Approximation Algorithms

Momin and Rex

September 2025

Contents

This document provides, or attempts to provide, solutions to some of the problems from "Approximation Algorithms" by Vijay V. Vazirani. We aim to solve at least 4 questions from each section (unless a section contains fewer than 4 questions).

1 Introduction

1.1 Exercise 1.1

Give a factor 1/2 algorithm for the following.
(Acyclic subgraph) Given a directed graph $G = (V, E)$, pick a maximum cardinality set of edges from E so that the resulting subgraph is acyclic.

Algorithm

Give V an arbitrary (strict total) order. We can then write $V = \{v_1, ..., v_n\}$. We obtain the following two subsets of E:

$$E_f = \{(v_i, v_j) \mid i < j\}, \tag{1}$$

$$E_b = \{(v_i, v_j) \mid i > j\}. \tag{2}$$

Obviously, $G[E_f]$ and $G[E_b]$ are both acyclic. We answer with the larger of the two.

Time Complexity

$O(m)$.

Approximation Factor

Let $G_{OPT} = (V_{OPT}, E_{OPT})$ be the optimal subgraph of G. For each (v_i, v_j) in E_{OPT}, either $i < j$ or $i > j$. In other words, $E_{OPT} \subseteq E_f \cup E_b$, and thus $|E_{OPT}| \leq |E_f| + |E_b|$. This means

$$\max\{|E_f|, |E_b|\} \geq \frac{|E_{OPT}|}{2}. \tag{3}$$

1.2 Exercise 1.2

Design a factor 2 approximation algorithm for the problem of finding a minimum cardinality maximal matching in an undirected graph.

Algorithm

Greedily keep selecting legal edges until you can not.

Time Complexity

This is clearly $O(E)$.

Approximation Factor

Consider the graph G' formed by interpolating G_{OPT} and G_{ALG} (allowing parallel edges).
Note that for all $v \in V, deg(v) \leq 1$ in both graphs $\implies deg_{G'}(v) \leq 2$. Moreover, let D_1, D_2 be the vertices in G' with degree $1, 2$ respectively. Then

$$|D_1| \leq |ALG| \quad , \quad |D_2| \leq 2|OPT|$$

. Then, it follows that

$$\begin{aligned}
|ALG| &= \frac{1}{2}\sum_{v\in V} deg_{G_{ALG}}(v) \\
&= \frac{1}{2}\sum_{v\in V} deg_{G'}(v) - deg_{G_{OPT}}(v) \\
&\leq \frac{1}{2}(2|D_2| + |D_1|) - \frac{1}{2}\sum_{v\in V} deg_{G_{ALG}}(v) \\
&\leq 2|OPT|
\end{aligned}$$

which is what we wanted.

1.3 Exercise 1.3

(R. Bar-Yehuda) Consider the following factor 2 approximation algorithm for the cardinality vertex cover problem. Find a depth first search tree in the given graph, G, and output the set, say S, of all the nonleaf vertices of this tree. Show that S is indeed a vertex cover for G and $|S| \leq 2 \cdot OPT$.

Solution

We first show that G is indeed a vertex cover. Assume otherwise, then $\exists e = (u, v)$ s.t. $e \notin S \implies$ both u, v are leaves in the depth-first search tree. However since u and v are connected, either u is visited before v or vice versa $\implies$ either u or v is not a leaf - which is the required contradiction.
To show the approximation factor, we will show that G has a matching of size $|S|$ so that

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

by including both vertices of the matching since it is clearly a valid vertex cover. We form a matching by selecting an edge for each $v \in S$. Assume that this is not a matching, so $\exists e = (u, v)$ s.t. $e + M$ is a matching. However, we showed that either u is a non-leaf or v is. In either cases, either $deg(u) = 1$ or $deg(v) = 1$ - hence $e + M$ is not a matching.

1.4 Exercise 1.7

Let $A = \{a_1, ..., a_n\}$ be a finite set, and let $\leq$ be a relation on A that is reflexive, antisymmetric, and transitive. Such a relation is called a *partial ordering* of A. Two elements $a_i, a_j \in A$ are said to be *comparable* if $a_i \leq a_j$ or $aj \leq a_i$. Two elements that are not comparable are said to be *incomparable.* A subset $S \subseteq A$ is a *chain* if its elements are pairwise comparable. If the elements of S are pairwise incomparable, then it is an *antichain.* A chain (antichain) cover is a collection of chains (antichains) that are pairwise disjoint and cover A. The size of such a cover is the number of chains (antichains) in it. Prove the following min-max result: the size of a longest chain equals the size of a smallest antichain cover.

Solution

Let M be the size of a longest chain in A, and m be the size of a smallest antichain cover. Define $\phi(a)$ to be the size of the longest chain starting at a for some $a \in A$. Next, let $\Phi_k = \{a \mid \phi(a) = k\}$. It is clear that $\Phi_1, ..., \Phi_M$ form a partition of A. Note that in for $k \in \{1, ..., M\}$, Φ_k is an antichain. To see this, suppose $a, b \in \Phi_k$ are comparable, and let A and B be the chains implied by ϕ_a and ϕ_b respectively. Without loss of generality, suppose $a \leq b$. Then inserting a before B already produces a larger chain than A, which is a contradiction. As such, $M \geq m$.

Now, suppose there is a chain $(a_{c_1}, ..., a_{c_k})$ such that $k > m$. By the pigeonhole principle, there must be a_{c_i} and a_{c_j} in the same antichain, where $c_i < c_j$. However, by transitivity, $a_{c_i} \leq a_{c_j}$, so this is impossible. This proves $m \leq M$, and thus $m = M$.

1.5 Exercise 1.8

(Dilworth's theorem) Prove that in any finite partial order, the size of a largest antichain equals the size of a smallest chain cover.

Solution

Build the graph $G = (W, E)$ as follows:

1. Add u_i, v_i to V for every $i \in \{1, ..., n\}$.
2. Add $\{u_i, v_j\}$ to E for every $i, j \in \{1, ..., n\}$ if $u_i < v_j$.

We claim that

1. $n - |\text{smallest vertex cover}| \leq |\text{largest antichain}|$,
2. $|\text{largest antichain}| \leq |\text{smallest chain cover}|$, and
3. $|\text{smallest chain cover}| \leq n - |\text{maximum matching}|$.

If all are true, then the statement is true by König's Theorem. We first show (1). Let $U = \{u_i\}$ and $V = \{v_i\}$, C be a vertex cover of G, and $I = W \setminus C$ be the implied independent set. For any set X, define $X_U = \{i \mid u_i \in X \cap U\}$, $X_V = \{i \mid v_i \in X \cap V\}$. Consider $M = I_U \cap I_V$. For any distinct $i, j \in M$, by construction, both $\{u_i, v_j\}$ and $\{u_j, v_i\}$ are not in E, which means a_i and a_j are not comparable. As such, M is an antichain. Consider that

$$\begin{aligned}
|\text{largest antichain}| \geq |M| &= |I_U \cap I_V| && (4)\\
&= |\{1, ..., n\} \setminus (C_U \cup C_V)| && (5)\\
&\geq n - |C|. && (6)
\end{aligned}$$

(2) can be easily shown using the pigeonhole principle.

