } (u,v) \in E \\ x_v \in \{0,1\}, \text{ for all } v \in V \end{cases}$$

- It is known that its **linear relaxation**, i.e., the problem we obtain if we replace the domain constraint $x_v \in \{0,1\}$ by $x_v \in [0,1]$, for all v, can be solved in polynomial time. Let $x^\star$ be its optimal solution.
- The boolean solution given by $x_v = 1$ if $x_v^\star \geq 1/2$ and $x_v = 0$ if $x_v^\star < 1/2$, for all $v \in V$, is a **feasible solution to VERTEXCOVER**. In fact, for each edge (u,v), either $x_u^\star \geq 1/2$ or $x_v^\star \geq 1/2$ (otherwise, $x_u^\star + x_v^\star \geq 1$ would be violated). So, $S = \{v \in V \mid x_v = 1\}$ is a vertex-cover.
- If $S^\star$ is a minimum vertex-cover, then $|S| \leq 2|S^\star|$. In fact, by construction $|S| \leq 2\sum_{v\in V} x_v^\star$, and $\sum_{v\in V} x_v^\star \leq |S^\star|$ because the optimal solution of the relaxation cannot be worse then the value of any other of its solutions (therefore, of the boolean solution induced by $S^\star$).

Minimum vertex-cover is APX-hard

Some known inapproximability bounds for minimum vertex cover on graphs:

- It is hard to approximate to within $2-\varepsilon$, for any constant $\varepsilon > 0$, if the *unique games conjecture* is true (S.Khot & O.Regev, 2008).
- Håstad (J.ACM, 2001) showed that it is NP-hard to approximate within constant factors less than 7/6. This factor was improved by Dinur and Safra (STOC'2002) to $10\sqrt{5}-21 \approx 1.36$.
- If the graph has degree bounded by 3, it cannot be approximated within $100/99-\epsilon$, for $\epsilon > 0$, unless P=NP; $53/52-\epsilon$ if the degree is bounded by 4. [Chlebík & Chlebíková, FCT 2003]. Improved to $1.0101215-\epsilon$ and $1.0194553-\epsilon$. (Chlebík & Chlebíková, TCS 354, 320–338, 2006);

Unique games conjecture: `https://en.wikipedia.org/wiki/Unique_games_conjecture`

2-approximation for the METRIC TSP

Traveling Salesman Problem (TSP): given a **complete** undirected weighted graph $G = (V, E, d)$ such that $d : E \to \mathbb{R}_0^+$, find an Hamiltonian cycle $C^\star$ in G such that $d(C^\star) = \sum_{e \in C^\star} d(e)$ is minimum.

The **Metric TSP** is TSP with triangle inequality, i.e., the cost function d satisfies $d(x, y) \leq d(x, z) + d(z, y)$, for all $x, y, z \in V$.

The approximation algorithms we will introduce for the metric TSP make use of the following property.

Property

Given any walk $\gamma = (x_1, x_2, x_3, \ldots, x_{p-1}, x_p)$, with $p \geq 3$, we can replace (x_{i-1}, x_i, x_{i+1}) by (x_{i-1}, x_{i+1}), to obtain a walk γ' from x_1 to x_p such that $d(\gamma) \leq d(\gamma')$, with $1 < i < p$.

Proof: (x_{i-1}, x_{i+1}) is an edge of G, because G is a complete graph. Thus, γ' is a walk in G. By the triangle inequality,

$d(\gamma') = d(\gamma) + d(x_{i-1}, x_{i+1}) - (d(x_{i-1}, x_i) + d(x_i, x_{i+1})) \leq d(\gamma)$.

2-approximation for the METRIC TSP

Traveling Salesman Problem (TSP): given a **complete** undirected weighted graph $G = (V, E, d)$ such that $d : E \to \mathbb{R}_0^+$, find an Hamiltonian cycle $C^\star$ in G such that $d(C^\star) = \sum_{e \in C^\star} d(e)$ is minimum.

The **Metric TSP** is TSP with triangle inequality, i.e., the cost function d satisfies $d(x, y) \leq d(x, z) + d(z, y)$, for all $x, y, z \in V$.

The approximation algorithms we will introduce for the metric TSP make use of the following property.

