Review session: Exam question 6
Approximation Algorithms and Local Search Heuristics

- Lecture 24
  - Approximation algorithms for VERTEX COVER, TSP, MAXE3SAT, WEIGHTED VERTEX COVER.
  - CLRS, pages 1106-1117 and 1123-1127
- Lecture 25 (one hour)
  - Approximation algorithms for SET COVER and KNAPSACK.
  - Supplementary notes for Set Cover and Knapsack
- Lecture 26
  - Local search heuristics for TSP.
  - Johnson and McGeoch, Sections 1-3.
- Lecture 27 (one hour)
  - Local search heuristics for TSP.
  - Johnson and McGeoch, Sections 4-8.

Approximation Algorithm

An approximation algorithm is for an optimization problem Γ is

- an efficient (polynomial time) algorithm A that
- given instance x outputs a feasible solution y such that
- an approximation ratio

$$\max\left(\frac{\mathrm{val}_x\,(y)}{OPT(x)}, \frac{OPT(x)}{\mathrm{val}_x(y)}\right) \leq \rho(|x|)$$

holds.

We say that A is a polynomial time ρ-approximation algorithm.

MIN VERTEX COVER

- Given an undirected graph $G = (V, E)$, find the smallest subset $C \subseteq V$ that covers E.

```
APPROX-VERTEX-COVER(G)
1  C = ∅
2  E' = G.E
3  while E' ≠ ∅
4      let (u, v) be an arbitrary edge of E'
5      C = C ∪ {u, v}
6      remove from E' every edge incident on either u or v
7  return C
```

2-approximation algorithm: Let M be matching(!) chosen in line 4. Then:

$|M| \leq |C^*|$ and $|C| = 2|M|$ which implies $|C| \leq |C^*|$.

Metric TSP

- Given $n \times n$ non-negative distance matrix (c_{ij}), satisfying Δ-inequality, find a permutation π on $\{0,1,2,..,n-1\}$ minimizing

$$\sum_{i=0}^{n-1} c_{\pi(i),\pi((i+1) \bmod n)}$$

APPROX-TSP-TOUR(G, c)

1 select a vertex $r \in G.V$ to be a "root" vertex
2 compute a minimum spanning tree T for G from root r
  using MST-PRIM(G, c, r)
3 let H be a list of vertices, ordered according to when they are first visited
  in a preorder tree walk of T
4 **return** the hamiltonian cycle H

2-approximation: Let W be Euler tour obtained by duplicating the edges of T. Then: $c(T) \leq c(H^*)$, $c(W) = 2c(T)$, and $c(H) \leq c(W)$ which implies $c(H) \leq 2c(H^*)$.

Christofides algorithm

- Obtain better Euler tour W as follows:
  - Compute MST T
  - Let V_{odd} be cities of odd degree in T. (Note: $|V_{\text{odd}}|$ must be even).
  - Compute min cost perfect matching M on V_{odd}.
  - Obtain Euler tour W from $T \cup M$.
  - Take shortcuts to obtain H.

- $^3/_2$-approximation:
  - H^* induces an even length tour $\widetilde{H}$ on V_{odd} with $c(\widetilde{H}) \leq c(H^*)$.
  - Alternating edges of $\widetilde{H}$ gives two matchings M_1 and M_2 on V_{odd} such that $c(M_1) + c(M_2) = c(\widetilde{H})$.
  - Thus $c(M) \leq \min\big(c(M_1), c(M_2)\big) \leq {}^1/_2\, c(\widetilde{H})$
  - Now $\mathrm{c(H)} \leq c(W) = c(T) + c(M) \leq {}^3/_2\, c(H^*)$

Approximating general TSP is NP-hard

Theorem

If there is an efficient approximation algorithm for TSP with *any* approximation factor ρ then P=NP.

Proof

We use a modification of the reduction of Hamiltonian cycle to TSP.

- Given instance $G = (V, E)$ of the Hamiltonian cycle problem and $\rho \geq 1$, construct TSP instance (V, d) as follows:

$$d(u, v) = 1 \qquad \text{if } (u, v) \in E$$
$$d(u, v) = \rho\,|V| + 1 \qquad \text{otherwise.}$$

- Suppose we have an efficient approximation algorithm for TSP with approximation ratio ρ. Run it on instance (V, d).
- (V, E) has a Hamiltonian cycle if and only if the returned solution has length at most $\rho|V|$.

Min weight vertex cover

- Given an undirected graph $G = (V, E)$ with non-negative weights $w(v)$, $v \in V$, find the subset $C \subseteq V$ that covers E.

- Formulate as ILP.
- Solve LP relaxation to optimality, giving optimal (fractional) solution x'.
- Round to integer solution x.