For (3), given any matching M, we build a corresponding chain cover. Let $G = (A, R)$ be a directed graph, where $(a_i, a_j) \in R$ if and only if $\{u_i, v_j\} \in M$. Define $\sim$ to be a relation on A such that $a_i \sim a_j$ if and only if a_j is reachable from a_i (denoted $a_i \to a_j$), or the reverse. We verify that it is an equivalence relation: reflexivity and symmetry follow directly from definition. For transitivity, suppose $a_i \sim a_j$ and $a_j \sim a_k$. The case is trivial when $a_i \to a_j \to a_k$. If $a_i \to a_j$ and $a_k \to a_j$, suppose the respective paths are $(a_i = u_1, ..., a_j = u_p)$ and $(a_k = v_1, ..., a_j = u_q)$. If $a_i \neq a_k$, there must be a smallest k such that $u_{p-k} \neq v_{q-k}$. However, this would imply that both $\{u_{p-k}, u_{p-k+1}\}$ and $\{v_{q-k}, v_{q-k+1}\}$ are in M. By construction, $u_{p-k+1} = v_{q-k+1}$, leading to a contradiction. The case is similar when $a_j \to a_i$ and $a_i \to a_k$. Observe that $A/\sim$ is exactly a chain cover of A, and every edge in M decreases the number of equivalence classes by 1. As such,

$$|\text{smallest chain cover}| \leq |A/\sim| = n - |M|. \qquad (7)$$

2 Set Cover

2.1 Exercise 2.1

Given an undirected graph $G = (V, E)$, the cardinality maximum cut problem asks for a partition of V into sets S and $\overline{S}$ so that the number of edges running between these sets is maximized. Consider the following greedy algorithm for this problem. Here v_1 and v_2 are arbitrary vertices in G, and for $A \subset V, d(v, A)$ denotes the number of edges running between vertex v and set A.

Algorithm 1: $\frac{1}{2}$ approximation algorithm for cardinality maximum cut problem

```
Data: Vertices $V$, initial vertices $v_1$, $v_2$
Result: Initialized sets $A$ and $B$
Initialize $A \leftarrow \{v_1\}$, $B \leftarrow \{v_2\}$;
for $v \in V \setminus \{v_1, v_2\}$ do
    if $d(v, A) \geq d(v, B)$ then
        $B \leftarrow B \cup \{v\}$;
    end
    else
        $A \leftarrow A \cup \{v\}$;
    end
end
return $A, B$
```

Show that this is a factor $\frac{1}{2}$ approximation algorithm and give a tight example. What is the upper bound on OPT that you are using? Give examples of graphs for which this upper bound is as bad as twice OPT. Generalize the problem and the algorithm to weighted graphs.

Solution

We first show that this achieves a $\frac{1}{2}$- factor approximation.
Let $v_3 \dots v_n$ be the order in which the vertices are assigned in the partition and consider the first v_k where it differs from OPT and consider the graph induced by $\{v_1, \dots, v_k\}$. WLOG assume $v_k \in A$ in ALG but $v_k \in B$ in OPT. Then by construction, it is clear that switching it can at most double the weight (proved in exercise 2.2).
By induction on n, we have that

$$|OPT| \leq 2 \cdot |OPT|$$

which is what we wanted.
To generalize this, we just define

$$d(v, A) = \sum_{e=v,u \text{s.t. } u \in A} w(e)$$

, and a similar argument follows.

2.2 Exercise 2.2

Consider the following algorithm for the maximum cut problem, based on the technique of local search. Given a partition of V into sets, the basic step of the algorithm, called flip, is that of moving a vertex from one side of the partition to the other. The following algorithm finds a locally optimal solution under the flip operation, i.e., a solution which cannot be improved by a single flip. The algorithm starts with an arbitrary partition of V . While there is a vertex such that flipping it increases the size of the cut, the algorithm flips such a vertex. (Observe that a vertex qualifies for a flip if it has more neigh- bors in its own partition than in the other side.) The algorithm terminates when no vertex qualifies for a flip. Show that this algorithm terminates in polynomial time, and achieves an approximation guarantee of $\frac{1}{2}$.

Solution

Running time: First note that

$$OPT \leq E$$

. At each iteration,

$$ALG_{i+1} \geq ALG_i + 1$$

, since we only switch if we get an increase. Each iteration takes $O(V)$ time and we have $O(E)$ iterations $\implies$ running time is $O(VE)$.
Analysis: We claim that

$$OPT \leq 2 \cdot ALG$$

. Let $A_{ALG} = \{v_1 \dots v_k\}$ and similarly $A_{OPT} = \{u_1 \dots u'_k\}$. Note that we can achieve the optimum configuration by doing the flips. However, for each $v_i \in A_{ALG}$ we have

$$d(v, A) \leq d(v, B) = E(v)$$

where $d(v, A)$ represents the neighbors of v in A and $E(v)$ represents the cut edges from v.
Therefore, at flip, we can at most have

$$E(v) = d(v, A) + d(v, B) \leq 2d(v, B)$$

. Therefore, we have

$$ALG = \sum_{v \in A_{ALG}} E(v) \leq \frac{1}{2} \sum_{v \in A_{OPT}} E(v) = \frac{1}{2} \cdot OPT$$

2.3 Exercise 2.13

Use layering to get a factor f approximation algorithm for set cover, where f is the frequency of the most frequent element. Provide a tight example for this algorithm.

Solution

Let the universal set be $U = \{x_1, ..., x_n\}$, and the covers be $\mathcal{C} = C_1, ..., C_k \subseteq U$. We will assume that $U = C_1 \cup ... \cup C_k$. Define $C(x) = \{C_i \mid x \in C_i\}$ and $f(x) = |C(x)|$, i.e. the covers and frequency of any $x \in U$. Clearly, $f = \max_{x \in U} f(x)$. Let $c : \mathcal{C} \to \mathbb{Q}_{\geq 0}$ be the cost function. We say c is *size-weighted* if for any $C \in \mathcal{C}$, $c(C) = k \cdot |C|$ for some fixed $k \in \mathbb{Q}_{\geq 0}$.

