1 Lecture 2c: Introduction to Convex Programming

1.1 Abstract

This lecture provides an introduction to the convex programming and covers various aspects of optimization. The lecture begins with an overview of optimization, including linear and nonlinear programming, duality and convexity, and approximation techniques. It then delves into more specific topics within continuous optimization, such as linear programming problems and their standard form, transformations to standard form, and the duality of linear programming problems. The lecture also touches on nonlinear programming, discussing the standard form of an NLPP (nonlinear programming problem) and the necessary conditions of optimality known as the Karush-Kuhn-Tucker (KKT) conditions. Convexity is another important concept explored in the document, with explanations on the definition of convex functions and their properties. The lecture also discusses the duality of convex optimization problems and their usefulness in computation. Finally, the document briefly mentions various unconstrained optimization techniques, descent methods, and approximation methods under constraints.

1.2 Overview

- Introduction
- Linear programming
- Nonlinear programming
- Duality and Convexity
- Approximation techniques
- Convex Optimization
- Books and online resources.

1.3 Classification of Optimizations

- Continuous
  - Linear vs Non-linear
  - Convex vs Non-convex
- Discrete
  - Polynomial time Solvable
  - NP-hard
    - Approximatable
    - Non-approximatable
- Mixed

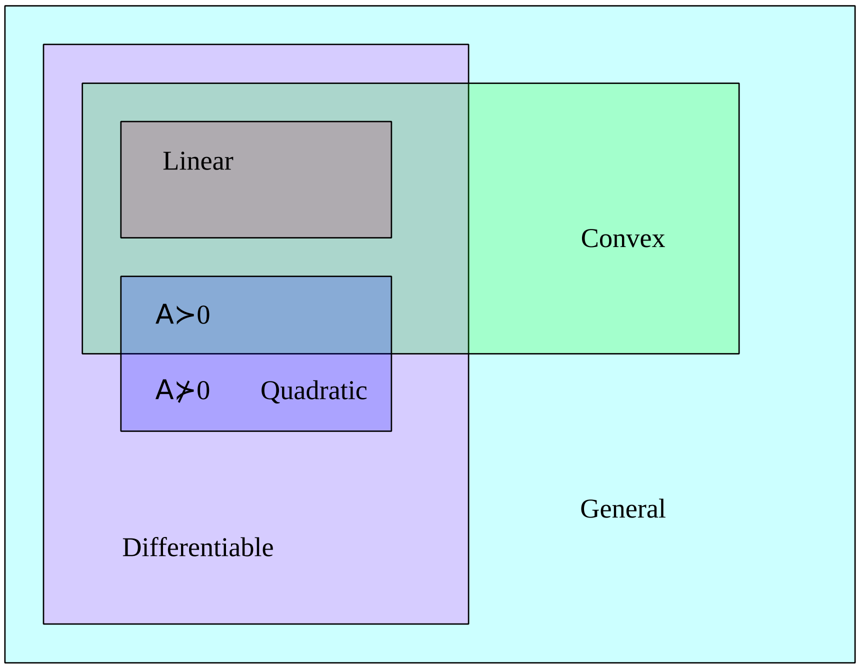

Figure 1: classification

1.4 Continuous Optimization

1.5 Linear Programming Problem

- An LPP in standard form is:

$$\min\{c^\top x \mid Ax = b, x \geq 0\}.$$

- The ingredients of LPP are:
  - An $m \times n$ matrix A, with $n > m$
  - A vector $b \in \mathbb{R}^m$
  - A vector $c \in \mathbb{R}^n$

1.6 Example

$$\begin{array}{lll}
\text{minimize} & 0.4x_1 + 3.4x_2 - 3.4x_3 & \\
\text{subject to} & 0.5x_1 + 0.5x_2 & = 3.5 \\
& 0.3x_1 - 0.8x_2 + 8.4x_2 & = 4.5 \\
& x_1, x_2, x_3 \geq 0 &
\end{array}$$

1.7 Transformations to Standard Form

- Theorem: Any LPP can be transformed into the standard form.

- Variables not restricted in sign:
  - Decompose x to two new variables $x = x_1 - x_2, x_1, x_2 \geq 0$
- Transforming inequalities into equalities:
  - By putting slack variable $y = b - Ax \geq 0$
  - Set $x' = (x, y), A' = (A, 1)$
- Transforming a max into a min
  - max(expression) = min(−expression);

1.8 Duality of LPP

- If the primal problem of the LPP: $\min\{c^{\mathsf{T}}x \mid Ax \geq b, x \geq 0\}$.
- Its dual is: $\max\{y^{\mathsf{T}}b \mid A^{\mathsf{T}}y \leq c, y \geq 0\}$.
- If the primal problem is: $\min\{c^{\mathsf{T}}x \mid Ax = b, x \geq 0\}$.
- Its dual is: $\max\{y^{\mathsf{T}}b \mid A^{\mathsf{T}}y \leq c\}$.

