Introduction to heuristics with performance guarantee

Christian Artigues

LAAS-CNRS

June 16, 2011

Discrete (or combinatorial) optimization

Discrete (or combinatorial) optimization

Find the minimum (of the maximum) of a function f defined on a discrete set $\mathcal{X}$.

$\min f(x), x \in \mathcal{X}$

Typical discrete optimization problems (1/8)

Traveling salesman problem

A traveling salesman must cover n cities starting from city 1 and returning to the starting city. c_{ij} is the travel cost from city i to city j. What is the minimum cost tour ?

Typical discrete optimization problems (2/8)

Knapsack problem

A company has a total budget B for funding a subset of n projets. Each project i has a cost c_i and a profit v_i. Which projects should the company select to maximize its total income ?

Typical discrete optimization problems (3/8)

Set covering problem

Communications on m regions are covered by antennas to be installed on n predefined sites. Each site i has an installation cost c_i and may cover a set of regions S_i. Which antennas should be installed to cover the set of regions at minimal cost ?

Typical discrete optimization problems (4/8)

Warehouse location problem

A company delivers goods to m customer and may use to that prupose n possible warehouses. Opening warehouse j yields a fixed cost f_j and transportation of an order to customer i from warehouse j has a cost c_{ij}. Which warehouses should be opened and what is the optimal delivery plan to minimize total cost ?

Typical discrete optimization problems (5/8)

One-machine scheduling problem

An assembly line must assemble n products. Each product i has an assembly duration p_i, the components to be assembled arrive at the assembly line at a known date r_i and a due date d_i is negociated with the customer for each product. Under the constraint that the assembly line may assemble a single product at a given time, what are the start times of the assembly operations that minimize the maximum lateness ?

Typical discrete optimization problems (6/8)

Assignment problem

n jobs have to be performed by n employees. As some employees are more experienced than others on doing certain jobs, c_{ij} denotes the time needed by employee i to perform job j. What is the job/employee assignment that minimizes the total work duration ?

Typical discrete optimization problems (7/8)

Shortest path problem

A road network is represented by a directed graph $G = (V, A)$. Direct transportation of a location i to a location j is possible if $(i, j) \in A$ and has a duration c_{ij}. Given two locations s and t, what is the minimum cost path from s to t ?

Typical discrete optimization problems (8/8)

Minimum cost flow problem

A transportation network is represented by a directed graph $G = (V, A)$. Each node $i \in V$ has a value b_i, corresponding to a production amount ($b_i > 0$), a demand ($b_i < 0$) or a transit ($b_i = 0$) of goods. Each arc $(i, j) \in A$ represents a transportation resource with a limited capacity h_{ij} and a unit cost c_{ij}. What is the transportation plan satisfying all demands at minimal cost ?

Easy and hard optimization problems

1. Polynomial problems
    - Assignment problem
    - Shortest path problem
    - Minimum cost flow problem
2. Weakly NP-hard
    - Knapsack problem
3. Strongly NP-hard
    - Traveling salesman problem
    - Set covering problem
    - Warehouse location problem
    - One-machine scheduling problem

Heuristics may be needed for the last two categories (and even for the first one !!)

Heuristics: definition

Definition (from Wikipedia)

Heuristic refers to experience-based techniques for problem solving, learning, and discovery. Heuristic methods are used to speed up the process of finding a good enough solution, where an exhaustive search is impractical. Examples of this method include using a "rule of thumb", an educated guess, an intuitive judgment, or common sense. In more precise terms, heuristics are strategies using readily accessible, though loosely applicable, information to control problem solving in human beings and machines.

In this lecture

- A heuristic is an algorithm for solving (a specific or general) discrete optimization problem which aims at finding a good solution rather than the optimal solution.
- It may or not have a **performance guarantee**.

