Approximation Algorithms — **Problem Set 1**
CS 783 / Or 634 Fall 2003 — **Solutions**

Problem 1

Consider a graph G_n on $2n$ vertices $u_0, u_1, \ldots, u_{n-1}$ and $v_0, v_1, \ldots, v_{n-1}$ with the following edges

- edges $(u_i, u_{(i+1)\bmod n})$ of length ε for $i = 0, \ldots, n-1$,
- edges $(v_i, v_{(i+1)\bmod n})$ of length $1+\varepsilon$ for $i = 0, \ldots, n-1$,
- edges (u_i, v_i) of length 1 for $i = 0, \ldots, n-1$.

For simplicity let us assume that n is even. We think of ε as being a small positive number going to 0 in the limit, such as $\varepsilon = 1/n$.

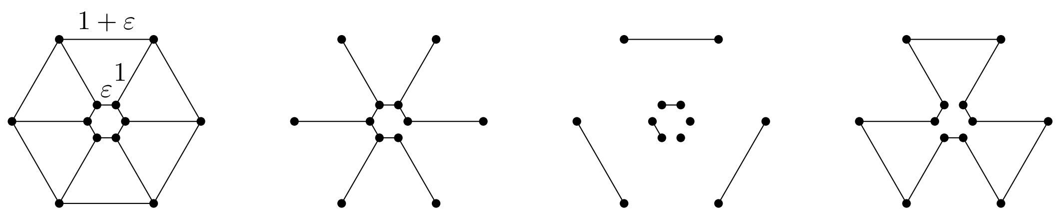

Figure 1: From left to right: the graph $G = G_6$, a minimum cost spanning tree T of G, a minimum cost matching in G of odd-degree vertices of T, and the final tour produced by Christofides's heuristic.

There is a TSP tour in G_n of length $n + 1 + 2(n-1)\varepsilon$: $v_0, v_1, \ldots, v_{n-1}, u_{n-1}, u_0, v_0$.

Christofides's algorithm first builds a minimum spanning tree on G_n. G_n has a unique (up to rotational symmetry) min-cost spanning tree, consisting of edges (u_{i-1}, u_i) for i=1,…,n-1, and edges $(u_i, v_i))$ i=1,…,n-1.

Then it finds a min-cost matching of odd-degree nodes. In our case, this matching includes edges (v_1, v_2), (v_3, v_4), …, v_{n-3}, v_{n-2}.

The final tour produced by Christofides's heuristic is $u_0, u_1, v_1, v_2, u_2, u_3, v_3, \ldots, v_{n-2}, v_{n-1}, u_0$ and has cost $(3/2 + \varepsilon)n$.

Problem 2

Note that a TSP tour is a feasible 2-matching, and hence the cost of the minimum 2-matching M is a lower bound on the TSP cost. Also, since each edge costs at least 1, any TSP tour costs at least n, where n is the number of vertices.

We give an algorithm that constructs a TSP tour of cost at most $\text{cost}(M) + n/3$ which is clearly at most $\frac{4}{3}$ of the optimum TSP cost. The algorithm is:

1. First find a min-cost 2-matching M in G. Note that since each vertex has degree 2 in M, M must consist of one or more disjoint cycles $C_1, C_2, \ldots, C_k$. Each cycle has length at least 3, hence $k \le n/3$.

2. We merge the cycles into one cycle, creating a TSP tour. From each cycle C_i, pick two consecutive vertices v_i and v'_i. Delete each edge (v_i, v'_i) and add edges $(v'_i v_{i+1})$ for $i = 1 \ldots, n$ (wrapping indices modulo n). We clearly created a cycle spanning all vertices. What is the cost? We deleted k edges of length at least 1, and added k edges of length at most 2. Hence in total, we increased the cost by at most k. Thus the resulting cycle costs at most $\text{cost}(M) + k \le \text{cost}(M) + n/3$.

Problem 3

The "classical" example (see Vazirani's book) has n elements $1 \dots n$, and $n+1$ sets $S_0, S_1, S_2, \dots S_n$. Each set S_i for $i = 1 \dots n$ consists of a single element i and has cost $1/(n-i+1)$. The set S_0 has cost $1+\epsilon$ and covers all n elements.

