HØGSKOLEN I BERGEN

BERGEN UNIVERSITY COLLEGE

Greedy algorithms and local search

Chapter 2

- Greedy algorithms
  Step by step: Next step make some decision that is locally best possible
  Typically primal infeasible
- Local search algorithms
  Starts with an arbitrary feasible solution
  Check if some small, local change results in improved objective function
- Both
  - popular and easy to implement
  - Good running time in practice

Scheduling jobs

HØGSKOLEN I BERGEN
BERGEN UNIVERSITY COLLEGE

- Scheduling jobs with deadlines on a single machine
  - n jobs, each job j
    - needs p_j time units of processing time
    - release date r_j
    - due date d_j
    - finish at time C_j
    - lateness $L_j = C_j - d_j$
  - one machine
    - can process at most one job at the time
    - must process a job to its completion
  - Minimize maximum lateness

$$L_{max} = \max_{j=1,\ldots,n} L_j$$

Negative result

- Difficult to obtain near-optimal values
  - If there were a p-approximation algorithm, then for any input with optimal value 0, the algorithm must still find a schedule of objective function value at most $p \cdot 0 = 0$
  - Would imply P = NP
  - Consider the problem were all due dates are negative
    - Implies optimal value is always positive

Algorithm

- Earlies due date rule (EDD-rule)
  - Process next an available job with the earlies due date

- Theorem 2.2. The EDD-rule is a 2-approximation algorithm for the problem of minimizing the maximum lateness on a single machine subject to release dates with negative due dates.

Clustering

- Examples
  - Finding similarities and dissimilarities in large amount of data
    - Customers with similar purchasing behavior
    - Voting behavior
    - Search engines group webpages by similarity of topic

The k-center problem

- Input
  - Undirected, complete graph G = (V, E)
  - Distance $d_{ij} \geq 0$ between every pair of vertices $i, j \in V$. Similar on previous slide means smaller d_{ij}
  - Integer k
- Assume
  - $d_{ii} = 0$ and $d_{ij} = d_{ji}$
  - Obey triangle inequality
- Goal: Choose k vertices (cluster centers) such that the maximum distance of a vertex to its cluster center is as small as possible

Notation

- Distance of a vertex i from a set $S \subseteq V$

$$d(i, S) = \min_{j \in S} d_{ij}$$

Greedy algorithm

Pick arbitrary $i \in V$

$S \leftarrow \{i\}$

while $|S| < k$ do

$$j \leftarrow \arg\max_{j \in V} d(j, S)$$

$$S \leftarrow S \cup \{j\}$$

- Theorem 2.3. The greedy algorithm is a 2-approximation algorithm for the k-center problem.

Negative result

- Theorem 2.4. There is no α-approximation algorithm for the k-center problem for $\alpha < 2$ unless P = NP.
- Consider the Dominating set problem which is NP-complete
  - Graph G=(V,E) and integer k
  - Decide if there exists a set $S \subseteq V$ of size k such that each vertex is either in S, or adjacent to a vertex in S

Proof

- Given an instance of dominating set problem
- Define instance of k-center problem by setting the distance between adjacent vertices to 1 and nonadjacent vertices to 2
- Any p-approximation algorithm with $p < 2$ must always produce a solution of radius 1 if such a solution exists

Scheduling jobs

- n jobs to be processed
- m identical machines (running in parallel) to which each job may be assigned
- Each job $1, \ldots, n$ must be processed on one of these machines for p_j time units without interruption
- Each job is available at time 0
- Each machine can process at most one job at the time
- The aim is to complete all jobs as soon as possible

Local search algorithm

- Start with any schedule
- loop
  - Consider the job ℓ that finishes last
  - if (job ℓ can be reassigned to another machine so it
    finishes earlier) {
    reassign it

    else

    exit
- continue

Performance analysis

- Bounds of overall finish time
  - At least as long as the longest job

    $C^*_{max} \geq \max_{j=1,\ldots n} p_j$
  - At least one machine must have work at least corresponding to the average work per machine

$$C^*_{max} \geq \frac{1}{m}\sum_{j=1}^{n} p_j$$

Theorem

- Theorem 2.5. The local search procedure for scheduling jobs on identical parallel machines is a 2-approximation algorithm