Heuristics: classification

- Greedy algorithms
  - Used to build a solution from scratch
  - Decision process is decomposed into steps
  - At each step take a (the best) decision
- Local search algorithms
  - Start from a complete solution S and improve it by searching in its neighborhood.
  - Define the neighborhood of a solution $\mathcal{N}(S)$
  - Hill-climbing : at each step take the best solution S' in the neighborhood $\mathcal{N}(S)$. if S' improves S then set $S = S'$ and reiterate. Otherwise STOP.
- Metaheuristics
  - General-purpose algorithmic schemes to escape from local optima
  - Single solution vs population-based metaheuristics
- Deterministic vs randomized algorithms

Performance guarantee

- Discrete optimization problem P
- Heuristic H for P
- Instance I of P
- $f_H(I)$ Objective obtained on I by H
- $OPT(I)$ Optimum of I
- Performance ratio of H on I

$$\rho_H(I) = \frac{f_H(I)}{OPT(I)}$$

  Note $\rho_H(I) \leq 1$ for maximization problems while $\rho_H(I) \geq 1$ for minimization problems.

- Performance guarantee ρ_H of H on P

$$\forall I \in P, \rho_H(I) \leq \rho_H \text{ for min. problem}$$

$$\forall I \in P, \rho_H(I) \geq \rho_H \text{ for max. problem}$$

Performance guarantee: how can we obtain it in practice

- NP-hard to compute OPT(I)
- Suppose we have a polynomial time algorithm to compute a lower bound $LB(I)$
- If we show that

$$\forall I \in P, f_H(I) \leq \rho'_H LB(I) \text{ with} \rho'_H \geq 1 \text{ for min. problem}$$

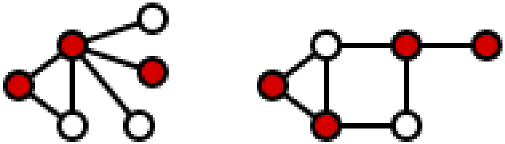
ρ'_H is a performance guarantee since $LB(I) \leq OPT(I)$

Vertex cover example

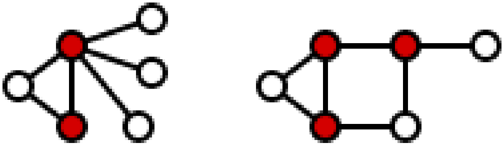
Cardinality vertex cover problem

Consider a graph $G = (V, E)$. A vertex cover is a set of vertices $V' \subseteq V$ such that every edge has an endpoint in V'. Find a vertex cover of minimum cardinality.

Vertex covers



Minimum vertex covers



A Lower bound for minimum vertex cover

- Matching of a graph $G = (V, E)$: a subset of the edges such that no two edges share an endpoint
- Maximal matching: a matching that is maximal in the inclusion sense
- Algorithm for finding a Maximal matching: select an edge and removes its endpoints from V. Reiterate until there are no more edges. Output the set of selected edges.

Theorem

The cardinality of any maximal matching is a lower bound for the minimum vertex cover

Proof.

Any vertex cover has to pick at least one endpoint of a matched edge and no two matched edges have a common endpoint. $\square$

An approximation algorithm for minimum vertex cover

- Algorithm 1 (vertex cover). Find a maximal matching and output the set of matched vertices.

Theorem

Algorithm 1 is a factor 2 approximation algorithm for the cardinality vertex cover problem

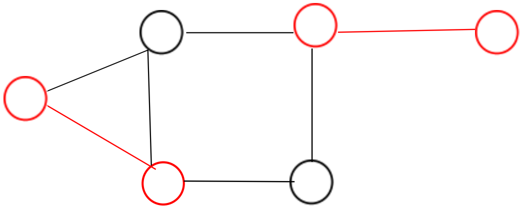
Proof.

- The set of matched vertices is a vertex cover since if an edge is not covered, it could have been added to the matching, which contradicts its maximality.
- We know the cardinality of the maximal matching is a lower bound, so it is lower than OPT, the size of the minimum vertex cover. Trivially, the size M of the matched vertices is twice the size of the matched edges, so $M \geq 2OPT$.