Lemma 1. *If c is size-weighted, any solution is an f-approximation.*

Proof. For any solution $S \subseteq \mathcal{C}$,

$$c(S) = k \sum_{C \in S} |C| \leq k \sum_{C \in \mathcal{C}} |C| = k \sum_{x \in U} f(x) \leq kf \cdot |U| \leq f \cdot OPT. \tag{8}$$

$\square$

Our algorithm is as follows:

Algorithm 2: An f-approximation algorithm for set cover.

```
$W, Z \leftarrow \varnothing$;
while $U \neq \varnothing$ do
    $i \leftarrow$ iteration count (starting at 0);
    $\mathcal{C}_i \leftarrow \mathcal{C}$;
    $U_i \leftarrow U$;
    $Z_i \leftarrow \{j \mid C_j = \varnothing \in C\}$;
    $Z \leftarrow Z \cup Z_i$;
    $\mathcal{C} \leftarrow \mathcal{C} \setminus \{C_j \mid j \in Z_i\}$;
    $c_i \leftarrow (C \mapsto \min_{C_j \in \mathcal{C}}\{c(C_j)/|C_j|\} \cdot |C|)$;
    $c \leftarrow c - c_i$;
    $W_i \leftarrow \{j \mid c(C_j) = 0, C_j \in C\}$;
    $W \leftarrow W \cup W_i$;
    $\mathcal{C} \leftarrow \mathcal{C} \setminus \{C_j \mid j \in W_i\}$;
    $U \leftarrow U \setminus \{x \in C_j \mid j \in Z_i \cup W_i\}$;
end
return $W$;
```

The algorithm is clearly polynomial. We first prove its correctness. Suppose there exists $x \in U$ that is not covered by W. Let $C(x) = \{C_{k_1}, ..., C_{k_{f(x)}}\}$, in the order in which the covers are removed from C. By construction, $C(x) \subseteq Z$. However, when C_{k_1} is removed in the Z_i step, u is still not removed from U. This means that $u \in C_{k_1}$ and $C_{k_1} \neq \varnothing$, which is a contradiction.

To show the approximation factor, observe that for $C \in W$, if $C \in W_i$,

$$c(C) = \sum_{j=0}^{i} c_j(C). \tag{9}$$

For $C \in \mathcal{C} \setminus W = Z$, because it is removed before its cost is completely decomposed, if $C \in Z_i$,

$$c(C) \geq \sum_{j=0}^{i} c_j(C). \tag{10}$$

Suppose the algorithm ends after the k^{th} iteration. Let W^* be the optimal set cover. Note that $W^* \cap \mathcal{C}_j$ for U_j for every $j \in \{0, ..., k\}$. Therefore,

$$\begin{aligned}
c(W) &= \sum_{j=0}^{k} c_j(W_j) && (11)\\
&\leq \sum_{j=0}^{k} c_j(\mathcal{C}_j) && (12)\\
&\leq f \cdot \sum_{j=0}^{k} OPT(U_j, \mathcal{C}_j, c_j) \text{ (Lemma 1)} && (13)\\
&\leq f \cdot \sum_{j=0}^{k} c_j(W^* \cap \mathcal{C}_j) && (14)\\
&\leq f \cdot c(W^*). && (15)
\end{aligned}$$

For a tight example, consider $K^f_{n,...,n}$, i.e. the complete f-partite f-hypergraph with n vertices in each partition. Its corresponding instance in set cover is $U = \{x_{1,1}, ..., x_{1,n}, ..., x_{f,1}, x_{f,n}\}$, $\mathcal{C} = \{\{x_{1,i_1}, ..., x_{f,i_f}\} \in U\}$ and the unit cost function. Since the cost function is size-weighted, our algorithm chooses the entirety of U, but the optimal solution is obviously any one partition.

2.4 Exercise 2.14

A *tournament* is a directed graph $G = (V, E)$, such that for each pair of vertices, $u, v \in V$, exactly one of (u, v) and (v, u) is in E. A *feedback vertex set* for G is a subset of the vertices of G whose removal leaves an acyclic graph. Give a factor 3 algorithm for the problem of finding a minimum feedback vertex set in a directed graph.

Solution

We start with the following key observation:

Lemma 2. *Every cycle in a tournament contains a 3-cycle.*

Proof. Consider the cycle $C = (v_1, ..., v_k, v_1)$. Since G is not a multigraph, $k \geq 3$. When $k = 3$, the lemma is trivial. Otherwise, there are two cases: first, if $(v_2, v_k) \in E$, then (v_1, v_2, v_k) is already our desired 3-cycle. If (v_k, v_2) is in E, however, then we can build a $k-1$-cycle $(v_2, ..., v_k, v_2)$. Therefore, by induction, we can always obtain a 3-cycle. □

Therefore, for every instance of feedback vertex set, we can derive an equivalent instance of set cover:

$$U = \{\{v_1, v_2, v_3\} \mid (v_1, v_2), (v_2, v_3), (v_3, v_1) \in E\}, \tag{16}$$
$$\mathcal{C} = \{\{x \in U \mid v \in x\} \mid v \in V\}, \tag{17}$$
$$c = C \mapsto w(v). \tag{18}$$

More explicitly, U is all 3-cycles in G, and each vertex v in G translates to a cover that includes all 3-cycles v is in. It is easy to see with Lemma 2 that the optimal costs of the two instances are identical. Clearly, each 3-cycle appears in exactly three covers. Consequently, we can use our algorithm in Exercise 2.13 to give a 3-approximation.

3 Steiner Tree and TSP

3.1 Exercise 3.1

The hardness of the Steiner tree problem lies in determining the optimal subset of Steiner vertices that need to be included in the tree. Show this by proving that if this set is provided, then the optimal Steiner tree can be computed in polynomial time.

Solution

We simply find MST of the graph induced by $R \cup O$, where R is the set of required vertices and O is the given set of optimal vertices.
Assume the optimal solution is such that

$$OPT < ALG$$

, then note that both ALG and OPT are spanning trees on the same vertices. This is a contradiction since ALG is the minimum spanning tree.

3.2 Exercise 3.3

Give an approximation factor preserving reduction from the set cover problem to the following problem, thereby showing that it is unlikely to have a better approximation guarantee than $O(logn)$.
Problem 3.14 (Directed Steiner tree) G = (V,E) is a directed graph with nonnegative edge costs. The vertex set V is partitioned into two sets, required and Steiner. One of the required vertices, r, is special. The problem is to find a minimum cost tree in G rooted into r that contains all the required vertices and any subset of the Steiner vertices.

Solution

Consider an instance of the set cover problems (U, S). We will transform it into an instance of the directed Stiener tree problem to show that it is just as hard.

Consider the graph with $V = r \cup U \cup S$. For each $s \in S$. For each $s \in S$,

$$\exists e = (s, u) \forall u \in U, w(e) = 0 \quad , \quad \exists e = (s, r), w(e) = w(s)$$

, and there are no other edges. The set of required vertices is simply the set U. Consider the optimal solution for the instance of the set cover OPT_S. Then, we claim that the following solution (with the same cost) is optimal for the corresponding directed Stiener tree instance OPT_T :

$$OPT_S = \{\cup_{s \in OPT_S}((r, s) \cup (\cup_{v \in s}(s, v))\}$$

Assume otherwise, then selecting the corresponding $s \in OPT_S$ leads to a solution with a better cost for the set cover instance - which contradicts the optimality OPT_S.
Therefore, if we can solve directed Steiner tree problem with factor c, we can solve the set cover problem with factor c as well by the given reduction $\implies$ it is unlikely to have a better guarantee than $O(\log n)$

3.3 Exercise 3.5

(Papadimitriou and Yannakakis) Let G be a complete undirected graph in which all edge lengths are either 1 or 2 (clearly, G satisfies the triangle inequality). Give a 4/3 factor algorithm for TSP in this special class of graphs.