Sketch of proof

- Let ℓ be the job that completes last in the schedule and let its completion time be C_ℓ
- Every other machine must be busy from time 0 until the start of job ℓ at time $S_\ell = C_\ell - p_\ell$
- Partition the schedule into two different time intervals 0 to S_ℓ and S_ℓ to C_ℓ.
- Both are less than C^*_{max} because of bounds on previous slide
- Have to show it is poly time (argue that we never transfer a job twice

Greedy algorithm

- List scheduling algorithm
  - Assign the jobs as soon there is a machine available

HØGSKOLEN I BERGEN
BERGEN UNIVERSITY COLLEGE

- Theorem 2.6. The list scheduling algorithm for the problem of minimizing the makespan on m identical parallel machines is a 2-approximation algorithm.

- Longest processing time rule
  - Order the list with the longest jobs first

- Theorem 2.7. The longest processing time rule is a 4/3-approximation algorithm for scheduling jobs to minimize the makespan on identical machines.

- Sketch of proof

  Assume we have a counterexample
  Show that such an example can not exist

Travelling Salesman problem (TSP)

- Given a set of cities $\{1,2,\ldots,n\}$
- Symmetric $n \times n$ cost matrix $C = (c_{ij})$ travelling from city i to city j.
- $c_{ii} = 0$ and $c_{ij} \geq 0$
- A feasible solution, or tour, traversal of the cities in the order $k(1), k(2), \ldots, k(n)$ where $k(i)$ is the i’th city to visit.
- Cost of tour: $c_{k(n)k(1)} + \sum_{i=1}^{n-1} c_{k(i)k(i+1)}$
- Want to find the tour with minimum cost

- NP-complete to decide whether a graph has a Hamiltonian cycle
- An approximation algorithm for TSP can be used to solve the Hamiltoinan cycle in the following way. Set
  - $c_{ij} = 1$ if $(i, j) \in E$
  - $c_{ij} = \alpha n + 2$ otherwise
- Now an α-approximation algorithm could solve the Hamiltonian cycle problem

- Theorem 2.9. For any $\alpha > 1$, there does not exist an α-approximation algorithm for the travelling salesman problem on n cities, provided P $\neq$ NP. In fact, the existence of an $O(2^n)$-approximation algorithm for TSP would imply that P = NP.

Metric TSP

- Satisfy the triangle inequality

$$c_{ij} \leq c_{ik} + c_{kj}$$

- Will look at three approximation algorithms for metric TSP
  - Nearest addition algorithm
  - Double tree algorithm
  - Chirstofides' algorithm