□

An approximation algorithm for minimum vertex cover

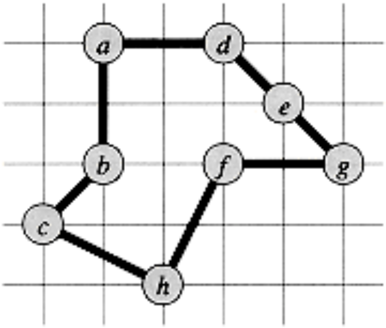
A cardinality 2 maximal matching yielding a cardinality 4 vertex cover.
Remark factor can also be reached on some instances !



TSP with triangle inequality example

Traveling salesman problem with triangle inequality

A traveling salesman must cover n cities starting from city 1 and returning to the starting city. c_{ij} is the travel cost from city i to city j. For any cities i, j, k we have $c_{ik} \leq c_{ij} + c_{jk}$. What is the minimum cost tour ?



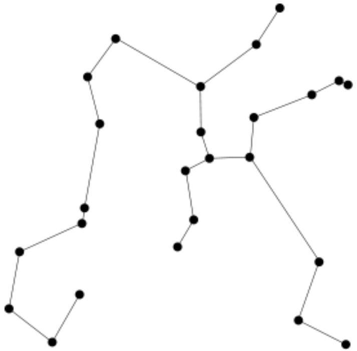
Example Taken from Introduction to Algorithms by Cormen, Leiserson, andRivest

Minimum spanning tree

Minimum spanning tree problem

consider a connected graph graph $G = (V, E)$. Each edge $(i, j) \in E$ has a cost c_{ij}. Find a spanning tree (e.g. a tree which covers all verties) of minimal cost

Polynomial problem solved by Prim's or Kruskal's algorithms.



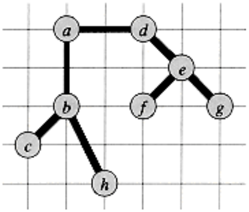
A lower bound for the TSP

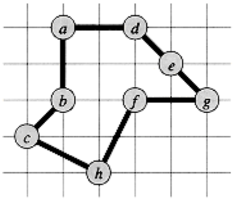
Theorem

The solution of the minimum spanning tree starting from any node is a lower bound for the TSP

Proof.

Consider the optimal tour and remove one edge. We obtain a chain which is a spanning tree for any root node. Its length is lower than the optimal tour as one edge was removed and larger than the minimum spanning tree from any root node. □





An approximation algorithm for the TSP with triangle inequality

Algorithm 2.

1. Compute the minimum spanning tree from the depot.
2. Duplicate each edge of the tree and consider the Eulerian cycle
3. Build an Hamiltonian cycle. Traverse the Eulerian cycle but each time an already visited vertice is about to be visited, jump to the next unvisited vertex.

Theorem

Algorithm 2 is a factor 2 approximation algorithm for the traveling salesman problem

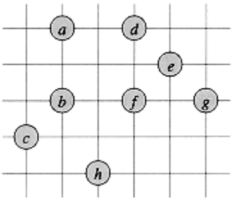
An approximation algorithm for the TSP with triangle inequality

Proof.

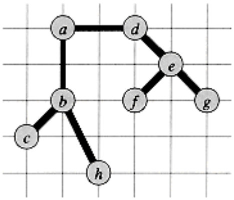
- The Eulerian cycle built at step 2 has a length equal to twice the length of the minimum spanning tree
- At step 3, each time we jump from i to an unvisited node j along the eulerian cycle, we replace cost $c_{ii_1} + c_{ii_2} + \ldots + c_{i_pj}$ by cost c_{ij} where $i_1, \ldots, i_p$ are already visited vertices. Since triangle inequality holds, the cost cannot be increased this way.
- The cost of the obtained Hamiltonian tour has a length lower than twice the length of the minimum spanning tree which is also lower than the optimal tour length.