Solution

We start by finding an optimal cycle cover of G. In other words, we would like to find a set $\mathcal{C} = \{C_1, ..., C_k\}$ of disjoint cycles such that $\cup \mathcal{C} = V$, at minimum $c(C)$. This is known to be solvable in polynomial time. Next, define C as the subgraph that encapsulates G without its edges. For every cycle C_i in $\mathcal{C}$, define an arbitrary order for its vertices such that $C_i = (c_{i,1}, ..., c_{i,|C_i|}, c_{i,1})$. Add the edges $(c_{i,1}, c_{i,2}), ..., (c_{i,|C_i-1|}, c_{i,|C_i|}), (c_{i,|C_i|}, c_{j,1})$ to C, where $j = (i + 1) \bmod k$. As the cycles in $\mathcal{C}$ are disjoint, this produces a Hamiltonian cycle. Observe that as there are no parallel edges in G, each cycle is at least of size 3. In other words, $k \leq n/3$ and

$$\begin{aligned} c(C) &= \sum_{i=0}^{k} c(C_i) - c((c_{i,|C_i|}, c_{i,1})) + c((c_{i,|C_i|}, c_{j,1})) && (19) \\ &\leq \sum_{i=0}^{k} c(C_i) + 1 && (20) \\ &\leq c(C_i) + n/3. && (21) \end{aligned}$$

As the optimal solution C^* is also a cycle cover, and the minimum cost of each edge is 1,

$$OPT = c(C^*) \geq c(C_i) \geq n. \tag{22}$$

Combining the two gives $c(C) \leq 4/3 \cdot c(C^*)$, giving a 4/3 approximation factor for our algorithm.

3.4 Exercise 3.6

(Frieze, Galbiati, and Maffioli) Give an $O(\log n)$ factor approximation algorithm for the following problem.

Problem 1. *(Asymmetric TSP) We are given a directed graph G on vertex set V, with a nonnegative cost specified for edge $(u \to v)$, for each pair $(u, v) \in V$. The edge costs satisfy the directed triangle inequality, i.e., for any three vertices u, v, and w, $c(u \to v) \leq c(u \to w) + c(w \to v)$. The problem is to find a minimum cost cycle visiting every vertex exactly once.*

Solution

We describe our algorithm below:

Algorithm 3: An $O(\log n)$-approximation algorithm for asymmetric TSP.

```
function main($G$, $c$):
    if $|G| = 1$ then
        return $G$;
    else
        $\mathcal{C} \leftarrow$ minimum-cost cycle cover of $G$;
        $G' \leftarrow G$;
        in $G'$, contract every cycle in $\mathcal{C}$ into a vertex;
        $u : \mathcal{C} \to V(G)$;
        for $C_i \in \mathcal{C}$ do
            /* we pick an arbitrary vertex in every cycle as the "start" */
            $u(C_i) \leftarrow$ any $v \in C_i$;
        end
        $c' : E(G') \to \mathbb{Q}_{\geq 0}$;
        for $C_i, C_j \in \mathcal{C}, C_i \neq C_j$ do
            $c'((C_i, C_j)) \leftarrow c(u(C_i), u(C_j))$;
        end
        $C' \leftarrow$ main($G'$,$c'$);
        $C \leftarrow$ empty path;
        for $C_i \in C'$ do
            append $u(C_i), ..., \text{pred}(u(C_i))$ to $C$;
        end
        close $C$ as a cycle;
        return $C$;
return main($G$, $c$);
```

The algorithm obvious always produces an Hamiltonian cycle. To show the approximation factor, observe that the cost of C can be divided into two parts:

1. ones within the cycle, plus $c((\text{pred}(u(C_i)), u(C_i)))$ for each $C_i \in C'$, and

2. the "connecting edges" between cycles in the cover, minus the extra term in (1).

Denote their costs by c_1 and c_2 respectively. Let $C^* = (c_1, ..., c_n, c_1)$ be the optimal Hamiltonian cycle. Note that C^* is itself a cycle cover, so $c_1 \leq OPT = c(C^*)$. Next, let $(u_1, ..., u_k)$ be the sorted version of u in the order of appearances in C^*. By the triangle inequality,

$$c_2 = \sum_{(i,j)\in\{(1,2),...,(k-1,k),(k,1)\}} c((\text{pred}(u(C_i)), u(C_j))) - c((\text{pred}(u(C_i)), u(C_i))) \tag{23}$$

$$\leq \sum c((\text{pred}(u(C_i)), u(C_i), u(C_j))) - c((\text{pred}(u(C_i)), u(C_i))) \tag{24}$$

$$= \sum c((u(C_i), u(C_j))). \tag{25}$$

Now let $E(n)$ be the maximum error of the algorithm on graphs of n vertices. By the triangle inequality, we know that $c((u_1, ..., u_k, u_1)) \leq OPT$. Also note that $(u_1, ..., u_k, u_1)$ corresponds to an Hamiltonian cycle in G'. As such,

$$E(n) = c_1 + c_2 - OPT \leq c_2 \leq OPT + E(|V(G')|). \tag{26}$$

Each cycle has a size of at least 2, so we have $E(n) = OPT \cdot \log n$. In other words, this algorithm has an approximation factor of $O(\log n)$.

4 Multiway Cut and k-Cut

4.1 Exercise 4.1

Show that Algorithm 4.3 can be used as a subroutine for finding a k-cut within a factor of $2-2/k$ of the minimum k-cut. How many subroutine calls are needed?

Solution

The algorithm is simply to try all k combinations of the n vertices as the terminals and take the cost-minimal cut. As the optimal k-cut must be one of them, the approximation is $2 - 2/k$. The number of subroutine calls is

$$C_k^n = \frac{n!}{k!(n-k)!} \in O\left(\frac{n^k}{k!}\right). \tag{27}$$

4.2 Exercise 4.3

Let $G = (V, E)$ be a graph and $w : E \to \mathbb{R}^+$ be an assignment of nonnegative weights to its edges. For $u, v \in V$ let $f(u, v)$ denote the weight of a minimum $u - v$ cut in G.

1. Let $u, v, w \in V$, and suppose $f(u, v) \leq f(u, w) \leq f(v, w)$. Show that $f(u, v) = f(u, w)$, i.e., the two smaller numbers are equal.

2. Show that among the $\binom{n}{2}$ values $f(u,v)$, for all pairs $u, v \in V$, there are at most $n-1$ distinct values.

3. Show that for $u, v, w \in V$,
$$f(u,v) \geq \min\{f(u,w), f(w,v)\}$$
.

4. Show that for $u, v, w_1, \ldots, w_r \in V$
$$f(u,v) \geq \min\{f(u,w_1), f(w_1,w_2), \ldots, f(w_r,v)\}$$

.