2-approximation algorithm.

ILP, LP, and rounding

- ILP:

Min $\sum_{v \in V} w(v) x_v$

s.t. $x_u + x_v \geq 1$, for all $(u, v) \in E$.

$x_v \in \{0,1\}$, for all $v \in V$

- Relaxation: Simply remove integrality constraint. Compute optimal x'.
- Rounding: Simple rounding - let $x_v = 1$ if and only if $x'_v \geq {}^1/_2$.
- Analysis:
  - Feasibility insured: If $x'_u + x'_v \geq 1$ then either $x_u = 1$ or $x_v = 1$ (or both!).
  - Approximation ratio: We at most double every variable, thereby at most doubling the cost.

MAXE3SAT

- Given Boolean formula in CNF form with
  find an assignment satisfying as many clauses as possible.

- Simple algorithm: Give completely random output.

- Analysis:
  - Every clause is satisfied with probability $\geq {}^{7}/_{8}$.
  - Hence, expected number of clauses satisfied must be at least a fraction ${}^{7}/_{8}$.
    (recall $E[X] = \Pr[X = 1]$ for random variable $X \in \{0,1\}$
    and we have linearity of expectation, $\mathrm{E}[\sum X_i] = \sum E[X_i]$)
  - The optimal solution can satisfy at most all clauses.
  - Hence, *expected* approximation ratio ${}^{8}/_{7}$.

Min set cover

- Given set system $S_1, S_2, \ldots, Sm \subseteq X$, find smallest possible subsystem covering X.

Input: Universe X, Family $\mathcal{F}$ of subsets of X.
Output: Set cover $\mathcal{C}$.

```
U ← X
C ← ∅
while U ≠ ∅ do
    pick S ∈ F that maximizes |S ∩ U|
    U ← U \ S
    C ← C ∪ {S}
end while
return C
```

- $\ln(n)$ approximation, more precisely, H_s-approximation, where $s = \max|S_i|$ and $H_s = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{s}$.

Min set cover, analysis of algorithm

- Having a greedy algorithm allows us to directly compare each set chosen to an optimal (but unknown) solution C^* of (unknown) size m.
- At every step of the algorithm, the optimal set cover is (trivially) a cover of all uncovered elements.
- Easy analysis:
  - Chosen set must cover at least a fraction $1/m$ of uncovered elements.
  - After i iterations, at most $n(1 - 1/m)^i \leq n \exp(-1/m)^i = \exp\left(-i \frac{\ln n}{m}\right)$ elements are non-covered.
- Refined analysis:
  - Assign costs c_x of covering elements x. When choosing a set to cover j non-covered elements, assign cost $1/j$ to these elements.
  - If a set T of the optimal solution has k members, then the cost assigned to the elements must be at most $1/1, 1/2, \ldots, 1/k$, since T is always a choice for the greedy algorithm.
  - The total cost of covering the elements of T is thus at most H_k.
  - The result then follows by subadditivity:

$$|C| = \sum_{x \in X} c_x \leq \sum_{S \in C^*} \sum_{x \in S} c_x \leq \sum_{S \in C^*} H_{|S|} \leq |C^*| \, H_{\max|S|}$$

Approximation Schemes, PTAS, FPTAS

- Some optimization problems can be approximated very well, with approximation ratio $1 + \varepsilon$ for any $\varepsilon > 0$.

- An approximation scheme takes an additional input, $\varepsilon > 0$, and outputs a solution within $1 + \varepsilon$ of optimal.

- PTAS: Running time polynomial for every *fixed* $\varepsilon > 0$.
- FPTAS: Running time polynomial in size of instance *and* $\frac{1}{\varepsilon}$.

FPTAS for Knapsack

- Given n items with weights $w_1, \ldots, w_n$, values $v_1, \ldots, v_n$ and weight limit W,
- Fit items within weight limit maximizing total value.

Input: Weight $w_1, \ldots, w_n$, values $v_1, \ldots, v_n$, weight limit W and parameter $\epsilon > 0$.
Output: Selection of items, $S' \subseteq \{1, \ldots, n\}$.

1: $V \leftarrow \max\{v_i \mid w_i \leq W\}$
2: $B \leftarrow \epsilon V/n$
3: $v'_i \leftarrow \lfloor v_i/B \rfloor$
4: Compute optimal solution S' using weights $w_1, \ldots, w_n$, new values $v'_1, \ldots, v'_n$ and weight limit W
5: **return** S'

Using DP algorithm for Knapsack running time is

$$O(n^2 \max v'_i) = O\left(n^2 V/B\right) = O\left(\frac{n^3}{\varepsilon}\right)$$

Analysis of FPTAS

Input: Weight $w_1, \ldots, w_n$, values $v_1, \ldots, v_n$, weight limit W and parameter $\epsilon > 0$.