Property

Given any walk $\gamma = (x_1, x_2, x_3, \ldots, x_{p-1}, x_p)$, with $p \geq 3$, we can replace (x_{i-1}, x_i, x_{i+1}) by (x_{i-1}, x_{i+1}), to obtain a walk γ' from x_1 to x_p such that $d(\gamma) \leq d(\gamma')$, with $1 < i < p$.

Proof: (x_{i-1}, x_{i+1}) is an edge of G, because G is a complete graph. Thus, γ' is a walk in G. By the triangle inequality,

$$d(\gamma') = d(\gamma) + d(x_{i-1}, x_{i+1}) - (d(x_{i-1}, x_i) + d(x_i, x_{i+1})) \leq d(\gamma).$$

2-approximation for the METRIC TSP (cont.)

A 2-approximation algorithm for Metric TSP

Construct a minimum spanning tree (MST) $T^\star$ of G; Double every edge of T to get an **eulerian graph**; Find an Eulerian tour W on this graph (e.g., induced by a traversal of T in depth-first order); Let C be the list of vertices obtained by deleting all duplicates in W (keep the last vertex); Return C.

Proof: C is an Hamiltonean cycle in G and, by the triangle inequality, $d(C) \leq 2d(T^\star)$ (to remove a duplicate, we replaced two edges in W by a single one in C). If $C^\star$ is the optimal cycle, $d(C) \leq 2d(C^\star)$ because if we delete an edge e from $C^\star$ we get a spanning tree T with $d(T^\star) \leq d(T) = d(C^\star) - d(e) \leq d(C^\star)$. Therefore, $d(C) \leq 2d(T^\star) \leq 2d(C^\star)$. □

2-approximation for the METRIC TSP (cont.)

A 2-approximation algorithm for Metric TSP

Construct a minimum spanning tree (MST) $T^\star$ of G; Double every edge of T to get an **eulerian graph**; Find an Eulerian tour W on this graph (e.g., induced by a traversal of T in depth-first order); Let C be the list of vertices obtained by deleting all duplicates in W (keep the last vertex); Return C.

Proof: C is an Hamiltonean cycle in G and, by the triangle inequality, $d(C) \leq 2d(T^\star)$ (to remove a duplicate, we replaced two edges in W by a single one in C). If $C^\star$ is the optimal cycle, $d(C) \leq 2d(C^\star)$ because if we delete an edge e from $C^\star$ we get a spanning tree T with $d(T^\star) \leq d(T) = d(C^\star) - d(e) \leq d(C^\star)$. Therefore, $d(C) \leq 2d(T^\star) \leq 2d(C^\star)$. □

1.5-approximation for the metric TSP

Christofides algorithm for the Metric TSP:

- Find a minimum spanning tree $T^\star$ of G.
- Instead of duplicating all edges of $T^\star$ (to form an Eulerian circuit), take the set of nodes $\mathcal{O}$ that have odd degree. (Recall that a graph has an Eulerian circuit iff every node has even degree). For the set $\mathcal{O}$, find a matching $M^\star$ of minimum weight in G. (Note that $M^\star$ exists because $|O|$ is always even and G is complete).
- Add M to $T^\star$ to obtain a subgraph G' of G, with V as vertex set and that has an Eulerian circuit. Find an Eulerian circuit $\mathcal{C}_e$ in G'.
- Visit $\mathcal{C}_e$, eliminating duplicates to produce an Hamiltonean cycle $\mathcal{C}$.

Theorem: Every step can be carried out in a polynomial time and $d(\mathcal{C}) \leq 1.5d(\mathcal{C}^\star)$.

A sketch of the proof: *Given an optimal solution $C^\star$ to the TSP, we can start from a vertex in $\mathcal{O}$ and remove from $C^\star$ all the vertices in $V \setminus \mathcal{O}$. This gives a cycle $\mathcal{C}_\mathcal{O}$ such that $d(\mathcal{C}^\star) \geq d(\mathcal{C}_\mathcal{O})$, by the triangle inequality. $\mathcal{C}_\mathcal{O}$ consists of two disjoint matchings, say M_1 and M_2, for the nodes in O. Since $M^\star$ is minimum, $d(\mathcal{C}_\mathcal{O}) = d(M_1) + d(M_2) \geq 2d(M^\star)$. Therefore, $d(M^\star) \leq 0.5d(\mathcal{C}_\mathcal{O}) \leq 0.5d(\mathcal{C}^\star)$. Moreover, $d(T^\star) \leq d(\mathcal{C}^\star)$ (when we remove an edge from $\mathcal{C}^\star$, we get a supporting tree). Thus, $d(\mathcal{C}) \leq d(T^\star) + d(M^\star) \leq 1.5d(\mathcal{C}^\star)$.* □