Solution

Part 3: Consider any 3 vertices x, y, z. Then,
$$f(x,y) \geq \min(f(x,z), f(z,y))$$
since any $x-y$ must cut z from x or from y.
Part 4: Extending the same idea, consider the path $u - w_1 - \ldots - w_r - v$. Then any u, v cut either separates $u - w_1$ or $w_1 - w_2$ or $\ldots$ or $w_r - v$ result in
$$f(u,v) \geq \min\{f(u,w_1), f(w_1,w_2), \ldots, f(w_r,v)\}$$
which is what we wanted to show.
Part 1 : Assume we have
$$f(u,v) \leq f(u,w) \leq f(v,w)$$
. Then from above, we have
$$f(u,v) \geq \min(f(u,w), f(v,w)) = f(u,w)$$
$\implies f(u,v) = f(v,w).\square$
Part 2: Construct a graph on V such that we greedily add an edge (u,v) of weight $f(u,v)$ if $f(u,v)$ is distinct so far. Then, it is clear that there are no parallel edges.
Assume this graph contains a cycle $x_1 \ldots x_n x_1$ and let $f(x_k, x_{k+1})$ be the smallest one. Then from part 4, we see that
$$f(x_k, x_{k+1}) \geq \min(f(x_k, x_{k-1}), \ldots, f(x_1, x_n), f(x_n, x_{n-1}), \ldots f(x_{k+2}, f_{k+1}))$$
. However, this contradicts the minimality of $f(x_k, x_{k+1})$ since all other values are distinct by construction.
Remark: I feel that the order of questions indicate that there is a different intended proof for part 2 that does not use part 4 - I'm unsure how to do that.

4.3 Exercise 4.4

Let T be a tree on vertex set V with weight function w' on its edges. We will say that T is a flow equivalent tree if it satisfies the first of the two Gomory–Hu conditions. i.e., for each pair of vertices $u, v \in V$, the weight of a minimum u–v cut in G is the same as that in T. Let K be the complete graph on V. Define the weight of each edge $(u, v) \in K$ to be $f(u, v)$. Show that any maximum weight spanning tree in K is a flow equivalent tree for G.

Solution

Let T be a max weight spanning tree and consider any (u, v). If $e = (u, v) \in T$, then $w'(u, v)$ is the min cut by construction.
If $e = (u, v) \notin T$, since T is a tree, the min $u - v$ cut in T is just an edge (x_i, x_{i+1}). In particular, the min cut is the min weight edge in the unique simple path from u to v - denoted by $ux_1 \dots x_k v$. From part 4 above, we have

$$f(u, v) \geq \min(f(u, x_1), \dots f(x_k, v)) = w'(x_i, x_{x+1})$$

. If $f(u, v) \leq w'(x_i, x_{i+1})$, then $f(u, v) = w'(x_i, x_{i+1})$ and we are done.
Otherwise, assume $f(u, v) > w'(x_i, x_{i+1})$ - then simply removing edge (x_i, x_{i+1}) and adding edge (u, v) (with weight $w'(u, v)$) results in a spanning tree with weight function w' and higher weight - this contradicts the maximality of T. $\square$

4.4 Exercise 4.7

Prove that if the Gomory-Hu tree for an edge-weighted undirected graph G contains all $n - 1$ distinct weights, then G can have only one minimum weight cut.

Solution

Let $w^*(u, v)$ represent the weight of the minimum cut separating $u, v \in V(G)$. Then the minimum cut of G has weight

$$w = \min_{u,v \in V(G)} w^*(u, v). \tag{28}$$

Let T be the Gomory-Hu tree of G. We know that

$$w = \min_{u,v \in V(G)} w''(u, v) = \min_{\{u,v\} \in E(G)} w'(u, v), \tag{29}$$

where w'' denotes the minimum weight of edges on the (unique) path from u to v in T. Let $c \in E(T)$ be the (unique) minimum-weight edge of T, and C be the corresponding cut of G. It is clear that $w'(c) = w$. Next, suppose there is a minimum cut $C' = (A, B)$ of G that is not C. In other words, $\delta(A) = \delta(B) \neq \{c\}$. We can then pick adjacent $a, b \in E(T)$ such that $a \in A, b \in B$ and $\{a, b\} \neq c$. As C' separates a and b, $C' \geq w'(a, b) > w$, leading to a contradiction.

5 k-Center

5.1 Exercise 5.1

Show that if the edge costs do not satisfy the triangle inequality, then the k-center problem cannot be approximated (by a polynomial algorithm) within factor $\alpha(n)$ for any polynomial-time computable function $\alpha(n)$, assuming $P \neq NP$.

Solution

We show that dominating set can be reduced to $\alpha(n)$-approximate k-center in polynomial time. Let $\mathcal{I} = (G = (V, E), k)$ be an instance of dominating set. Let $G' = K_{|V|}$. We define the weight function w on $E(G')$ as follows:

$$w((u,v)) = \begin{cases} 1 & \text{if } (u,v) \in E \\ \alpha(n) + \varepsilon & \text{if } (u,v) \notin E \end{cases}. \tag{30}$$

Suppose $\mathcal{I}$ is a yes-instance. Then there exists a collection of vertices $D \subseteq V$ of size at most k such that for every $v \in V \setminus D$, there exists an edge between v and D in G. Next, take D as the centers, add arbitrary vertices to it to make up to size k, and call the resulting collection C. For $v \in V \setminus C'$, we have

$$d_{G'}(v, C) \leq d_{G'}(v, D) = 1. \tag{31}$$

Otherwise, if $\mathcal{I}$ is a no-instance, for any $C \subseteq V$ of size within k, there exists $v \in V$ such that there is no edge between v and C. Then, by construction,

$$d_{G'}(v, C) = \alpha(n) + \varepsilon > \alpha(n). \tag{32}$$

Therefore, if $\mathcal{I}' = (G, w, k)$ of k-center returns a distance within $\alpha(n)$, we can always deduce that $\mathcal{I}'$ is a yes-instance, and vice versa.

5.2 Exercise 5.2

Consider Step 2 of Algorithm 5.3, in which a maximal independent set is found in G_i^2. Perhaps a more natural choice would have been to find a minimal dominating set. Modify Algorithm 5.3 so that M_i is picked to be a minimal dominating set in G_i^2. Show that this modified algorithm does not achieve an approximation guarantee of 2 for the k-center problem. What approximation factor can you establish for this algorithm?

Solution

Note that the 2-factor relied on the lower bound

$$|I| \leq dom(H)$$

, where I is an independent set in H^2. This clearly does not hold if we instead choose the minimal dominating set.
Instead we will show (with a tight example) that the algorithm achieves a 3 factor.

3-Factor Approximation

Observe that the returned set M_j is a dominating set in G_j^2, which implies that every vertex is at most distance $2r_j$ from some center in M_j, where r_j is the value of r at step j. Thus, the radius ρ of the solution satisfies $\rho \leq 2r_j$.
It remains to show that $r_j \leq 1.5 \cdot \mathrm{OPT}$. Consider $r = 1.5 \cdot \mathrm{OPT}$. At this value, $G^2 = U_{3\cdot\mathrm{OPT}}$, where U_s denotes the graph with edges between vertices at distance at most s.
The optimal solution partitions the vertices into k clusters

$$C_1, \ldots, C_k$$

, each of diameter at most $2 \cdot \mathrm{OPT}$. In $U_{3\cdot\mathrm{OPT}}$, each cluster C_i induces a clique, since all distances within C_i are at most $2 \cdot \mathrm{OPT} < 3 \cdot \mathrm{OPT}$.
Let L be the graph formed by the disjoint union of these k cliques (removing any inter-cluster edges present in $U_{3\cdot\mathrm{OPT}}$). In L, every minimal dominating set has size exactly k (one vertex per clique). Thus, the upper domination number $\Gamma(L) = k$.
Adding the inter-cluster edges back to obtain $U_{3\cdot\mathrm{OPT}}$ does not increase the upper domination number, as additional edges make domination easier and can only render larger sets non-minimal. Therefore,

$$\Gamma(U_{3\cdot\mathrm{OPT}}) \leq k$$

. This means every minimal dominating set in $U_{3\cdot\mathrm{OPT}}$ has size at most k. Regardless of which minimal dominating set the algorithm picks as M_i, $|M_i| \leq k$. Thus, the algorithm stops no later than $r = 1.5 \cdot \mathrm{OPT}$, so

$$r_j \leq 1.5 \cdot \mathrm{OPT} \quad , \quad \rho \leq 3 \cdot \mathrm{OPT}$$

.