1.9 Nonlinear Programming

- The standard form of an NLPP is

$$\min\{f(x) \mid g(x) \leq 0, h(x) = 0\}.$$

- Necessary conditions of optimality, Karush- Kuhn-Tucker (KKT) conditions:
  - Gradient Condition: f(x) + µ g(x) + h(x) = 0, where f(x), g(x), and h(x) are the gradients of the objective function, inequality constraints, and equality constraints, respectively. This condition states that the sum of the directional derivatives of the objective function and the constraints must be zero at the optimal solution.
  - Complementary Slackness Condition: µg(x) = 0, where µ is a Lagrange multiplier associated with the inequality constraints. This condition implies that either the constraint is inactive (g(x) 0) or its corresponding Lagrange multiplier is zero.
  - Feasibility Condition: µ 0, g(x) 0, h(x) = 0. This condition ensures that the inequality and equality constraints are satisfied at the optimal solution.

1.10 What is the significance of the KKT conditions mentioned?

The significance of the KKT conditions lies in their ability to provide necessary conditions for a solution to be optimal in nonlinear programming problems. By satisfying these conditions, a point can be determined as a possible optimal solution. Moreover, if the objective function is strictly convex, and the KKT conditions are satisfied, then the solution obtained is the unique optimal solution. In essence, the KKT conditions serve as a powerful mathematical tool for analyzing and solving optimization problems.

1.11 Convexity

- A function f: $K \subseteq \mathbb{R}^n \mapsto R$ is convex if K is a convex set and $f(y) \geq f(x) + \nabla f(x)(y-x),\ y, x \in K$.
- **Theorem**: Assume that f and g are convex differentiable functions. If the pair (x, m) satisfies the KKT conditions above, x is an optimal solution of the problem. If in addition, f is strictly convex, x is the only solution of the problem.

  (Local minimum = global minimum)

1.12 Duality and Convexity

- Dual is the NLPP:
$$\max\{\theta(\mu, \lambda) \mid \mu \geq 0\},$$
  where $\theta(\mu, \lambda) = \inf_x[f(x) + \mu g(x) + \lambda h(x)]$
- Dual problem is always convex.
- Useful for computing the lower/upper bound.

1.13 Applications

- Statistics
- Filter design
- Power control
- Machine learning
  - SVM classifier
  - logistic regression

class: nord-light, middle, center

2 Convexify the non-convex's

2.1 Change of curvature: square

Transform:
$$0.3 \leq \sqrt{x} \leq 0.4$$
into:
$$0.09 \leq x \leq 0.16\,.$$

Note that $\sqrt{\cdot}$ are **monotonic concave** functions in $(0, +\infty)$.

Generalization: - Consider $|H(\omega)|^2$ (power) instead of $|H(\omega)|$ (magnitude). - square root -> Spectral factorization

2.2 Change of curvature: square

Transform:

$$x^2 + y^2 \geq 0.16, \quad \text{(non-convex)}$$

into:

$$x' + y' \geq 0.16, \quad x', y' \geq 0$$

Then:

$$x_{\text{opt}} = \pm\sqrt{x'_{\text{opt}}}, \quad y_{\text{opt}} = \pm\sqrt{y'_{\text{opt}}}.$$

2.3 Change of curvature: sine

Transform:

$$\sin^2 x \leq 0.4, \quad 0 \leq x \leq \pi/2$$

into:

$$y \leq 0.4, \quad 0 \leq y \leq 1$$

Then:

$$x_{\text{opt}} = \sin^{-1}(\sqrt{y_{\text{opt}}}).$$

Note that $\sin(\cdot)$ are monotonic concave functions in $(0, \pi/2)$.

2.4 Change of curvature: log

Transform:

$$\pi \leq x/y \leq \phi$$

into:

$$\pi' \leq x' - y' \leq \phi'$$

where $z' = \log(z)$.

Then:

$$z_{\text{opt}} = \exp(z'_{\text{opt}}).$$

Generalization: - Geometric programming

2.5 Change of curvature: inverse

Transform:

$$\log(x) + \frac{c}{x} \leq 0.3, \; x > 0$$

into:

$$-\log(y) + c \cdot y \leq 0.3, \; y > 0\,.$$

Then:

$$x_{\text{opt}} = y_{\text{opt}}^{-1}.$$

Note that $\sqrt{\cdot}$, $\log(\cdot)$, and $(\cdot)^{-1}$ are monotonic functions.