$\square$

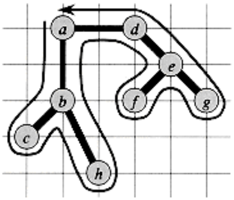
An approximation algorithm for the TSP with triangle inequality



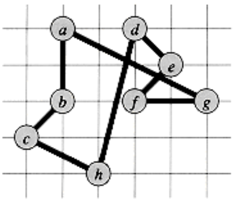
(a)



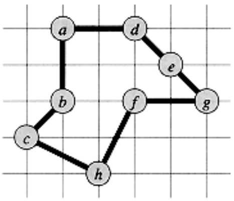
(b)



(c)



(d)



(e)

Example Taken from Introduction to Algorithms by Cormen, Leiserson, andRivest

Knapsack problem example

Knapsack problem (KP)

A company has a total budget B for funding a subset of n projets. Each project i has a cost c_i and a profit v_i. Which projects should the company select to maximize its total income ?

- Greedy Algorithm: Sort the items according to decreasing ratio $\frac{v_i}{c_i}$ and pick greedily the items in that order.
- Performance arbitrarily bad. Consider a $n = 2$ example with an item with profit 2 and cost 1 and an item with profit B and size B. Greedy algorithm takes only the first item. Performance ratio $\frac{B}{2}$!

A better greedy algorithm for the Knapsack problem

- Improved Greedy Algorithm: Sort items in decreasing $\frac{v_i}{c_i}$ and take items in this order until reaching item s which exceeds capacity $(B - \sum_{i=1}^{s-1} c_i < c_s)$. Solution s_1 selects items $1, ..., s-1$. Consider also the solution s_2 picking only the largest item s. Take the best of s_1 and s_2

Theorem

Improved Greedy Algorithm is a factor 2 approximation algorithm for the knapsack problem

Proof on following slides.

An upper bound for the Knapsack problem

LP relaxation

$$\begin{aligned} LKP = \max \quad & \sum_{i=1}^{n} v_i x_i \\ s.c. \quad & \sum_{i=1}^{n} c_i x_i \leq B \\ & 0 \leq x_i \leq 1, \quad i = 1, \ldots, n \end{aligned}$$

- LKP is an upper bound for the knapsack problem

Lemma

LKP is equal to the solution obtain by the following greedy algorithm. Sort items in decreasing $\frac{v_i}{c_i}$ and take items ($x_i = 1$) in this order until item s which exceeds capacity ($B - \sum_{i=1}^{s-1} c_i < c_s$). Then take a fractional part of item s by setting $x_s = \frac{c_s}{B - \sum_{i=1}^{s-1} c_i}$.

An upper bound for the Knapsack problem (contd)

Proof.

- Assume all items have different ratios $\frac{v_i}{c_i}$ (we can merge items having the same ratio)
- Suppose optimal solution x^* of the relaxed KP is different from $x_1 = \ldots = x_{s-1} = 1$, $x_s = \frac{c_s}{B - \sum_{i=1}^{s-1} c_i}$ and $x_{s+1} = \ldots = x_n = 0$.
- We must have $x_j^* > 0$ for at least one $j > s$ and $x_i^* < 1$ for at least one $i < s$.
- set $c = \min\left(c_j x_j^*, c_i(1 - x_i^*)\right)$.
- Consider solution x' identical to c^* except that $x_j' = x_j^* - \frac{d}{w_j}$ and $x_i' = x_i^* + \frac{d}{w_i}$.
- x' is feasible since its weight is equal to $\sum_{k=1}^n c_k x_k^* + \frac{w_i d}{w_i} - \frac{w_j d}{w_j} = B$ and its profit is equal to $\sum_{k=1}^n v_k x_k^* + d(\frac{v_i}{c_i} - \frac{v_j}{c_j}) > \sum_{k=1}^n v_k x_k^*$, which contradicts optimality of x^*.