Inapproximability of the general TSP

Traveling Salesman Problem (TSP): given a complete undirected weighted graph $G = (V, E, d)$ such that $d : E \to \mathbb{R}_0^+$, find an Hamiltonian cycle $C^\star$ in G such that $d(C^\star) = \sum_{e \in C^\star} d(e)$ is minimum.

Inapproximability of the general TSP

If $P \neq NP$, there is no polynomial time $\alpha(n)$-approximation algorithm for TSP, for any polynomial time computable function $\alpha(n)$, where $n = |V|$.

Proof: By reduction from the HAMILTONIAN CYCLE PROBLEM. Let $G = (V, E)$ be an undirected graph. Construct a complete graph $G' = (V, E')$ from V, and define $d(e) = 1$ if $e \in E$ and $d(e) = \alpha(n)n + 1$ if $e \notin E$, for each $e \in E'$.

Suppose $\mathcal{A}$ is a polynomial time $\alpha(n)$-approximation algorithm for TSP. Run $\mathcal{A}$ on G'. If G has an Hamiltonean cycle $C^\star$, then $\mathcal{A}$ must return a cycle C in G' such that $d(C) \leq \alpha(n)d(C^\star) = \alpha(n)n$. If G has no Hamiltonean cycle, then $\mathcal{A}$ must return a cycle C in G' such that $d(C) \geq (\alpha(n)n + 1) + (n - 1) > \alpha(n)n$, because C must have at least one edge $e \notin E$ (i.e., with $d(e) = \alpha(n)n + 1$). Thus, we can use $\mathcal{A}$ to decide the existence of an hamiltonian cycle in G. Therefore, $\mathcal{A}$ cannot exist if $P \neq NP$. □

Inapproximability of the general TSP

Traveling Salesman Problem (TSP): given a complete undirected weighted graph $G = (V, E, d)$ such that $d : E \to \mathbb{R}_0^+$, find an Hamiltonian cycle $C^\star$ in G such that $d(C^\star) = \sum_{e \in C^\star} d(e)$ is minimum.

Inapproximability of the general TSP

If $P \neq NP$, there is no polynomial time $\alpha(n)$-approximation algorithm for TSP, for any polynomial time computable function $\alpha(n)$, where $n = |V|$.

Proof: By reduction from the HAMILTONIAN CYCLE PROBLEM. Let $G = (V, E)$ be an undirected graph. Construct a complete graph $G' = (V, E')$ from V, and define $d(e) = 1$ if $e \in E$ and $d(e) = \alpha(n)n + 1$ if $e \notin E$, for each $e \in E'$.

Suppose $\mathcal{A}$ is a polynomial time $\alpha(n)$-approximation algorithm for TSP. Run $\mathcal{A}$ on G'. If G has an Hamiltonean cycle $C^\star$, then $\mathcal{A}$ must return a cycle C in G' such that $d(C) \leq \alpha(n)d(C^\star) = \alpha(n)n$. If G has no Hamiltonean cycle, then $\mathcal{A}$ must return a cycle C in G' such that $d(C) \geq (\alpha(n)n + 1) + (n - 1) > \alpha(n)n$, because C must have at least one edge $e \notin E$ (i.e., with $d(e) = \alpha(n)n + 1$). Thus, we can use $\mathcal{A}$ to decide the existence of an hamiltonian cycle in G. Therefore, $\mathcal{A}$ cannot exist if $P \neq NP$. □

BIN PACKING: Approximation and Inapproximability

- NP-hardness by reduction from PARTITION
- Inapproximabiity to $3/2 - \varepsilon$, for $\varepsilon > 0$, if P=NP, by reduction from PARTITION
- Belongs to APX: 2-approximation algorithms (proof for First Fit Strategy); mention 3/2-approximation for “first fit decreasing”

