Algorithm Miscellany

Jiyue Wang (jiyue@stanford.edu)

March 17, 2015

1 Job Scheduling / Makespan problem

Given m machines and n jobs with workload $p_1, ..., p_n$, give a schedule that

$$\min_{m \text{ machines}} \max\{\text{workload for a single machine}\}$$

This problem is NP hard. We can use a Greedy algorithm to achieve 4/3 approximation(TODO). If the number of distinct workload is restricted to k , there is a DP solution of $O(n^{2k})$ which gives the exact solution to the corresponding decision problem : Can the m machines finish the job within T times. (Suppose the workload is the time it takes to complete the job for one machine.)

Suppose there are b_i jobs for workload p_i, and we have $(b_1, ..., b_k)$ jobs in total. Let $M(c_1, ..., c_k)$ denote the minimum number of machines needed to complete $(c_1, ..., c_k)$ jobs within time T. Then it's easy to check whether $M(c_1, ..., c_k) > 1$ and quitely clearly, $M(0, ..., 0) = 0$

```
for c_1 in range(1, b_1 + 1):
  for c_2 in range(1, b_2 + 1):
    ...
    for c_k in range(1, b_k + 1):
      if M(c_1, ..., c_k) > 1:
        S = {(j_1, ..., j_k)| j_i < c_i for all i, M(j_1, ..., j_k) = 1}
        M(c_1, ..., c_k) = 1 + min(M(c_1 - j_1, ..., c_k - j_k) over S)
if M(b_1, ..., b_k) > m:
  return False
else:
return True
```

2 Knapsack Problem

Given a knapsack capacity B, and a set N of n items. Each item has a weight $w_i \geq 0$ and a profit $p_i \geq$. The goal is to find a subset of the items $S \subset N$ s.t. $w(S) = \sum_{i \in S} w_i \leq B$ and $p(S) = \sum_{i \in S} p_i$ is maximized.

If we further assume that the profits and weights are integers, we can find a dynamic programming solution to the problem in either $O(nB)$ or $O(nP)$ time, where $P = \sum_{i \in N} p_i$. Even so, it gives a pseudo-polynomial time algorithm and the problem is still NP-hard.

```
Let profit[i][b] be the maximum profit we can get given item $1, 2, ..., i$ and capacity $b$.
initilize profit with profit[0][0] = 0
for i in range(1, n + 1):
for b in range(1, B + 1):
if $w_i$ ¿ b:
profit[i][b] = profit[i - 1][b]
else:
profit[i][b] = max(profit[i - 1][b], profit[i][b - $w_i$] + $w_i$)
return profit[n][B]
```

3 (Minimum Weight) Perfect Matching/ Minimum Weight Cycle Cover

A **perfect matching** is a matching which matches all vertices of the graph. That is, every vertex of the graph is incident to exactly one edge of the matching. [1] The mini-weight perfect matching problem can be solved in polynomial time.

A **minimum weight cycle cover** of a graph is stated as follows. Let $H = (V, E)$ be a directed graph with non-negative arc weights given by $w : E \to R^+$. We wish to find a minimum weight collection of vertex-disjoint directed cycles in H such that every vertex is in exactly one of those cycles. [2]. We need this to give an approximation algo to the Asymmetric Traveling Salesman Problem (ATSP).

Algorithm: For each node $v \in V$, split it into v^+ and v^-, where v^+ is the 'in-node' and v^- is the 'out-node'. Solve the minimum perfect matching problem on the new graph. Done.

4 Matroids

4.1 Definitions

A matroid $M = (E, I)$ is defined on a ground set E and an independent set I, where I is a collection of subsets of E and I must satisfy two axioms:

- If $X \subseteq Y$, and $Y \in I$, then $X \in I$
- If $X, Y \in I$ and $|Y| > |X|$, then $\exists e \in Y \backslash X$, s.t. $X \cup \{e\} \in I$

Remark 1 *Each **maximal** independent set in I has the same cardinality. We call such independent set **base**.*

Example 1 ***Graphic Matroids**: Given a graph $G = (V, E)$, define the independent set to be thoses subsets of edges which are forests.*