Tight Example

The distances are given by $a - b = OPT, b - c = OPT, c - d = OPT, a - d = 2OPT, b - d = 2OPT$, and each additional vertex has distance $3OPT$ to d, with all other distances being the shortest path distances to ensure the metric property.
In this instance, the optimal solution is to pick a set of k centers from the main chain a-b-c-d, with radius OPT (adjusted for $k > 1$ by replicating the structure).

The modified algorithm returns a solution with radius $3OPT$ because for thresholds $r < 1.5OPT$, the graph U_{2r} has the additional vertices isolated (distances $> 2r$), so all minimal dominating sets must include all the additional vertices (size $n > k$), forcing $|M_i| > k$ no matter which one is picked. It continues until $r = 1.5OPT$, where the edges to the additional vertices are included, allowing smaller dominating sets, and the returned M_j has radius $3OPT$.

5.3 Exercise 5.3

Let $G = (V, E)$ be a complete undirected graph with edge costs satisfying the triangle inequality, and let k be a positive integer. The problem is to partition V into sets $V_1, ..., V_k$ so as to minimize the costliest edge between two vertices in the same set, i.e., minimize

$$\max_{1\le i\le k,\ u,v\in V_i} c(u, v). \tag{33}$$

(1) Give a factor 2 approximation algorithm for this problem, together with a tight example.

(2) Show that this problem cannot be approximated within a factor of $2 - \varepsilon$, for any $\varepsilon > 0$, unless $P = NP$.

Solution for (1)

Consider the following algorithm:

Algorithm 4: A 2-approximation algorithm for k-cluster.

```
/* list of heads */
$H \leftarrow \varnothing$;
/* distance to the nearest head */
$D \leftarrow$ an array of size $|V|$, with each element initialized to $+\infty$;
for $i \in \{1, ..., k\}$ do
    $j_i \leftarrow$ index of the (first) max element in $D$;
    $H$.append($V[j_i]$);
    for $k \in \{1, ..., |V|\}$ do
        $D[k] \leftarrow d(V[k], H)$;
    end
end
return vertices clustered by closest distance to heads in $H$;
```

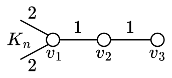
Let H' be the final H together with the farthest vertex u from H. As $G[H']$ is a $(k + 1)$-clique, we get

$$OPT \ge h = d(u, H'). \tag{34}$$

Let C be a cluster given by the algorithm above and h be its head. For any two vertices $u, v \in C$, by triangle inequality, we have

$$d(u, v) \le d(u, h) + d(h, v) \le 2h \le 2 \cdot OPT. \tag{35}$$

For a tight example for any given n and $k = 2$, consider metric closure of the following graph:



Suppose v_1 is selected as the first head in the first iteration. Note that every vertex is of the same distance to v_1. As such, it is possible that some other vertex u in K_n is chosen as the second head. In this case, a possible output by the algorithm is $\{u\}$ and $(K_n \setminus u) \cup \{v_2, v_3\}$ as the two clusters, with a maximum inter-cluster distance of 4. However, the optimal partition is K_n and $\{v_2, v_3\}$, with a maximum distance of 2.

Solution for (2)

Suppose there exists a $(2 - \varepsilon)$-approximation for metric k-cluster. We reduce clique cover to it. Consider an instance $\mathcal{I} = (G = (V, E), k)$ of (decision) clique cover. We assign costs c to the edges as follows:

$$c((u,v)) = \begin{cases} 1 & \text{if } \{u,v\} \in E \\ 2 & \text{if } \{u,v\} \notin E. \end{cases} \tag{36}$$

It is easy to show that c is a metric. We claim that $\mathcal{I}$ is a yes-instance if and only if the optimal k-cluster of (G, c) has a maximum distance of 1. For the forward direction, simply taking the cliques as the clusters gives the optimal solution. Conversely, if the maximum distance within a cluster C is 1, for every pair of vertices $u, v \in V(C)$, $(u, v) \in E$. In other words, C is a clique. This verifies the reduction.

5.4 Exercise 5.4

(Khuller, Pless, and Sussmann [169]) The fault-tolerant version of the metric k-center problem has an additional input, $\alpha \le k$, which specifies the number of centers that each city should be connected to. The problem again is to pick k centers so that the length of the longest edge used is minimized.
A set $S \subset V$ in an undirected graph $H = (V, E)$ is an α-dominating set if each vertex $v \in V$ is adjacent to at least α vertices in S (assuming that a vertex is adjacent to itself). Let $dom_\alpha(H)$ denote the size of a minimum cardinality α-dominating set in H.

1. Let I be an independent set in H^2. Show that $\alpha|I| \le dom_\alpha(H)$.
2. Give a factor 3 approximation algorithm for the fault-tolerant k-center problem.

Solution

Part (a)

Let D be a minimum $\alpha-$dominating set in H. For each $v \in I$, since D is $\alpha-$dominating,

$$|D \cap N_H[v]| \geq \alpha$$

. If two distinct vertices $u, v \in I$ shared a common neighbor $w \in D$, then u, v would be distance at most $2 \in H$, contradicting the independence of I in H^2. Therefore, the sets $D \cap N_H[v]$ are disjoint. It follows that

$$|D| \geq \alpha|I| \implies \alpha|I| \leq dom_\alpha(H)$$

.

Part (b)

Algorithm 5: 3 approximation algorithm for fault tolerant k-center problem

```
Data: $G = (V, E)$
Result: fault tolerant $k - center\ S \subseteq V$
Sort distances $r_1 < r_2 < \ldots < r_m$ between pairs of cities. ;
for $i = 1$ to $m$ do
    Compute $G_i^2$ ;
    Compute $M_i \in G_i^2$ ;
    if $M_i \leq \lceil \frac{k}{\alpha} \rceil$ then
        if $\forall v \in M_i, deg(v) \leq \alpha - 1 \in G_i$ then
            return $(M_i)$
        end
    end
end
```

From part(a),

$$I \in G(r^*)^2 \implies \alpha|I| \leq dom_\alpha(G(r^*)) \leq k \implies |I| \leq \lceil \frac{k}{\alpha} \rceil$$

, where I is an independent set. Thus for any the optimal radius r^*, any maximal independent set in $G(r^*)^2$ has size at most $\lceil \frac{k}{\alpha} \rceil$.
The algorithm chooses the smallest j s.t. $|M_j| \leq \lceil \frac{k}{\alpha} \rceil$ and each vertex in M_j has degree at most $\alpha - 1$. This ensures $r_j \leq 3r^*$.
Since M_j is a maximal independent set in G_j^2, every city is within distance $2r_j$ of some vertex in M_j. Including all closed neighborhoods of the vertices in M_j ensures that every city within distance $3r_j$ of at least α centers.

