1 Lecture 2d: Complexity Theory

1.1 @luk036

2022-09-21

1.2 Overview

- Complexity theory
- NP-completeness.
- Approximation classes
- Books and online resources.

1.3 Complexity Theory

- Big O-notation: $\mathrm{O}(N)$, $\mathrm{O}(N \log N)$, $\mathrm{O}(N^2)$, $\mathrm{O}(N!)$ …
- Interest in discrete problems in which N is large.
- Indeed, N could be very large (multi-million) in EDA problems, except:
  - Pins of a signal net (usually < 200)
  - Vertices of polygon shapes (usually < 100)
  - Number of routing layers (usually < 10)
- Many Physical Design problems are geometrically related. Complexity (either time or space) could be reduced by exploiting properties such as locality, symmetry, planarity, or triangle inequality.

1.4 NP-completeness

- Many EDA problems are in fact NP-hard.
- Whereas, some NP-complete problems admit good approximations with guarantee performance ratio (*pseudo-polynomial*). E.g. bin-packing problem and knapsack problem.
- Whereas, some NP-complete problems (e.g. SAT) are intrinsically not "approximatable" unless P=NP.
- See the book "Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties" for more details.

1.5 Approximation Classes

- NPO-hard
- APX-hard
- PTAS: polynomial-time approximation scheme

- FPTAS: Fully PTAS (pseudo-polynomial)

  P < FPTAS < PTAS < APX < NPO

1.6 E.g. Minimum Vertex Cover

- Instance: Graph $G = (V, E)$
- Solution: A vertex cover for G, i.e., a subset V' such that, for each edge $(u, v) \in E$, at least one of u and v belongs to V'
- Measure: Cardinality of the vertex cover, i.e. $|V'|$
- Bad News: APX-complete.
- Comment: Admits a PTAS for *planar* graphs [Baker, 1994]. The generalization to k-hypergraphs, for $k > 1$, is approximable within k [Bar-Yehuda and Even, 1981] and [Hochbaum, 1982a]. (HW: Implement the algorithms.)
- Garey and Johnson: GT

1.7 Minimum Maximal Matching

- Instance: Graph $G = (V, E)$.
- Solution: A maximal matching E', i.e., a subset E' such that no two edges in E' shares a common endpoint and every edge in $E - E'$ shares a common endpoint with some edge in E'.
- Measure: Cardinality of the matching, i.e. $|E'|$.
- Bad News: APX-complete [Yannakakis and Gavril, 1980]
- Comment: Transformation from Minimum Vertex Cover (HW: Implement the algorithm)
- Garey and Johnson: GT10

1.8 Minimum Steiner Tree

- Instance: Complete graph $G = (V, E)$, a metric given by edge weights $s : E \mapsto N$ and a subset $S \subset V$ of required vertices.
- Solution: A Steiner tree, i.e., a sub-tree of G that includes all the vertices in S.
- Measure: The sum of the weights of the edges in the sub-tree.
- Bad News: APX-complete.
- Garey and Johnson: ND12

1.9 Minimum Geometric Steiner Tree

- Instance: Set $P \subset Z \times Z$ of points in the plane.
- Solution: A finite set of Steiner points, i.e., $Q \subset Z \times Z$
- Good News: Admits a PTAS [Arora, 1996]
- Comment: Admits a PTAS for any *geometric space* of constant dimension d, e.g. in the rectilinear metric [Arora, 1997].
- Garey and Johnson: ND13

1.10 Traveling Salesman

- Instance: Set C of m cities, distances $d(c_i, c_j) \in N$ for each pair of cities $c_i, c_j \in C$.
- Solution: A tour of C, i.e., a permutation $\pi : [1..m] \mapsto [1..m]$.
- Measure: The length of the tour.