2.6 Generalize to matrix inequalities

Transform:

$$\log(\det X) + \mathrm{Tr}(X^{-1}C) \le 0.3,\ X \succ 0$$

into:

$$-\log(\det Y) + \mathrm{Tr}(Y \cdot C) \le 0.3,\ Y \succ 0$$

Then:

$$X_{\mathrm{opt}} = Y_{\mathrm{opt}}^{-1}.$$

2.7 Change of variables

Transform:

$$(a + b \cdot y)x \le 0,\ x > 0$$

into:

$$a \cdot x + b \cdot z \le 0,\ x > 0$$

where $z = yx$.

Then:

$$y_{\mathrm{opt}} = z_{\mathrm{opt}} x_{\mathrm{opt}}^{-1}$$

2.8 Generalize to matrix inequalities

Transform:

$$(A + BY)X + X(A + BY)^T \prec 0,\ X \succ 0$$

into:

$$AX + XA^T + BZ + Z^T B^T \prec 0,\ X \succ 0$$

where $Z = YX$.

Then:

$$Y_{\mathrm{opt}} = Z_{\mathrm{opt}} X_{\mathrm{opt}}^{-1}$$

2.9 Some operations that preserve convexity

- $-f$ is concave if and only if f is convex.
- Nonnegative weighted sums:
  - if $w_1, \ldots, w_n \ge 0$ and $f_1, \ldots, f_n$ are all convex, then so is $w_1 f_1 + \cdots + w_n f_n$. In particular, the sum of two convex functions is convex.
- Composition:
  - If f and g are convex functions and g is non-decreasing over a univariate domain, then $h(x) = g(f(x))$ is convex. As an example, if f is convex, then so is $e^{f(x)}$, because e^x is convex and monotonically increasing.

  - If f is concave and g is convex and non-increasing over a univariate domain, then $h(x) = g(f(x))$ is convex.
  - Convexity is invariant under affine maps.

2.10 Other thoughts

- Minimizing any quasi-convex function subject to convex constraints can easily be transformed into a convex programming.
- Replace a non-convex constraint with a sufficient condition (such as its lower bound). Less optimal.
- Relaxation + heuristic
- Decomposition

2.11 Unconstraint Techniques

- Line search methods
- Fixed or variable step size
- Interpolation
- Golden section method
- Fibonacci's method
- Gradient methods
- Steepest descent
- Quasi-Newton methods
- Conjugate Gradient methods

2.12 General Descent Method

```
1. **Input**: a starting point $x \in \text{dom } f$
2. **Output**: $x^*$
3. **repeat**
    1. Determine a descent direction $p$.
    2. Line search. Choose a step size $\alpha > 0$.
    3. Update. $x := x + \alpha p$
4. **until** stopping criterion satisfied.
```

2.13 Some Common Descent Directions

- Gradient descent: $p = -\nabla f(x)^\mathsf{T}$
- Steepest descent:
  - $\triangle x_{nsd} = \text{argmin}\{\nabla f(x)^\mathsf{T} v \mid \|v\| = 1\}$
  - $\triangle x = \|\nabla f(x)\| \triangle x_{nsd}$ (un-normalized)
- Newton's method:
  - $p = -\nabla^2 f(x)^{-1} \nabla f(x)$
- Conjugate gradient method:
  - p is "orthogonal" to all previous p's
- Stochastic subgradient method:

    - p is calculated from a set of sample data (instead of using all data)
- Network flow problems:
    - p is given by a "negative cycle" (or "negative cut").

2.14 Approximation Under Constraints

- Penalization and barriers
- Dual method
- Interior Point method
- Augmented Lagrangian method

2.15 Books and Online Resources

- Pablo Pedregal. Introduction to Optimization, Springer. 2003 (O224 P371)
- Stephen Boyd and Lieven Vandenberghe, Convex Optimization, Dec. 2002
- Mittlemann, H. D. and Spellucci, P. Decision Tree for Optimization Software, World Wide Web, http://plato.la.asu.edu/guide.html, 2003

3 Lecture 2d: Complexity Theory

3.1 @luk036

2022-09-21

3.2 Overview

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

3.3 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.

3.4 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.

3.5 Approximation Classes

- NPO-hard
- APX-hard
- PTAS: polynomial-time approximation scheme
- FPTAS: Fully PTAS (pseudo-polynomial)

  P < FPTAS < PTAS < APX < NPO

3.6 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

3.7 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

3.8 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

3.9 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

3.10 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.