□

An approximation algorithm for the Knapsack problem (contd)

- Assume items are numbered according to decreasing v_i/c_i
- Let OPT denote the optimal solution of KP.
- $LKP = \sum_{i=1}^{n} v_i + \frac{c_s}{B-\sum_{i=1}^{s-1} c_i} v_s \geq OPT \geq \sum_{i=1}^{n} v_i$
- Hence we have also $\sum_{i=1}^{n} v_i + v_s \geq OPT$
- For this, we must have $\max(\sum_{i=1}^{n} v_i, v_s) \geq \frac{1}{2} OPT$
- This is the value of the improved greedy algorithm. Hence the improved greedy algorithm is a factor 2 approximation algorithm.

Polynomial Time Approximation Scheme (PTAS)

- Algorithms seen so far have a constant approximation factor.
- Can we design algorithms that can get arbitrarily close to the optimum ?
- Yes ! These algorithms are called PTAS.

PTAS for minimization problem P

- A family of approximation algorithms $A_\epsilon : \epsilon > 0$ for P.
- A_ϵ is a factor $(1+\epsilon)$ - approximation algorithm for P.
- A_ϵ runs in time polynomial in input size for a fixed ϵ

A PTAS for the knapsack problem

Consider the following algorithm (AKP)

- Consider an integer constant $k \geq 1$
- Enumerate all subsets of k or less objects among n ($O(kn^k)$)
- for each subset, fill up the knapsack by the remaining items in descending order of the ratio $\frac{v_i}{c_i}$.

Theorem

The above algorithm is a factor $(1 + 1/k)$ approximation algorithm for KP.

$$\frac{OPT}{v(AKP)} \leq (1 + 1/k)$$

Proof in next slide.

A PTAS for the knapsack problem (Proof)

Proof that $\frac{OPT}{v(AKP)} \leq (1 + 1/k)$

- Let O be the optimal set of selected items. We suppose $|O| > k$.
- Let $H \subset O = \{a_1, \ldots, a_k\}$ be the subset of k most profitable items in terms of ratio $\frac{v_i}{c_i}$ (one of the enumerated subsets).
- Let G be the items selected by the greedy part.
- Let $L = O \setminus H = \{a_{k+1}, \ldots, a_h\}$ the "optimal" subset that was not selected by the enumerative part.
- Let m the first index in L that was not selected by the greedy part (because its size exceeded the remaining capacity).
- Knapsack contains H, $a_{k+1}, \ldots, a_{m-1}$ and other items not in O.
- Let B_e denote the remaining empty capacity. Size of $G \setminus O$ is $\Delta = B - B_e - \sum_{i=1}^{m-1} c_{a_i}$.
- Since the items of $G \setminus O$ have a ratio not smaller than v_{a_m}/c_{a_m}, profit of G is bounded by $v(G) \geq \sum_{i=k+1}^{m-1} v_{a_i} + \Delta v_{a_m}/c_{a_m}$.

A PTAS for the knapsack problem (Proof contd)

Proof that $\frac{OPT}{v(A-KP)} \leq (1+1/k)$

- Optimal profit is

$$OPT = v(O) = \sum_{i=1}^{k} v_{a_i} + sum_{i=k+1}^{m-1} v_{a_i} + sum_{i=m}^{|O|} v_{a_i}$$

$$\leq v(H) + (v(G) - \Delta v_{a_m}/c_{a_m}) + (B - \sum_{i=1}^{m-1} c_{a_i}) v_{a_m}/c_{a_m}$$

$$= v(H) + v(G) + B_e v_{a_m}/c_{a_m} < v(H \cup G) + v_{a_m}$$

- We also know that $v(AKP) \geq v(H \cup G)$ as H is part of the enumerated subsets and G is the set of items selected by the greedy part if H is selected. This yields $OPT - v(AKP) < v_{a_m}$.
- Furthermore we have $v(a_m) \leq v(O)/(k+1)$ (since all items in G have a ratio of at least v_{a_m}/c_{a_m}). This yields the approximation ratio.

Limits of PTAS

- With algorithm AKP, we have a $1-\epsilon$ approximation where $1/\epsilon = k+1$.
- The running time is $O(1/\epsilon^{1/\epsilon})$ which is polynomial in n but exponential in $1/\epsilon$ and drammatically increases as k increases !