Output: Selection of items, $S' \subseteq \{1, \ldots, n\}$.

1: $V \leftarrow \max\{v_i \mid w_i \leq W\}$
2: $B \leftarrow \epsilon V/n$
3: $v'_i \leftarrow \lfloor v_i/B \rfloor$
4: Compute optimal solution S' using weights $w_1, \ldots, w_n$, new values $v'_1, \ldots, v'_n$ and weight limit W
5: **return** S'

$$\sum_{i \in S'} v_i \geq B \sum_{i \in S'} v'_i \geq B \sum_{i \in S} v'_i \geq B \sum_{i \in S} \left(\frac{v_i}{B} - 1 \right) \geq \left(\sum_{i \in S} v_i \right) - nB$$

$$\geq \left(\sum_{i \in S} v_i \right) - \varepsilon V \geq (1 - \varepsilon) \sum_{i \in S} v_i$$

Local Search and Heuristics

LocalSearch(ProblemInstance x)

$y :=$ feasible solution to x;

while $\exists z \in N(y) : v(z) < v(y)$ **do**

  $y := z$;

od;

return y;

Heuristics: Okay to end up in *local but not global minima*, if they are not too bad….

Case story: TSP

The Initial Solution

- Christofides
- Greedy heuristic
- Nearest neighbor heuristic
- Clarke-Wright

Other possibilities considered were worse with respect to both quality and running time.

It is important to carefully engineer each algorithm before comparison.

It is OK to use heuristics here as well.

Test instances: Euclidean instances and distance matrix instances of several thousands cities

Quality compared relative to Held-Karp lower bound.

Natural neighborhood for TSP

- k-OPT for $k = 2,3,4, \ldots$
- Remove k edges from tour, and add k edges to form different tour.
- Tricks for speeding up neighborhood search:
  - Neighbor lists
  - Prune lists to 20 elements
  - Use don’t look bits
- Running 3-OPT becomes feasible even for millions of cities!

Boosting local search

- Theme: How to escape local optima
- General-purpose (heuristics for making heuristics)
  - Taboo search
  - Simulated annealing
  - Evolutionary algorithms

- Best approach for TSP: Lin-Kernighan (incorporates taboo-like features)

- Advice: Don’t fall into the “everything looks like a nail if you have a hammer” trap.

Taboo search

- When the local search reaches a local minimum, **keep searching.**
- After a certain “move” has been made, it is declared and may not be used for a while.
- “Move” should be defined so that it becomes taboo to go right back to the local optimum just seen.

Simulated Annealing

FeasibleSolution SA(ProblemInstance x)

$y :=$ feasible solution to x; T: =big;

repeat

  $T := 0.99\,T$;

  Pick a random member z of $N(y)$;

  with probability $\min\left(\exp\left(\frac{v(y)-v(z)}{T}\right), 1\right)$ let $y: = z$

until tired;

return the best y found;

Simulated annealing

- **Theorem:** If T is lowered sufficiently slowly (exponentially many moves must be made), **the final solution will with high probability be optimal**!

- In practice *T* must be lowered faster.

- Johnson and McGeoch: Simulated annealing with 2OPT neighborhood is promising but neighborhood must be pruned to make it efficient.

- Still, not competitive with LK or ILK on a time-equalized basis (for any amount of time).

Evolutionary Algorithm

FeasibleSolution EvolSearch(ProblemInstance x)

P := initial population of feasible solutions to *x*;

while !tired **do**

  Expand(*P)*;

  Selection(*P)*

od;

return best solution obtained at some point;