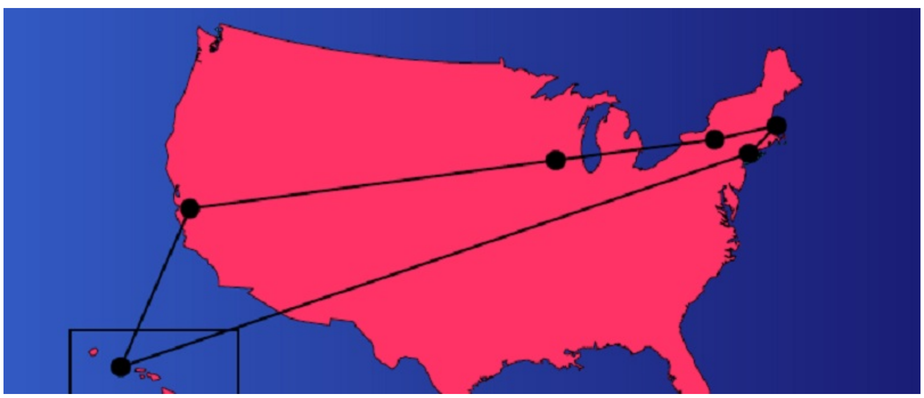

Figure 1: TSP

1.11 Traveling Salesman

- Bad News: NPO-complete
- Comment: The corresponding maximization problem (finding the tour of maximum length) is approximable within 7/5 if the distance function is *symmetric* and 63/38 if it is asymmetric [Kosaraju, Park, and Stein, 1994]
- Garey and Johnson: ND22

1.12 Minimum *Metric* TSP

- Instance: Set C of m cities, distances $d(c_i, c_j) \in N$ satisfying the *triangle inequality* (i.e. $d(a, b) + d(b, c) \geq d(a, c)$)
- Solution: A permutation $\pi : [1..m] \mapsto [1..m]$.
- Measure: The length of the tour.
- Good news: Approximable within 3/2 [Christofides 76]
- Bad News: APX-complete.
- Comment: A variation in which vertices can be revisited and the goal is to minimize the sum of the latencies of all vertices, where the latency of a vertex c is the length of the tour from the starting point to c, is approximable within 29 and is APX-complete

1.13 Minimum Geometric TSP

- Instance: Set $C \subset Z \times Z$ of m points in the plane.
- Solution: A tour of C, i.e., a permutation $\pi : [1..m] \mapsto [1..m]$.
- Measure: The length of the tour, where the distance is the discretized Euclidean length.
- Good news: Admits a PTAS [Arora, 1996]
- Comment: In $\mathbb{R}^m$ the problem is APX-complete for any l_p metric [Trevisan, 1997].
- Garey and Johnson: ND23

1.14 Application - Punching Machine

1.15 Summary

- Some problems are intrinsically hard – even good approximation does not exist unless P=NP (NPO-complete). In such cases, heuristic methods are used (see the [next lecture]).
- “Better” algorithm could be obtained by exploiting more problem’s properties: locality, symmetry, sparsity, planarity, convexity, monotonity, ... etc.

1.16 Books and Online Resources

- G. Ausiello et al. Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties. Springer, 1999. (O224 C737)

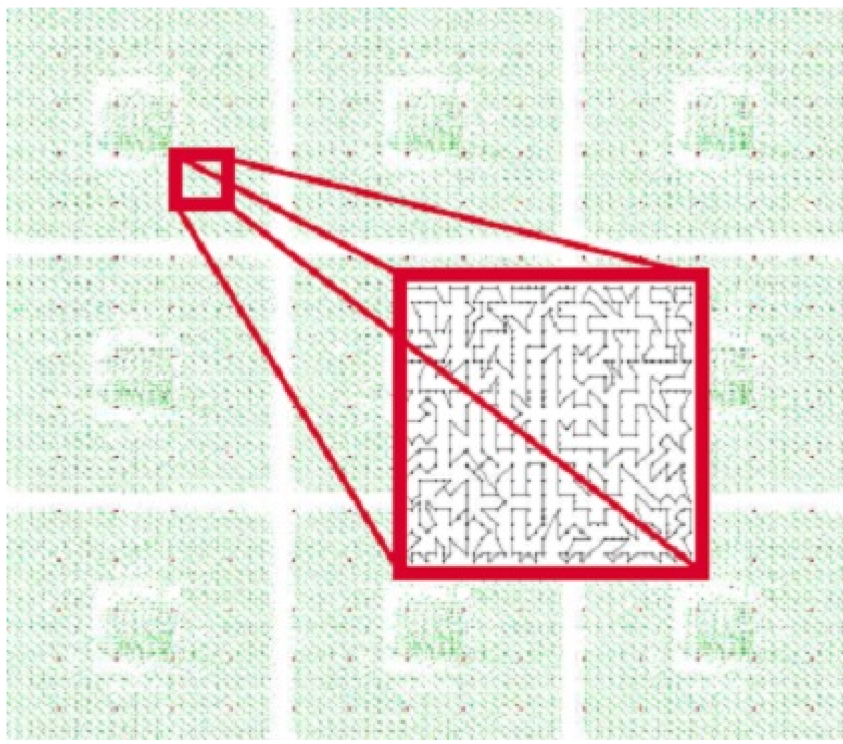

Figure 2: TSP

- M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, 1979.