Proof:

- If $Y \in I$ is a forest, then $\forall X \subseteq Y$ has no circle, indicating that $X \in I$

[1]Matching (graph theory), wikipediea
[2]HW0/6, Spring 2011, CS 598CSC, University of Illinois

- For $Y \in I$, let $K(Y)$ denote the number of connected components of Y (each isolated vertex is regarded as one component), then $K(Y) = |V| - |Y|$. Thus, if $X, Y \in I$ and $|X| < |Y|$, then $K(X) > K(Y)$. We can always find an edge $e \in Y \backslash X$ which connects two components in X and we still have $\{e\} \cup X \in I$

$\square$

4.2 Matroid Optimization

Given a matroid $M = (E, I)$ and a cost function $c : E \to \mathbb{R}$, find $S \in I$ which maximizes $c(S) = \sum_{e \in S} c(e)$. If the cost is nonnegative, then it's equivalent to find the maximum cost base of I. If there is a negative e, then it cannot appear in the optimal set, thus we can simply remove it.

Remark 2 *If we think of this problem on graphic matroids, this is exactly the same as finding the maximum spanning tree (solved by greedy algo).*

Similar to the algo for finding MST, we can solve this problem using the greedy algorithm :

```
Sort the elements s.t. $c(e_1) \geq c(e_2) \geq ... \geq c(e_{|M|})$
$S_0 = \varnothing$; k = 0
res = []
for j in range(1, —E— + 1):
if $S_k + e_j \in I$ :
k += 1
$S_k = S_{k-1} + e_j$
res.append($e_j$)
return res
```

Theorem 1 *For any matroid $M = (E, I)$, the greedy algorithm above finds, for every k, an independent set S_k of maximum cost among all independent sets of size k.*

Proof: Denote the edges in res as $e_1, ..., e_k$. Suppose not, let $S_k = \{e_1, ..., e_k\}$ with $c(e_1) \geq c(e_2) ... \geq c(e_k)$ and suppose T_k has greater cost, where $T_k = \{t_1, ..., t_k\}$ (as optimal is a base, $|T_k| = k$. Let p be the first index s.t. $c(t_p) > c(e_p)$. Let $A = \{t_1, ..., t_p\}$, $B = \{e_1, ..., e_{p-1}\}$. Since $|A| > |B|$, there exists $t_i \notin B$ s.t. $t_i + B \in I$, since $t_i \geq t_p > e_p$, t_i should be selected when we are choosing e_p. $\square$

Remark 3 *The greedy algorithm actually characterizes matroids. If M is an independence system, i.e. it satisfies (Axiom 1), then M is a matroid iff the greedy algo finds a maximum cost set of size k for every k and every cost function.*

Here is a **not that related problem**: Given a Minimum Spanning Tree $T = \{e_1, ..., e_{n-1}\}$ of a graph $G = (V, E)$, find the second-minimum spanning tree of G.

The straight forward approach is to delete each edge in T and run the MST algorithm. This repeats n - 1 times, then pick the minimum. The greedy MST algo takes $O(n^2 \log n)$ time, so the total is $(n^3 \log n)$

What about just delete one edge in T and try the rest of the edges and see if it's the second-MST. This should cost $O(n^3)$. As when we delete an edge, we are left with two spanning trees, which should be the corresponding MST in each subgraph. We just need to find another bridge to connect the two trees.

4.3 Totally Unimodular Matrices(TUM)

A matrix A is **TUM** if every square submatrix M of A has $\det(M) \in \{+1, -1, 0\}$. This indicates every entry of A should be +1, -1 or 0.

Example 1 Incidence matrix of a directed graph

Proof Suppose the incidence matrix $S \in R^{|V|\times|E|}$ and $e = (u, v) \in E$ iff $M_{ue} = 1 \wedge M_{ve} = -1$, Use induction on the size n of the submatrix M. When $n = 1$, it's trivial. When $n > 1$, there are three cases.