6 Feedback Vertex Set

6.1 Exercise 6.1

A natural greedy algorithm for finding a minimum feedback vertex set is to repeatedly pick and remove the most cost-effective vertex, i.e., a vertex minimizing $w(v)/\delta_H(v)$, where H is the current graph, until there are no more cycles left. Give examples to show that this is not a constant factor algorithm. What is the approximation guarantee of this algorithm?

Solution

This is similar to the greedy set cover algorithm. Let G_i be the remaining graph after i removals, with $G_0 = G$ being the original graph. Denote by F^* the optimal feedback vertex set. For any i, as $H_i = F^* \cap V(G_i)$ remains a feedback vertex set for G_i, if we represent by v_i the vertex chosen by the algorithm at the $(i+1)^{\text{th}}$ iteration, we have

$$\frac{w(v_i)}{\delta_{G_i}(v_i)} \leq \min_{v \in H_i} \frac{w(v)}{\delta_{G_i}(v)} \leq \frac{w(H_i)}{\delta_{G_i}(H_i)} \leq \frac{OPT}{\text{cyc}(G_i)}. \tag{37}$$

Suppose the algorithm stops after k iterations. Let $F = \{v_1, ..., v_k\}$. Then

$$\begin{aligned}
w(F) &\leq \sum_{i \in \{0,...,k-1\}} \left(OPT \cdot \frac{\delta_{G_i}(v_i)}{\text{cyc}(G_i)} \right) && (38)\\
&= OPT \cdot \left(\underbrace{\frac{1}{\text{cyc}(G_0)} + ... + \frac{1}{\text{cyc}(G_0)}}_{\delta_{G_0}(v_0) \text{ times}} + ... + \underbrace{\frac{1}{\text{cyc}(G_{k-1})} + ... + \frac{1}{\text{cyc}(G_{k-1})}}_{\delta_{G_{k-1}}(v_{k-1}) \text{ times}} \right) && (39)\\
&\leq OPT \cdot \left(\frac{1}{\text{cyc}(G)} + ... + 1 \right) && (40)\\
&\in O(\log(\text{cyc}(G))) \cdot OPT. && (41)
\end{aligned}$$

We now show that this upper bound is tight by adapting the standard tight example for set cover. For any arbitrary n, consider the (multi)graph $G = (V, E)$ defined as follows:

$$V = \{u, o_1, ..., o_n\}, \tag{42}$$
$$E = \{\{u, o_i\} \times 2 \mid i \in [n]\}. \tag{43}$$

Assign weights on V as follows:

$$w(u) = 1, \tag{44}$$
$$w(o_i) = \frac{1}{n-i+1} \; \forall i \in [n]. \tag{45}$$

Using Theorem 6.2 and Claim 6.3, we have

$$\text{cyc}(G) = |E| - |V| + \text{comps}(G) = 2n - (n+1) + 1 = n \tag{46}$$
$$\delta_{G_i}(o_i) = \deg_{G_i}(o_i) - \text{comps}(G_i - o_i) = 2 - 1 = 1 \; \forall i \in [n]. \tag{47}$$

As such, a possible output F given by the algorithm is $\{o_1, ..., o_k\}$ with weight $1/n + ... + 1$, while the optimal solution is $\{u\}$ with weight 1.

6.2 Exercise 6.2

Give an approximation factor preserving reduction from the vertex cover problem to the feedback vertex set problem

Solution

Consider the instance $G = (V, E, w)$ of the vertex cover problem. We reduce it to a feedback vertex set instance $G' = (V, E', w)$ simply by duplicating the edges, that is

$$E' = \{e_1 = \{u, v\} \cup e_2 = \{u, v\} | e = \{u, v\} \in E\}$$

. Let $ALG_1 \subseteq V$ be the output of this instance G' with approximation ratio r. We claim that ALG_1, is also an r approximate solution to G.
Feasibility: Consider any $e = \{u, v\} \in E \implies e_1, e_2 \in E'$. It is clear that either $u \in ALG_1$ or $v \in ALG_1$, otherwise we have a simple cycle.
Approximation ratio: By assumption,

$$ALG_1 \leq rOPT(G')$$

. It suffices to show that $OPT(G) = OPT(G')$.
Assume otherwise, then we can simply pick the smaller set and it would be a feasible solution - contradicting the optimality.

7 Shortest Superstring

7.1 Exercise 7.1

Show that Lemma 7.3 cannot be strengthened to

$$\text{overlap}(r, r') < \max\{wt(c), wt(c')\}.$$

Solution

The lemma relies on the key fact that

$$\text{overlap}(r, r') \geq wt(c) + wt(c') \implies \alpha \circ \alpha' = \alpha' \circ \alpha.$$

This would allow us to cover elements of c and c' with lesser cost, contradicting minimality. This is not true if we consider $m = \max\{wt(c), wt(c')\}$.
Consider the following example :

$$S = \{"abab", "baba", "abaa", "baab", "aaba"\}.$$

Observe that the prefix graph has a minimum weight cycle $C : \{c, c'\}$ where

$$c = "abab" \to "baba" \to "abab" \quad , \quad wt(c) = 2,$$

$$c' = "abaa" \to "baab" \to "aaba" \to "abaa" \quad , \quad wt(c') = 3.$$

Choosing representation $r = "baba", r' = "abaa"$ gives

$$\text{overlap}(r, r') = 3 \not< \max\{2, 3\} = 3.$$

7.2 Exercise 7.2

Obtain constant factor approximation algorithms for the variants of the shortest superstring problem given in Exercise 2.16.

(a) Find the shortest string that contains, for each string $s_i \in S$, both s_i and s_i^R as substrings.

(b) Find the shortest string that contains, for each string $s_i \in S$, either s_i or s_i^R as a substring.

Solution (for (a))

Simply run Algorithm 7 on $S = \{s_i, s_i^R\}_{i\in[n]}$. It is obvious that the approximation factor still stays at 3.

Solution (for (b))

Note that (a) already gives a 6-approximation algorithm: let w^* be the optimal solution of (b). As $w' = w^* \circ (w^*)^R$ is an instance of (a), if w is the output from (a), we have

$$|w| \le |w'| = 2|w^*| \le 6 \cdot OPT. \tag{48}$$

8 Knapsack

8.1 Exercise 8.1

Consider the greedy algorithm for the knapsack problem. Sort the objects by decreasing ratio of profit to size, and then greedily pick objects in this order. Show that this algorithm can be made to perform arbitrarily badly

Solution

Let the knapsack capacity $B = n$. There are two elements x_1, x_2 where

$$\text{profit}(x_1) = 1 \quad , \quad \text{size}(x_1) = 1,$$
$$\text{profit}(x_2) = n \quad , \quad \text{size}(x_1) = n.$$

The profit to size ratio is 1 in both cases, so the greedy algorithm may suboptimally pick x_1. This gives us that ratio

$$r = \frac{OPT}{ALG} = n,$$

which can be arbitrarily bad.