2 Lecture 2e: Algorithmic Paradigms

2.1 @luk036

2022-09-21

2.2 Overview

- Greedy approach
- Mathematical programming
- Primal-dual algorithm
- Randomized method
- Dynamic programming
- Local search
- Simulated annealing
- Books and online resource

2.3 Greedy Approach

- Excellent for Minimum Spanning Tree (MST) and Channel Routing Problem
  - Obtain optimal solution
- Not bad for Knapsack problem
  - At least half of optimal solution
- Very bad for Feedback Arc Removal problem
  - Even worse than a naïve method: randomly remove edges when traversing a graph, then reverses the set if $|E'|$ is greater than $0.5|E|$.
- Question: Any theory to predict the performance?

2.4 Knapsack Problem

.pull-left[

- A thief considers taking b pounds of loot . The loot is in the form of n items, each with weight a_i and value p_i. Any amount of an item can be put in the knapsack as long as the weight limit b is not exceeded

] .pull-right[

]

2.5 Greedy Approach

- Take as much of the item with the highest value per pound (p_i/a_i) as you can. If you run out of that item, take from the next highest (p_i/a_i) item.

Figure 3: knapsack

Continue until knapsack is full.

2.6 Program 1: Greedy Knapsack

- **Input**: Set of n items, for each $x_i \in X$, values p_i, a_i, positive integer b;
- **Output**: Subset $Y \subset X$ such that $\sum a_i \leq b$;
- Sort X in non-increasing order with respect to the ratio p_i/a_i;
- Let $(x_1, x_2, ..., x_n)$ be the sorted sequence
- $Y := 0$;
- **for** i:=1 **to** n **do**
  - **if** $b \geq a_i$ **do**
    - $Y := Y \cup \{x_i\}$;
    - $b := b - a_i$;
- **return** Y

2.7 C++ code

```
template <class InputIt, typename T, typename F1, typename F2>
InputIt greedy_knapsack(InputIt first, InputIt last,
                        const T& b, F1&& price, F2&& weight)
{
    using Item = typename InputIt::value_type;
    std::sort(first, last, [&](const Item& i1, const Item& i2) {
        return weight(i1) * price(i2) < weight(i2) * price(i1);
    });
    T init(0);
    InputIt it = std::find_if(first, last, [&](Item& i) {
        return (init += weight(i)) > b;
    });
    return it;
}
```

- Test program can be found in http://ideone.com/9ZK6ol.

2.8 Can the thief do better?

- Theorem 1. Let $\mathrm{mH}(x) = \max(p\max, \mathrm{mGR}(x))$, where $p\max$ is the maximum profit of an item in x. Then $\mathrm{mH}(x)$ satisfies the following inequality: $\mathrm{m}(x)/\mathrm{mH}(x) < 2$. (p.42) ($\mathrm{m}(x)$ is the optimal solution)

- As a consequence of the above theorem, a simple modification of Program 1 allows us to obtain a provably better algorithm.

- HW: Implement the algorithm using C++ Template technique and iterators (generic programming style)

2.9 Linear Programming Relaxation

- Formulate a problem as an integer linear program.
- By relaxing the integrality constraints we obtain a new linear program, whose optimal solution can be found in polynomial time.
- This solution, in some cases, can be used to obtain a feasible solution for the original integer linear program, by "rounding" the values of the variables that do not satisfy the integrality constraints.

2.10 Weighted Vertex Cover

- Given a weighted graph $G = (V, E)$, Minimum Weighted Vertex Cover (MWVC) can be formulated as the following integer program ILPVC(G):
- Minimize $\sum_{vi \in V} c_i x_i$
- Subject to $x_i + x_j \geq 1$ for all $(v_i, v_j) \in E$
- $x_i \in \{0, 1\}$ for all $v_i \in V$

2.11 Program 2.6 Rounding WVC

- **Input** Graph $G = (V, E)$ with non-negative vertex weights;
- **Output** Vertex cover V' of G;
- Let ILPVC be the linear integer programming formulation of the problem;
- Let LPVC be the problem obtained from ILPVC by relaxing the integrality constraints;
- Let $x(G^*)$ be the optimal solution for LPVC;
- $V' := \{v \mid x_v(G^*) \geq 0.5\}$;
- **return** V'

2.12 Linear Programming

- Theorem 2.15. Given a graph G with non-negative vertex weights, Program 2.6 finds a feasible solution of MWVC with value mLP(G) such that mLP(G)/m(G^*) ≤ 2.
- Problem: need to solve the LP optimally.