The optimum solution is to take S_0: we pay $1+\epsilon$.

Greedy picks all sets $S_1, S_2, \dots, S_n$ at cost

$$\frac{1}{n} + \frac{1}{n-1} + \dots + \frac{1}{1} = H_n.$$

Problem 4

Let G be the input graph, and let T denote the DFS tree constructed by our algorithm. Let $L \subseteq V(T)$ be the set of leaves and I the set of internal nodes of T.

Claim 0.1 *The set I is a valid vertex cover of G.*

Proof. Consider any edge $e \in E(G)$. If $e \in E(T)$, then clearly it is adjacent to some internal node of T and hence is covered. Now consider any edge $e = (uv) \in E(G) - E(T)$. If one of its endpoints is an internal node, then e is covered. Thus, the only remaining option is that both u and v are leaves of T. We claim that if T is a DFS tree, there are no edges joining the leaves of T.

Suppose there was an edge $e = (uv)$, and u, v were leaves of T. Consider the order in which the DFS search visited vertices of G and let u was visited before v. Since u is a leaf, we know that the DFS search did not find any unvisited neighbors of u at the time u was being visited. But that is a contradiction, because v is a neighbor of u and was visited before u. ■

Claim 0.2 *For any valid vertex cover C of G, $|I| \leq 2|C|$.*

Proof. Note that any set of vertices C that is valid vertex cover of the graph G is also a valid vertex cover of T. Hence, without loss of generality we can assume that $G = T$.

Consider any (partial) matching M in T. Note that the size of M is a lower bound on the size of any vertex cover. By showing the existence of a matching of size at least $\lceil |I|/2 \rceil$, we prove the approximation guarantee of the algorithm.

For each vertex $u \in I$, let e_u be an edge from u to an arbitrary child of u in T. Note that the edges e_u are distinct, and hence there is exactly I of them.

We split the set $M' = \{e_u \mid u \in I\}$ into two sets M_A and M_B, such that each of M_A and M_B is a matching – one of them has to be of size at least $|I|/2$.

Since T is a tree, it is a bipartite graph. Let A and B be the two sides of the bipartition. We define $M_A = \{e_u \mid u \in I \cap A\}$ and $M_B = \{e_u \mid u \in I \cap B\}$. ■

Problem 5

Let n denote the number of items, w_i the weight and v_i the value of item i. We assume that the item are indexed in the order considered by the greedy algorithm, i.e. by decreasing ratio $r_i = v_i/w_i$. Let W denote the total size of the knapsack.

Let us fix an optimal solution $S^* = \{i_1, i_2, \dots, i_k\}$ and let $S = \{j_1, j_2, \dots, j_l\}$ be the solution produced by greedy. Let a be the first item in our ordering that belongs to the optimal solution but greedy did not take it: $a \in S^* - S$. Let $S_a = \{j \mid j \in S \wedge j < a\}$ be the set of items greedy picked before considering the item a. Similarly, let $S_a^* = \{j \mid j \in S^* \wedge j < a\}$ be the set of items in the optimum solution that come before a in our ordering.

Clearly, $S_a^* \subseteq S_a$, hence $\mathrm{val}(S_a^*) \leq \mathrm{val}(S_a)$. Since all items after a have value to weight ratio at most r_a, we have

$$v(S^*) = v(S_a^*) + v(S^* - S_a^*) \leq v(S_a) + r_a(W - w(S_a)).$$

Since the item a did not fit into the knapsack, we must have $w_a > W - w(S_a)$, and hence $w_a > r_a(W - w(S_a))$. That means that the value of the optimum solution is bounded above by the sum of two terms: $v(S_a)$ and v_a, and hence one of them must be at least $v(S^*)/2$. If $v(S_a) \geq v(S^*)/2$, then the greedy solution is a 2-approximation. If $v_a > v(S^*)/2$, then the solution consisting of a single item (namely, item a) has value half of the optimum, hence picking the item with highest value must be a 2-approxiamte solution in this case.