Please refer to:

- http://ac.informatik.uni-freiburg.de/lak_teaching/ws11_12/combopt/notes/bin_packing.pdf
- https://sites.cs.ucsb.edu/~suri/cs130b/BinPacking
- http://ac.informatik.uni-freiburg.de/lak_teaching/ws07_08/algotheo/Slides/13_bin_packing.pdf

$O(\log n)$-approximation for the SET COVER

MIN-SET-COVER Given a collection $\mathcal{F}$ of nonempty subsets of $A = \{a_1, \ldots, a_n\}$, find a covering $C^\star \subseteq \mathcal{F}$ of A such that $|C^\star|$ is minimum (if we consider all possible coverings $C \subseteq \mathcal{F}$).

GREEDY APPROXIMATION ALGORITHM: while there are uncovered elements, selects the set that covers the maximum number of uncovered elements.

$O(\log n)$-approximation for the SET COVER (cont.)

Lemma

Suppose there are k sets covering everything. After t choices, the greedy algorithm has at most $(1 - 1/k)^t$ fraction uncovered.

Proof:

If $\mathcal{C}$ is a covering with $|\mathcal{C}| = k$, there is at least a set C_i in $\mathcal{C}$ such that $|C_i| \geq n/k$ (Pigeon's hole principle).

Since the first set selected by the **greedy** algorithm, say F_1, must have at least $|C_i|$ elements, there remain at most $n - n/k$ elements uncovered after the first iteration, i.e., $(1 - 1/k)n$ elements uncovered.

Let $\mathcal{F}' = \{F_j \setminus F_1 \mid F_j \in \mathcal{F}, F_j \setminus F_1 \neq \emptyset\}$ and $\mathcal{C}' = \{C_i \setminus F_1 \mid C_i \in \mathcal{C}, C_i \setminus F_1 \neq \emptyset\}$.

Clearly, $\mathcal{C}' \subseteq \mathcal{F}'$ and $|\mathcal{C}'| \leq k$ and we can use $\mathcal{C}'$ to cover $A \setminus F_1$. If we note that $(1 - 1/|\mathcal{C}'|) \leq (1 - 1/k)$, the result follows. $\square$

$O(\log n)$-approximation for the SET COVER (cont.)

Lemma

Suppose there are k sets covering everything. After t choices, the greedy algorithm has at most $(1 - 1/k)^t$ fraction uncovered.

Proposition

The greedy algorithm is a $(1 + \ln n)$ approximation algorithm.

Proof: Let $k^\star$ be the optimal value. Once we have $< 1/n$ fraction uncovered, we are done.

Since $e^x = \sum_{n \in \mathbb{N}} x^n/n!$, we have $1 - 1/k^\star < e^{-1/k^\star}$ and $(1 - 1/k^\star)^t < (e^{-1/k^\star})^t$.

Thus, $e^{-t/k^\star} < 1/n$ if $t > (\ln n)\, k^\star$.

So, the greedy algorithm performs at most $\lfloor (\ln n) k^\star \rfloor + 1$ iterations (and adds a set to the covering per iteration). Hence, $|\mathcal{C}_{greedy}| \leq (1 + \ln n) k^\star$. $\square$

$O(\log n)$-approximation for the SET COVER (cont.)

It can be proved in fact that:

Theorem: GREEDY-SET-COVER is a polynomial time $\alpha - \mathit{approximation}$ algorithm, where

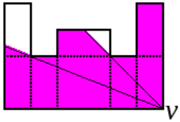
$$\alpha \quad = \quad H\left(max\left\{|S| \,:\, S \in F\right\}\right) \tag{2}$$

and $H(d)$ = the d^{th} harmonic number, which is equal to $1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{d} = \sum_{i=1}^{d} \frac{1}{i} = \log d + O(1)$ (from equation (A.7) in Appendix A).

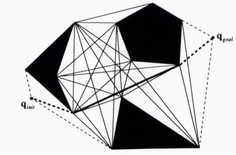
Check CLRS or https://www.cs.dartmouth.edu/~ac/Teach/CS105-Winter05/Notes/wan-ba-notes.pdf

Art Gallery Problems (AGP) – Guarding an art gallery

The region visible to v



Shortest Path / Visibility graph



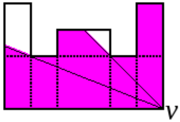
- **Visibility** is central to many areas: sensor networks, wireless networks, security and surveillance, and architectural design.
- An art gallery can be viewed as a polygon with or without holes.