3.11 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

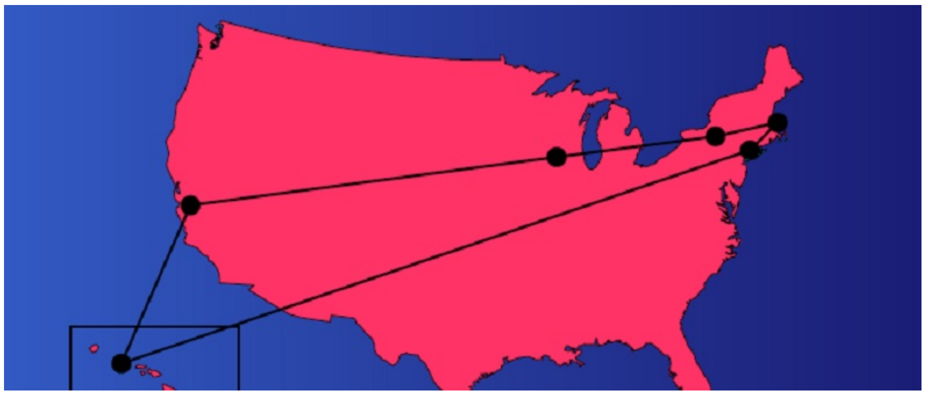

Figure 2: TSP

3.12 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

3.13 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

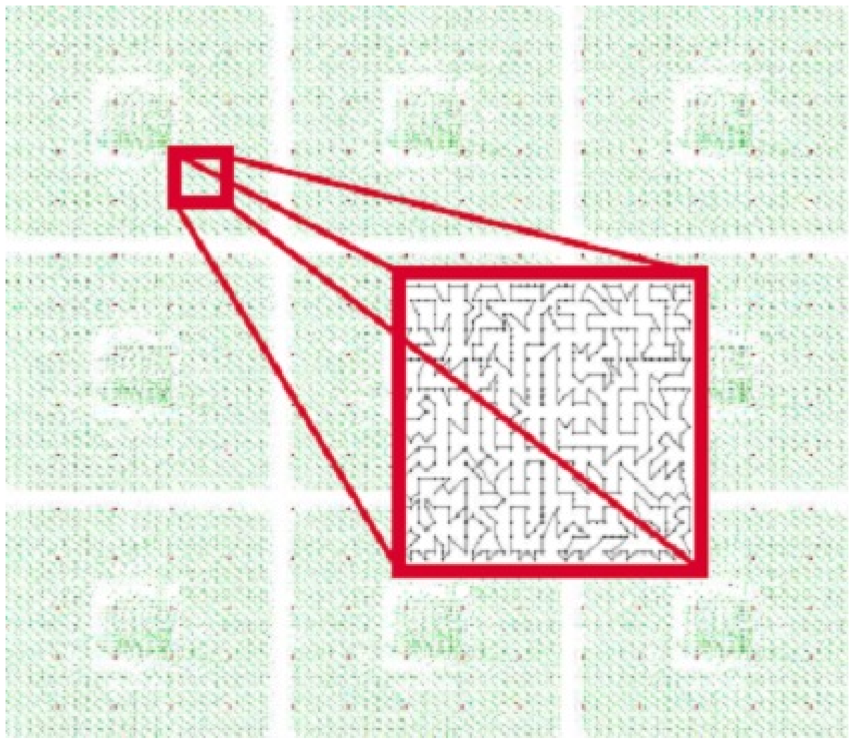

Figure 3: TSP

3.14 Application - Punching Machine

3.15 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.

3.16 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.

4 Lecture 2e: Algorithmic Paradigms

4.1 @luk036

2022-09-21

4.2 Overview

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

4.3 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?

4.4 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[

Figure 4: knapsack

]

4.5 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. Continue until knapsack is full.

4.6 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 $Y$
```

4.7 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.

4.8 Can the thief do better?

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

4.9 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.

4.10 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 $v_i \in V$

4.11 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'

4.12 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.

4.13 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 $V'$
```

4.14 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.

4.15 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 $\Pr\{x = v\} = w(t)/(w(v)+w(t))$;
    – $U := U \cup \{x\}$;
    – $E := E - \{e \mid x$ is an endpoint of $e\}$
• return $U$
```

4.16 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

4.17 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

4.18 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)]

4.19 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)]

4.20 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 $s$;
```

4.21 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** $s$;
```

4.22 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)