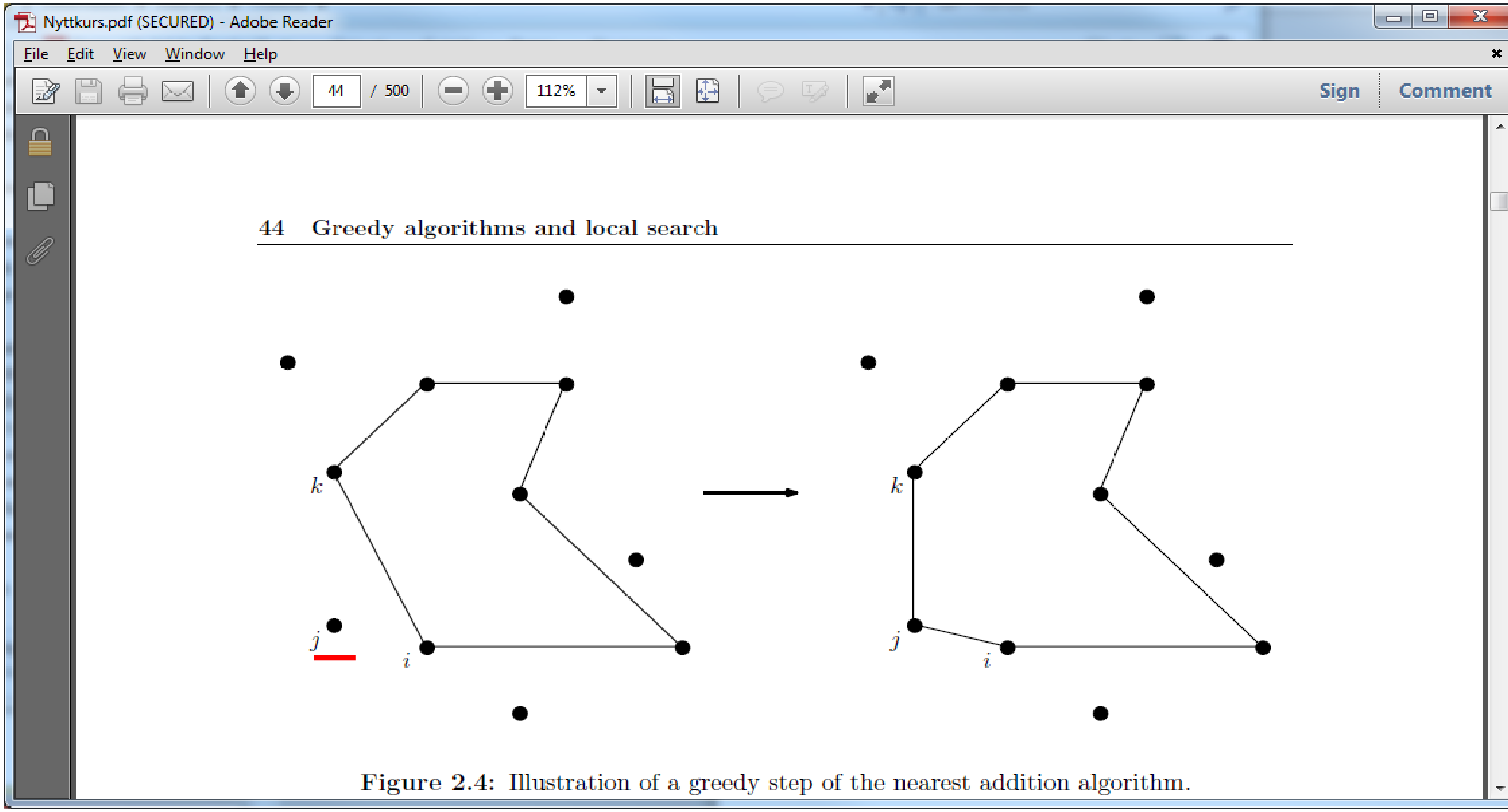
Useful fact

- Lemma 2.10. For any input to the travelling salesman problem, the cost of the optimal tour is at least the cost of the minimum spanning tree on the same input
- Proof by contradiction. Assume it is not. Consider the tour. Remove the edge with larger weight. Then we have a spanning tree with cost less than a minimum spanning which is impossible.

Nearest addition algorithm

- Build at tour between the two closest cities, i and j.
- Tour $\mathrm{T} = (i \rightarrow j \rightarrow i)$
- $S = \{i, j\}$
- while $(|S| < n)$
  - find a pair of cities $i \in S$ and $j \notin S$ for which cost c_{ij} is minimum
  - Let k be the city that follows i in current tour. Must consider the tour both clockwise and counterclocwise
  - Insert j in tour such that $T = (\cdots \rightarrow i \rightarrow j \rightarrow k \rightarrow \cdots)$

Figure


Nyttkurs.pdf (SECURED) - Adobe Reader
File Edit View Window Help
44 / 500
112%
Sign
Comment
44 Greedy algorithms and local search
k
j
i
k
j
i
Figure 2.4: Illustration of a greedy step of the nearest addition algorithm.