- Some column of M is all zero, then $\det(M) = 0$
- Some column of M has only one of $\{+1, -1\}$, then induce on $n - 1$
- All columns of M has two entries 1 and -1. Then the row sum yields $\mathbf{0}$, indicating that $\det(M) = 0$

Example 2 A graph is bipartite iff its incidence matrix is TUM. (Here we should redefine the incidence matrix)

Proof $\Leftarrow$: If the graph contains an odd cycle, the submatrix corresponding to the cycle has determinant 2, contradiction.

$\Rightarrow$: Prove by induction. The first two cases are trivial and the same as Example 1. The last case is when all columns of M has exactly two 1. Now permutate the rows to split M into two halves, each representing a group. Now we can multiply the bottom half with -1 to make it a directed graph. Multiplying rows by -1 only changes the sign of the determinant of M. We then apply the result from example 1 to finish the proof.

5 Traveling Salesman's Problem

5.1 Undirected Version

Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?[3]. This problem is NP-hard as if we have a blackbox solving TSP, we could decide the Hamiltonian cycle in an undirected graph G.

Proof: Given a graph $G = (V, E)$, we construct a new weighted **complete** graph $G' = (V, E', W)$ as follows: $w(u, v) = \inf$ if $(u, v) \notin E$, else $w(u, v) = 1$. If the TSP on G' returns $|E|$, then it finds a Hamiltonian cycle in the original graph G; if not, there is no Hamiltonian cycle in G. □

Actually, we cannot have a very good approximation to TSP. If we have an approximation algorithm A for TSP with an approximation ratio $\alpha(|I|)$ (I is an instance of TSP), then we can exactly solve the Hamiltonian cycle problem. [4]

We can relax conditions to get better approximation (actually the following two are equivalent)

- Metric-TSP, where cost of edge satisfies $c(u, v) \leq c(u, w) + c(w, v)$ for $\forall u, v, w \in V$.
- TSP-R, where repitition of vertices allowed. Thus instead of finding a minimum weight Hamiltonian cycle, we now seek a minimum-cost walk which visits each vertex and return to the starting vertex in the end.

[3]wiki

[4]`https://courses.engr.illinois.edu/cs598csc/sp2011/lectures/lecture_2.pdf`

Up to now, Christofides' MST-based algorithm gives the best approximation ratio (3/ 2 -approximation) on Metric-TSP.

5.2 Directed Version: Asymmetric TSP

We can think of this problem as aysmmetric traffic: it could take more time to travel from A to B than from B to A. In this scenario, a MST-based algorithm is no longer helpful. Here use a Cycle Shrinking Algorithm ($\log_2 n$-apprximation, prove by induction)

6 Vertex Cover

6.1 Minimum Vertex Cover (connected graph)

Algorithm:

Find a **maximal matching** M in graph G. Then output the endpoints of edges in M, denoted as

$$S = \cup_{(u,v)\in M}\{u, v\}$$

Analysis:

This is a 2-approximation. The first inequality below is critical

$$|M| \leq OPT \leq |S| \leq 2|M| \leq 2OPT$$

6.2 Minimum Weight Vertex Cover

Use Linear Programming Approximation

TODO

7 Max Cut

7.1 Deterministic Greedy Algorithm

Algorithm:

```
Let V = {v_0}
while There exists v s.t. moving v to the other side increases cut size:
  move v to the other side

Output 2 sets
```

Analysis:

This is a 1/2-approximation algo. As for each vertex v, at least half of the incident edges contribute to the cut. (otherwise, it will be in the other set). Thus

$$\begin{aligned}
GreedyOPT &\geq \frac{1}{2}\sum_{v\in V}\frac{d(v)}{2} = \frac{|E|}{2} \\
OPT &\leq |E| \\
\Rightarrow GreedyOPT &\geq \frac{1}{2}OPT
\end{aligned}$$

7.2 Semidefinite Programming Approximation

TODO

- Solve the SDP and get X
- Do Cholesky factorization on X to get V
- Now convert V back to the original problem. The intuition is that if $X_{ij} = v_i \cdot v_j$ is close to -1, then we should make the cut. In this algorithm, we round v_i to +1 or -1 by a Random Hyperplane algorithm.

Analysis:

This is a randomized approximation algorithm. We have $E[cut] \geq 0.878OPT$.