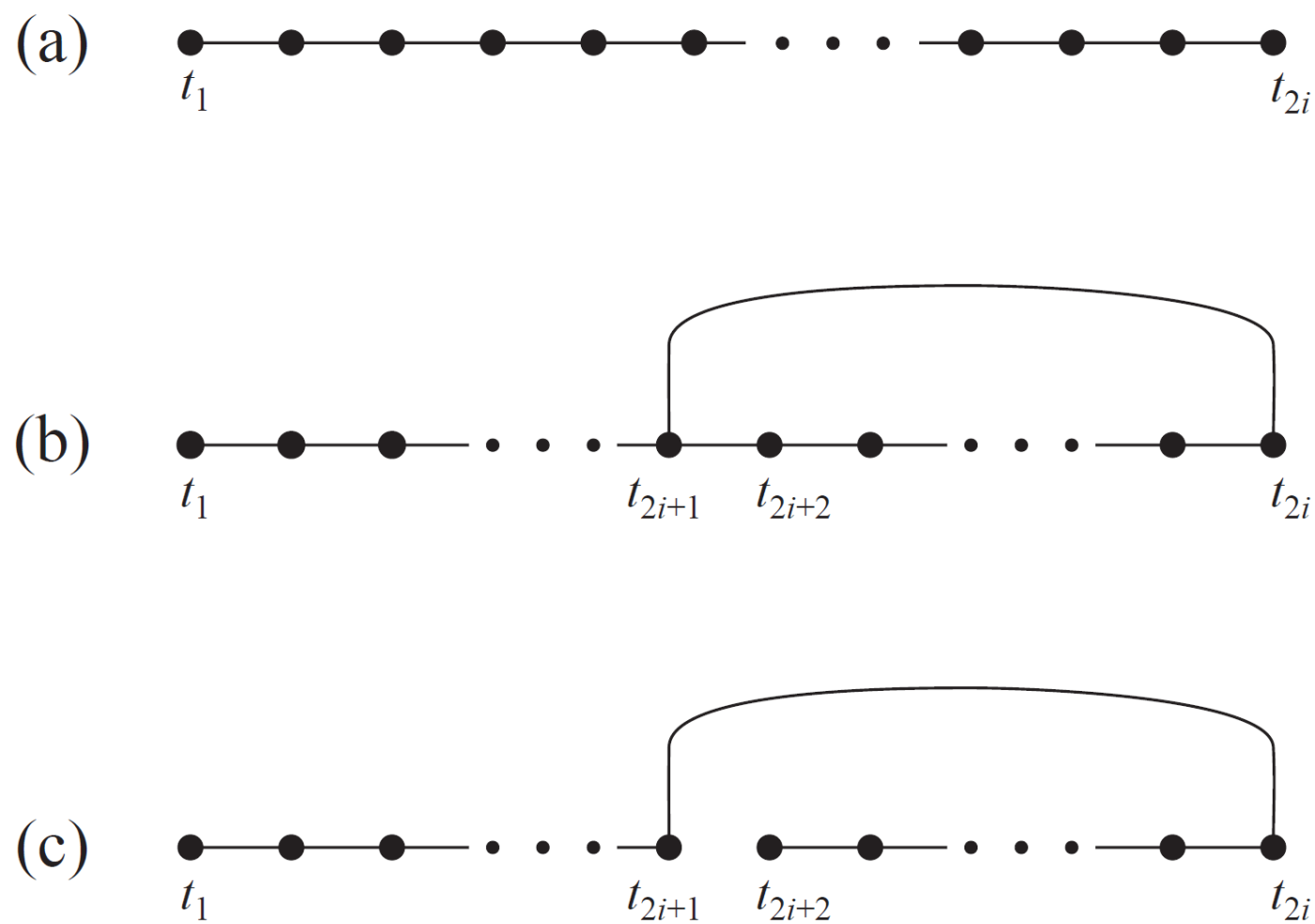
Combine(*x*,*y*) for TSP

- Combine(*x*,*y*): Take the graph consisting of edges of x and y. Find the optimal TSP solution using only edges from that graph.

- Finding the optimal TSP tour in a graph which is the union of two Hamiltonian paths can be done efficiently in practice.

- More “obvious” versions of combine (like the generic combine) yield evolutionary algorithms which are not competitive with simpler methods.

Lin-Kernighan move



Lin-Kernighan Search

- 3opt search with “intensification”.

- Whenever a 3opt move is being made, we view it as two LK-moves and see if we **in addition** can perform a number of LK-moves (an LK-search) that gives an even better improvement.

- During the LK-search, we never delete an edge we have added by an LK-move, so we consider at most *n*-2 additional LK-moves (“taboo criterion”). We keep track of the $\leq n$ solutions and take the best one.

- During the LK-search, the next move we consider is the best LK-move we can make. **It could be an uphill move.**

- We only allow one-trees lighter than the current tour. Thus, we can use neighbor lists to speed up finding the next move.

Iterated Lin-Kernighan

- After having completed a Lin-Kernighan run (i.e., 3-OPT, boosted with LK-searches), make a **random** 4-OPT move and do a new Lin-Kernighan run.

- Repeat for as long as you have time. Keep track of the best solution seen.

- The 4-opt moves are restricted to **double bridge** moves (turning A_1 A_2 A_3 A_4 into A_2 A_1 A_4 A_3.)

Latest TSP algorithms

LKH-3 **Version 3.0.2 (April 2018)**

By Keld Helsgaun

http://www.akira.ruc.dk/~keld/research/LKH-3/