The classical Art Gallery Problem by Victor Klee (1973)

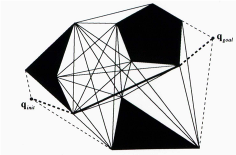
How many guards are always sufficient to guard any polygon with n vertices (with a 360° view, unlimited range)?

Art Gallery Problems (AGP) – Guarding an art gallery

The region visible to v



Shortest Path / Visibility graph



- **Visibility** is central to many areas: sensor networks, wireless networks, security and surveillance, and architectural design.
- An art gallery can be viewed as a polygon with or without holes.

The classical Art Gallery Problem by Victor Klee (1973)

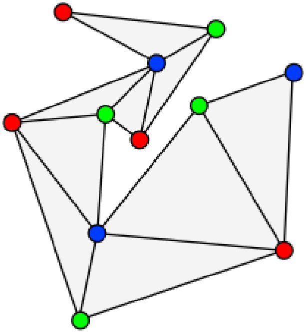
How many guards are always sufficient to guard any polygon with n vertices (with a 360° view, unlimited range)?

Chvatal's art gallery theorem (1975)

To cover a polygon of n vertices, $\lfloor \frac{n}{3} \rfloor$ stationary guards are always sufficient (and occasionally necessary).

"*A proof from* **THE BOOK**" by **Fisk** (1978):

- The polygon may be partitioned into $n-2$ triangles by adding $n-3$ internal diagonals.
  The dual graph of a triangulated simple polygon is a tree.
- The triangulation graph can always be 3-coloured. (adjacent vertices must have distinct colour)
- Vertices having the same colour form a guard set.
- One of the colours is used by at most $\lfloor n/3 \rfloor$ vertices. Place guards at these vertices.



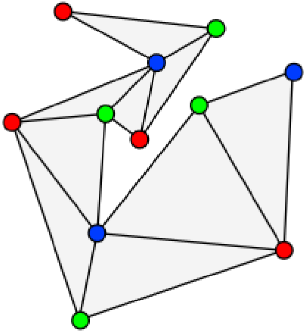
Chvatal's art gallery theorem (1975)

To cover a polygon of n vertices, $\lfloor \frac{n}{3} \rfloor$ stationary guards are always sufficient (and occasionally necessary).

"*A proof from* **THE BOOK**" by **Fisk** (1978):

- The polygon may be partitioned into $n-2$ triangles by adding $n-3$ internal diagonals.

  The dual graph of a triangulated simple polygon is a tree.
- The triangulation graph can always be 3-coloured. (adjacent vertices must have distinct colour)
- Vertices having the same colour form a guard set.
- One of the colours is used by at most $\lfloor n/3 \rfloor$ vertices. Place guards at these vertices.



For orthogonal polygons (Kahn, Klawe and Kleitman, 1980, O'Rourke, 1983)

To cover a polygon of n vertices, $\lfloor \frac{n}{4} \rfloor$ stationary guards are always sufficient (and occasionally necessary).

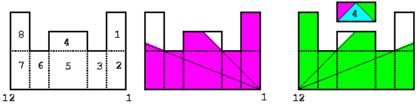
Extensions

- Polygons with or without holes;
- Different types of guards: **stationary** guards (point guards, vertex guards), **mobile** guards (edge guards), ...;
- Distinct notions of visibility: (un)limited range, 2π or α-view ($\pi/2$- or π-floodlights), ...