- Theorem 2.11.The nearest addition algorithm for the metric TSP is a 2-approximation algorithm
- We notice that nodes are added in the same sequence as in Prims algorithm for minimum spanning trees.

Proof of Theorem 2.11

- Idea. After each iteration the cost is at most twice the cost of the corresponding MST found by Prim
  - First iteration obvious
  - Later
    - Prim adds c_{ij}
    - We add $c_{ij} + c_{jk}$-c_{ik}
    - By triangle inequality $c_{jk} \leq c_{ji} + c_{ik}$ or $c_{jk} - c_{ik} \leq c_{ji}$
    - Hence increase is at most $c_{ij} + c_{ji} = 2c_{ij}$
    - Thus the cost of tour found is at most 2*cost(Prim) <=2*OPT and Theorem is proved

The double-tree algorithm

- (Saw it the first day)
- Find MST
- Double all the edges in the tree (then all nodes will have even degree)
- Find Eulerian tour (exist when all nodes have even degree. Easy to find)
- Consider tour. Skip nodes that are already visited. Because of triangle inequality cost can not increase

- Theorem 2.12. The double tree algorithm for the metric TSP is a 2-approximation algorithm
- Proof: Same ideas as nearest addition algorithm

Perfect matching

- If we have $2k$ nodes, a perfect matching is k edges where all nodes appears exactly once
- It is possible to compute the perfect matching of minimum total cost in polynomial time.

Better algorithm

- If we sum all node degrees in a graph, the sum must be even (every edge contributes 2 in total degree)
- In double-tree algorithm, we doubled every edge in MSP so we could find an Eulerian tour
- However, the number of nodes of odd degree must be even
- Sufficient to find a minimum perfect matching among the nodes with odd degree and add these edges to the tree

Christofides’ algorithm

- Find a MST
- Find a minimum perfect matching among the nodes with odd degree in MST. Add these edges to the MST
- Find a Eulerian tour in the resulting graph
- Consider the tour. Skip already visited vertices

Christofides' algorithm

- Theorem 2.13. Christofides' algorithm for the metric TSP is a 3/2-approximation algorithm

Proof of Christofides' algorithm

- Let O be set of odd degree vertices in MST
- Consider optimal tour of all vertices.
- Make a tour on just vertices in O where the vertices appears in the same order as in the optimal tour.
- This tour can not be longer than before because of the triangle inequality
- Color the edges red and blue, alternating colors as the tour is traversed
- The red and blue edges represent two perfect matchings. The cheapest must have cost at most OPT/2
- The minimum perfect matching can not be more expensive
- Hence Christofides' algorithm find a tour of cost at most cost(MST) + cost(Perfect matching) <= OPT + OPT/2 = 3/2OPT

TSP, Negative results

- No better algorithm for the metric TSP is known
- Theorem 2.14. Unless P = NP, for any constant $\alpha < \frac{220}{219} \approx 1.0045$, no α-approximation algorithm for the metric TSP exists.
- Improved to 185/184 ≈ 1.0054

Maximizing Float in Bank Accounts

- Wish to open k bank accounts
- Let
  - B be the set of banks
  - P be the set of people we regularly pays
  - $v_{ij} \geq 0$ value created to be able to pay person $j \in P$ from bank account $i \in B$ (takes into account time, interest rate and other things)
- Wish to find
  - $S \subseteq B, |S| \leq k$ that maximizes $v(S) = \sum_{j \in P} \max_{i \in S} v_{ij}$

Greedy algorithm

- $S \leftarrow \emptyset$
- while $|S| < k$ do
  - $i \leftarrow \arg\max_{i \in B} v(S \cup \{i\}) - v\,(S)$
  - $S \leftarrow S \cup \{i\}$
- return S

Performance

- Theorem 2.16. The greedy algorithm gives a $(1 - \frac{1}{e})$-approximation algorithm for the float maximization problem. ($e \approx 2.72$)

- Notice. Since this is a maximization problem, we have performance guarantee less than 1.