Could we find a family of algorithms A_ϵ which are both polynomial in input data and ϵ ?

Fully polynomial time approximation algorithm (FPTAS) for minimization problem P

- A family of approximation algorithms $A_\epsilon : \epsilon > 0$ for a problem P.
- A_ϵ is a factor $(1+\epsilon)$ - approximation algorithm for P.
- A_ϵ runs in time polynomial in input size and $1/\epsilon$

Pseudo polynomial algorithm for the knapsack problem

- There exists a pseudo-polynomial (dynamic programming or DP) algorithm (running in $O(n^2V)$) to solve the knapsack problem, where V is the maximum profit $V = \max_{i=1,\dots,n} v_i$.
- Let $S_{i,v}$ denote a subset of $\{a_1, \dots, a_p\}$ **that has a profit of exactly** v, taking the least amount of capacity possible.
- Let $C_{i,p}$ the size of $S_{i,v}$ with $C_{i,v} = \infty$ if there is no such subset
- We have the simple case $C(1, v) = c_{a_1}$ for $v = v_{a_1}$ and $A(1, v) = \infty$ otherwise.
- The following recurrence allows to compute all $C(i, v)$:

$$C(i+1, v) = \begin{cases} \min(C(i, v), c_{a_{i+1}} + C(i, v - v_{a_{i+1}}))) \text{ if } v_{a_{i+1}} \leq v \\ C(i, v) \text{ otherwise} \end{cases}$$

The optimal subset corresponds to the set $S_{n,v}$ that maximizes v and such that $C_{n,v} \leq B$.

FPTAS for the knapsack problem

Idea : scaling the profit down so as to obtain a polynomial algorithm.
Consider the following algorithm (AKP2)

- Given $\epsilon > 0$, set $K = \frac{\epsilon V}{n}$
- For each object a_i define $v'_{a_i} = \lfloor \frac{v_{a_i}}{K} \rfloor$
- use DP to find optimal subset S' for profits v' and objects $a_1, \ldots, a_n$

Theorem

AKP2 is a fullly polynomial approximation scheme for KP as S' verifies $v(S') \geq (1 - \epsilon)OPT$

Proof in next slide

FPTAS for the knapsack problem (Proof)

Proof.

- Algorithm has an $O(n^2\lfloor\frac{V}{K}\rfloor) = O(n^2\lfloor\frac{n}{\epsilon}\rfloor)$ time complexity which is polynomial in n and $1/\epsilon$
- Let O denote the optimal set. Each profit verifies $v_a \geq Kv'_a$ so in the scaled problem, profit $v'(O)$ verifies $v(O) - Kv'(O) \leq nK$
- Solving the scaled instance we obain a set S' such that $v'(S') \geq v'('O)$ and so $v(S') \geq Kv'(O) \geq v(O) - nK = OPT - \epsilon v$. It yields $v(S') \geq (1-\epsilon)OPT$

$\square$

Inapproximability

Proof.

- Complexity issues : strongly NP-hard problem cannot have a FPTAS
- Other problems cannot have a PTAS
- Some other problems cannot even have a constant factor appproximation unless $P = NP$
- Efficiency of approximation algorithms in practive ?

$\square$

Sources

- Handbook of Scheduling, Algorithms Models and Performance Analysis, J.Y.T. Leung, Chapmann&Hall/CRC, 2004
- Approximation Algorithms, V.V. Vazirani, Springer, 2003
- The Knapsack Problem and Fully Polynomial Time Approximation Schemes (FPTAS), Katherine Lai, 18.434: Seminar in Theoretical Computer Science, Prof. M. X. Goemans, `math.mit.edu/~goemans/18434S06/knapsack-katherine.pdf`
- Anupam Gupta and R. Ravi, 15-854: Approximation Algorithms, `http://www.cs.cmu.edu/afs/cs/academic/class/15854-f05/www/`