References: books by O'Rourke, Ghosh...; surveys by Shermer, Urrutia...; two handbooks; several papers

Some Art Gallery theorems:

- Any polygon with n vertices and h holes can always be guarded with $\lfloor \frac{n+2h}{3} \rfloor$ *vertex* guards. Conjecture (Shermer): $\lfloor \frac{n+h}{3} \rfloor$ (Still open for $h > 1$)
- $\lceil \frac{n+h}{3} \rceil$ *point* guards are always sufficient and occasionally necessary.
- To guard an orthogonal polygon with n vertices and h holes, $\lfloor \frac{n}{4} \rfloor$ *point* guards or $\lfloor \frac{n}{3} \rfloor$ *vertex* guards are always sufficient.
- Always sufficient and occasionally necessary
    - $\lfloor \frac{n}{4} \rfloor$ mobile guards for a n-vertex simple polygon;
    - $\lfloor \frac{3n+4}{16} \rfloor$ mobile or edge guards for a n-vertex orthogonal polygon;
    - $\lfloor \frac{3n+4h+4}{16} \rfloor$ mobile guards for an orthogonal polygon with n vertices and h holes.
- ...



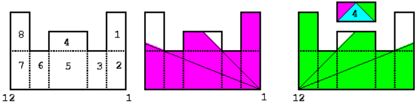
Stationary guards, unlimited visibility range, magnitude 2π. Two points p and q in P see each other if $\overline{pq} \cap Ext(P) = \emptyset$.

How many guards are always sufficient?

Two classical AGP theorems for n-vertex simple polygons: $\lfloor n/3 \rfloor$ guards are sufficient and occasionally necessary [Chvátal, 1975]; $\lfloor n/4 \rfloor$ for orthogonal polygons [Kahn, Klawe & Kleitman, 1983].

What is the fewest number needed for an given polygon P?

NP-hard [Lee & Lin, 1986], even for ortho-polygons [Schuchardt & Hecker, 1995]. APX-hard [Eidenbenz et al., 2001].
Could it be solved exactly in poly-time for some subclasses?



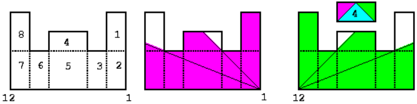
Stationary guards, unlimited visibility range, magnitude 2π. Two points p and q in P see each other if $\overline{pq} \cap Ext(P) = \emptyset$.

How many guards are always sufficient?

Two classical AGP theorems for n-vertex simple polygons: $\lfloor n/3 \rfloor$ guards are sufficient and occasionally necessary [Chvátal, 1975]; $\lfloor n/4 \rfloor$ for orthogonal polygons [Kahn, Klawe & Kleitman, 1983].

What is the fewest number needed for an given polygon P?

NP-hard [Lee & Lin, 1986], even for ortho-polygons [Schuchardt & Hecker, 1995]. APX-hard [Eidenbenz et al., 2001].
Could it be solved exactly in poly-time for some subclasses?



Stationary guards, unlimited visibility range, magnitude 2π. Two points p and q in P see each other if $\overline{pq} \cap Ext(P) = \emptyset$.

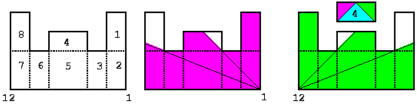
How many guards are always sufficient?

Two classical AGP theorems for n-vertex simple polygons: $\lfloor n/3 \rfloor$ guards are sufficient and occasionally necessary [Chvátal, 1975]; $\lfloor n/4 \rfloor$ for orthogonal polygons [Kahn, Klawe & Kleitman, 1983].

What is the fewest number needed for an given polygon P?

NP-hard [Lee & Lin, 1986], even for ortho-polygons [Schuchardt & Hecker, 1995]. APX-hard [Eidenbenz et al., 2001].

Could it be solved exactly in poly-time for some subclasses?



Stationary guards, unlimited visibility range, magnitude 2π. Two points p and q in P see each other if $\overline{pq} \cap Ext(P) = \emptyset$.

How many guards are always sufficient?

Two classical AGP theorems for n-vertex simple polygons: $\lfloor n/3 \rfloor$ guards are sufficient and occasionally necessary [Chvátal, 1975]; $\lfloor n/4 \rfloor$ for orthogonal polygons [Kahn, Klawe & Kleitman, 1983].

What is the fewest number needed for an given polygon P?

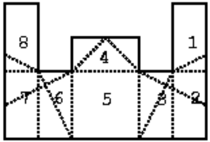
NP-hard [Lee & Lin, 1986], even for ortho-polygons [Schuchardt & Hecker, 1995]. APX-hard [Eidenbenz et al., 2001].
Could it be solved exactly in poly-time for some subclasses?