Finding minimum degree spanning trees

- Given a graph $G = (V, E)$
- Want to find a spanning tree T of G so as to minimize the maximum degree of nodes in T.

Finding minimum degree spanning trees

- Theorem 2.18. It is NP-complete to decide whether or not a given graph has a minimum-degree spanning tree of maximum degree two.
- Proof: We know that Hamiltonian path is NP-complete. A Hamiltonian path is a MSP where the maximum degree is two

Algorithm

- Start with an arbitrary spanning tree T
- Pick a vertex u.
- Look at all edges (v, w) that are not in T but if added to T creates a cycle containing u.
- Suppose $\max(d_T(v), d_T(w)) \leq d_T(u) - 2$
- Add (v, w) and remove edge (u, y) where (u, y) is part of the cycle (figure next slide)

Nyttkurs.pdf (SECURED) - Adobe Reader

File Edit View Window Help

50 / 500 112% Sign Comment

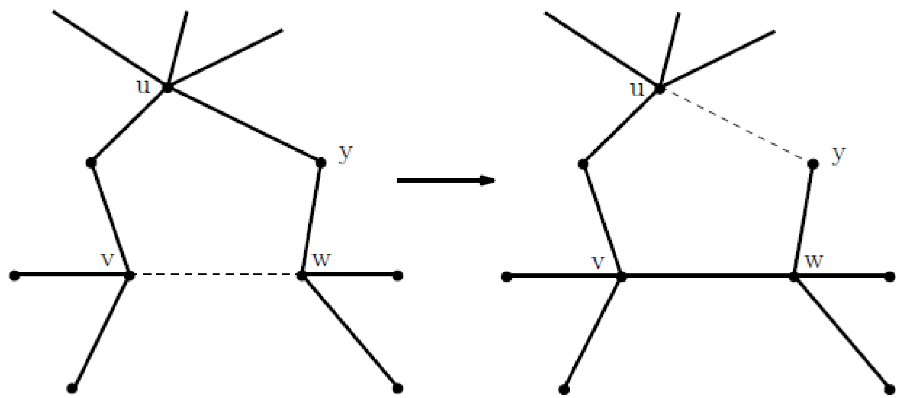


Figure 2.5: Illustration of a local move for minimizing the maximum degree of a spanning tree. The bold solid lines are in the tree, and the dashed lines are graph edges not in the tree.

- Remember. For an algorithm to be an approximation algorithm
  - Must have a performance guarantee
  - Run in polynomial time
- Problem: to show that algorithm takes poly time
- Solution: apply local moves only to nodes whose degree is relatively high.
  - Let $\Delta(T)$ be the maximum degree of a node in T
  - Algorithm picks node that has degree at least $\Delta(T) - \ell$

- Theorem 2.19. Let T be a locally optimal tree. Then $\Delta(T) \leq 2OPT + \ell$, where $\ell = \lceil \log_2 n \rceil$

Polynomial time

- Idea
  - Defines a potential function
  - Defines a minimum potential
  - The potential drops by a certain percentage in each iteration
  - Can limit the number of iterations

Edge Coloring

- An undirected graph is k-edge-colorable if each edge can be assigned exactly one of k colors in such a way that no two edges with same colors share an endpoint
- Want to obtain a k-edge-coloring with k as small as possible

- Let Δ be the maximum degree of a vertex in the given graph
- Theorem 2.22. For graphs with $\Delta = 3$, it is NP-complete to decide whether the graph is 3-edge-colorable or not.

Algorithm that uses Δ+1 colors

- Algorithm that has elements of both greedy algorithms and local search
  - Find an uncolored edge (u,v)
    - If possible, color it with one of the Δ+1 colors (greedy)
    - Else, locally change some edge colors such that it is possible to color (u, v) (local search)

- Will not look into the details of the algorithm (page 49)

Theorem

- Theorem 2.23. There is a polynomial-time algorithm to find a $(\Delta + 1)$-edge-coloring of a graph.