2.13 Primal-Dual WVC

- **Input** Graph $G = (V, E)$ with non-negative vertex weights;
- **Output** Vertex cover V' of G;
- Let DLPVC be the dual of the LP relaxation of ILPVC;
- **for** each dual variable y of DLPVC **do** $y := 0$;
- $V' := 0$;
- **while** V' is not a vertex cover **do**
    - Let (v_i, v_j) be an edge not covered by V';

- Increase y_{ij} until a constraint of DLPVC becomes tight;
- **if** sum$(y_{ij}|(i,j) \in E)$ is tight **then**
  - $V' := V' \cup \{v_i\}$ (* the i-th dual constraint is tight *)
- **else**
  - $V' := V' \cup \{v_j\}$ (* the j-th dual constraint is tight *)

- **return** V'

2.14 Primal-Dual WVC

- Theorem 2.16. Given a graph G with non-negative weights, Program 2.7 finds a feasible solution of MWVC such that $m_{\text{PD}}(G)/m(G^*) \leq 2$. (p. 69)
- Much faster than Program 2.6 (only take linear time) because we don't need to solve the LP optimally.
- Bonus: Sum of dual variables y_{ij} gives the lower bound of the optimal solution.

2.15 Program - Random WVC

- **Input** Graph $G = (V, E)$, weight function $w : V \mapsto N$;
- **Output** Vertex cover U;
- $U := \emptyset$;
- **while** E is not empty **do**
  - Select an edge $e = (v, t) \in E$;
  - Randomly choose x from $\{v, t\}$ with $\Pr\{x = v\} = w(t)/(w(v)+w(t))$;
  - $U := U \cup \{x\}$;
  - $E := E - \{e \mid x \text{ is an endpoint of } e\}$
- **return** U

2.16 Randomized Algorithms

- In many cases, a randomized algorithm is either simpler or faster (or both) than a deterministic algorithm.
- However, it does not guarantee that the algorithm always finds a good approximation solution.
- Theorem 5.1. The expect measure of the solution returned by the previous algorithm satisfied the following inequality:

$$E[m_{\text{RWVC}}(x)] \leq 2m^*(x)$$

- HW: Implement MWVC solvers using all the above methods. Also extend all the methods to handle hypergraph

2.17 Dynamic Programming (I)

- One passenger wants to go from city A to city H through the *shortest path* according to the map on the right, where number of indicate distance between corresponding cities.
- Reference: Pablo Pedregal, *Introduction to Optimization*, chapter 5.8, Springer, 2003

2.18 Dynamic Programming (II)

- Proposition 5.24 (Fundamental property of dynamic programming)
  - If $S(t_j, x)$ denotes the optimal cost from (t_0, x) to (t_j, x)
  - then we must have S(t_{j+1}, y) = minj [S(t_j, x) + c(j,x,y)]

2.19 Dynamic Programming (III)

- According to Proposition 5.24, we must proceed successively to determine S(t_j, x) for each x in Aj to end with S(t_n, x_n). In the proposed example, we have four stages t_0, t_1, t_2, t_3 with associated sets of feasible states
  - A0 = {A}, A1 = {B, C, D}, A2 = {E,F,G}, A3 = {H}
- For each city in A1, there is a unique path from A, so that it must be optimal, and
  - S(t_1, B) = 7, S(t_1, C) = 4, S(t_1, D) = 1.
- For each city in A2, we determine the optimal cost based on the fundamental property of dynamic programming,
  - S(t_{j+1}, y) = minj [S(t_j, x) + c(j,x,y)]

2.20 Local Search

- **Input**: Instance x;
- **Output**: Solution s
- s := initial feasible solution s_0;
- (* $\mathcal{N}$ denotes the neighborhood function *)
- **repeat**
  - Select any $s' \in \mathcal{N}(x, s)$ not yet considered;
  - **if** $m(x, s') < m(x, s)$ **then**
    - $s := s'$;
- **until** all solutions in $\mathcal{N}(x, s)$ have been visited;
- **return** s;

2.21 Simulated Annealing

- **Input**: Instance x;
- **Output**: Solution s

- $\tau := t$;
- $s :=$ initial feasible solution s_0;
- **repeat**
  - **for** l times **do**
    - Select any unvisited $s' \in \mathcal{N}(x, s)$
    - **if** $(m(x, s') < m(x, s))$
      - $s := s'$;
    - **else**
      - $\delta := m(x, s') - m(x, s)$;
      - $s := s'$ with probability $\exp(-\delta/t)$;
  - $\tau := r \cdot \tau$; (* update of temperature *)
- **until** FROZEN;
- **return** s;

2.22 Books and Online Resources

- G. Ausiello et al. Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties. Springer, 1999. (O224 C737)
- M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, 1979.
- Pablo Pedregal. Introduction to Optimization. Springer, 2003 (O224 P371)