$O(\log n)$-approximation algorithm for MVG by Ghosh

MVG is **NP-hard** optimization problem [Lee & Lin, 1986], even for ortho-polygons [Schuchardt & Hecker, 1995].
Can we find approximate solutions with provable quality?

- The problem is APX-hard [Eidenbenz et al., 2001].
- The **algorithm by Ghost** (1987, 2010):

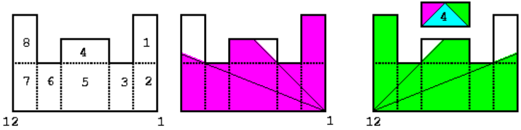
– Consider the decomposition induced by the visibility regions to reduce MINIMUM VERTEX GUARD to MINIMUM SET COVER;
– The **greedy algorithm** for MINIMUM SET COVER gives approximation ratio $O(\log n)$.
– Running time: $\mathcal{O}(n^5 \log n)$, improved to $\mathcal{O}(n^4)$ for simple polygons and $\mathcal{O}(n^5)$ for polygons with holes.



An anytime algorithm for MVG

MINIMUM VERTEX GUARD (MVG): What is the fewest number of vertex guards needed for an given polygon P?

[Tomás, Bajuelos & Marques (2003, 2006)]: MVG by sucessive approximations.



Transform MVG instances into MINIMUM SET COVER using a partition of P. Refine the initial partition to tight upper and lower bounds for $OPT(P)$:

$$OPT_{\square}(\Gamma_i) \leq OPT_{\square}(\Gamma_{i+1}) \leq OPT(P) \leq OPT_{\square}(\Pi_{i+1}) \leq OPT_{\square}(\Pi_i),$$

where Γ_i is the set of pieces of the current partition Π_i that are known to be not visible by sections, up to iteration i.

An anytime algorithm for MVG (cont.)

MINVERTEXGUARD(P)

Π := DECOMPOSE(P)
(each piece must be $\square$-visible to at least one vertex)
Compute G_v^t, G_v^s, for all vertices v
Compute G_R^t and G_R^s for all $R \in \Pi$
Γ := Γ_0^{Π}
while $(OPT_{\square}(\Gamma) < OPT_{\square}(\Pi))$ **do**
  Γ, Π := REFINE(Π).

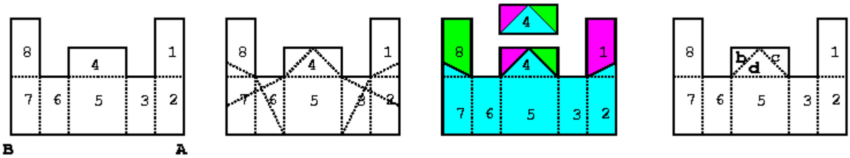
- $\square$-visibility disallows cooperation: a guard $\square$-sees a piece only if it sees it completely. $OPT_{\square}(\Pi)$ optimal number for Π under $\square$-visibility.
- G_v^t, G_v^s the pieces that vertex v sees totally and partially; G_R^t, G_R^s the vertices that see piece R totally and partially.
- $\Gamma_0 \subseteq \{R \mid R \text{ is not visible by sections}\} \subseteq \Pi$
(Γ the pieces that we know already that cannot be guarded in cooperation)

MVG by Sucessive Approximations

For the sequence $(\Gamma_i, \Pi_i)_{i\geq 0}$, it holds:

$$OPT_\square(\Gamma_i) \leq OPT_\square(\Gamma_{i+1}) \leq OPT(P) \leq OPT_\square(\Pi_{i+1}) \leq OPT_\square(\Pi_i)$$

An “anytime algorithm”: at each iteration, can return a solution; if it is not optimal, it can find better solutions if we let the algorithm continue to run. Not an approximation algorithm (see below).



$\Gamma_0 = \{R_1, R_2, R_5, R_7, R_8\}$ and $OPT_\square(\Pi_0) = 3 > 2 = OPT_\square(\Gamma_0)$.
By refining Piece 4 (i.e., R_4), we get $\Gamma_1 = \{R_1, R_2, R_5, R_7, R_8, R_b, R_c, R_d\}$ and, when we solve optimization problems (using a solver), we get $OPT_\square(\Pi_1) = OPT_\square(\Gamma_1) = 2$.