8.2 Exercise 8.2

Consider the following modification to the algorithm given in Exercise 8.1. Let the sorted order of objects be $a_1, \ldots, a_n$. Find the lowest k such that the size of the first k objects exceeds B. Now, pick the more profitable of $\{a_1, \ldots, a_{k-1}\}, \{a_k\}$ (we have assumed that the size of each object is at most B). Show that this algorithm achieves an approximation factor of 2.

Solution

First, it is clear that if all objects can be contained in the knapsack, then this algorithm will already match the optimal solution.
Consider otherwise, so such k does exist. We first claim that

$$OPT \le \sum_{i=1}^{k} \text{Profit}(a_i).$$

Let the set $S = \{s_1, \ldots, s_m\}$ selected by OPT in order of profitability of total size $S' \le B$. Consider the knapsack as a B-array where each cell c (potentially empty) is filled by the profitability of s_i - let this be cell profitability P_c. Then, by our greedy choice

$$\begin{aligned} OPT &= \sum_{i=1}^{S'} P_i^{\text{OPT}} \\ &\le \sum_{i=1}^{S'} P_i^{\text{ALG}} \le \sum_{i=1}^{B} P_i^{\text{ALG}} \le \sum_{i=1}^{S''} P_i^{\text{ALG}} \\ &= \sum_{i=1}^{k} \text{Profitability}(a_i) \cdot \text{size}(a_i) = \sum_{i=1}^{k} \text{Profit}(a_i) \end{aligned}$$

where S'' is the total size of elements chosen by ALG. $\square$
Now, it follows clearly since we are picking the feasible set with better profit hence it must be at least half of OPT.

8.3 Exercise 8.3

Obtain an FPTAS for the following problem.

(Subset-sum ratio problem) Given n positive integers, $a_1 < ... < a_n$, find two disjoint nonempty subsets $S_1, S_2 \subseteq \{1, ..., n\}$ with $\sum_{i \in S_1} a_i \geq \sum_{i \in S_2} a_i$, such that the ratio

$$\frac{\sum_{i \in S_1} a_i}{\sum_{i \in S_2} a_i} \tag{49}$$

is minimized.

Solution

We first obtain a pseudo-polynomial algorithm ALG for the problem. Let $A_i = \{a_1, ..., a_i\}$ for any $i \in [n]$ and $s = \sum A_n$. Define the array I for $i \in \{0, ..., n\}$, $s_1, s_2 \in [s]$ and $s_1 \geq s_2$ as follows:

$$I[i, s_1, s_2] = \mathbb{1}[\exists S_1, S_2 \subseteq A_i, S_1 \cap S_2 = \varnothing : \sum S_1 = s_1, \sum S_2 = s_2]. \tag{50}$$

Initialize all entries of I to 0. We fill in the array as follows for all $s_1, s_2 \in [s]$ and $s_1 \geq s_2$, iterating i from 1 to n:

$$I[i, s_1, s_2] = I[i-1, s_1, s_2] \vee I[i-1, s_1 - a_i, s_2] \vee I[i-1, s_1, s_2 - a_i]. \tag{51}$$

After that, we iterate through all valid pairs of s_1 and s_2 and find the minimal s_1/s_2 where $I[n, s_1, s_2] = 1$. The correctness of this algorithm is obvious and the running time follows $O(ns^2) = O(n^3 m^2)$, where $m = \max A_n$.

Inspired by the knapsack FPTAS, we round each a_i to the nearest r. Let $\hat{a}_i = \lceil a_i / r \rceil$ and $\bar{a}_i = \hat{a}_i \cdot r$. Denote the optimal instance for the original and the reduced instance by (S_1, S_2) and (S_1', S_2') respectively, where S_1, S_2, S_1', S_2' are indices. Observe that

$$\widehat{OPT} = \frac{\sum_{i \in S_1'} \hat{a}_i}{\sum_{i \in S_2'} \hat{a}_i} \leq \frac{\sum_{i \in S_1} \hat{a}_i}{\sum_{i \in S_2} \hat{a}_i} \tag{52}$$

$$\Longrightarrow \widehat{OPT} \leq \frac{\sum_{i \in S_1} \bar{a}_i}{\sum_{i \in S_2} \bar{a}_i} = \frac{\sum_{i \in S_1} a_i}{\sum_{i \in S_2} \bar{a}_i} \cdot \frac{\sum_{i \in S_1} \bar{a}_i}{\sum_{i \in S_1} a_i} \leq OPT \cdot \frac{\sum_{i \in S_1} \bar{a}_i}{\sum_{i \in S_1} a_i}. \tag{53}$$

Let $s_1 = \sum_{i \in S_1} a_i$. Then the approximation ratio θ in (53) is bounded by

$$\frac{s_1 + |S_1| r}{s_1}, \tag{54}$$

as $a_i \geq 1$ for all $i \in [n]$. Assume for the moment that $a_n \in S_1 \cup S_2$, in which case $s_1 \geq m$. Then

$$\theta \leq \frac{m + nr}{m} = 1 + \frac{n}{m} \cdot r. \tag{55}$$

As such, if we let $r = \varepsilon m/n$, we obtain an $(1+\varepsilon)$-approximation algorithm with running time $O(n^5/\varepsilon^2)$. To resolve our non-general assumption, consider that for any $i \in \{2, ..., n\}$, if a_i is not part of S_1 or S_2 in the instance of A_i, the instance of A_{i-1} already provides the optimal solution. Define an array R with n elements and calculate them as follows:

$$R[1] = ALG(A_1), \tag{56}$$

$$R[i] = \min\{R[i-1], ALG(A_i)\} \; \forall i \in \{2, ..., n\}. \tag{57}$$

Finally, return $R[n]$ as the output. The time complexity of this procedure is $O(n^6/\varepsilon^2)$.

8.4 Exercise 8.4

Show that a strongly NP-hard problem cannot have a pseudo-polynomial time algorithm, assuming $P \neq NP$.

Solution

As the textbook does not provide a rigorous definition of strong NP-hardness, we supplement them below to aid our proof:

1. A problem Π is **strongly NP-complete** if remains NP-complete when every input must be given in the form $(n, a_1, ..., a_n)$, where $n \in \mathbb{N}$ and $a_1, ..., a_n \in \text{poly}(n)$.
2. A problem Π is **strongly NP-hard** if every strongly NP-hard problem has a pseudo-polynomial reduction to it. That is, if given any strongly NP-hard problem with input format $(n, a_1, ..., a_m)$, it can be reduced, in polynomial time, to an instance of Π with input format $(m, b_1, ..., b_m)$, where $m \in \text{poly}(n)$ and $b_i \in \text{poly}(a_i)$ for all i.

We now prove the statement. It is obvious that 3SAT is strongly NP-complete, using the input format $(3n, a_{1,1}, a_{1,2}, a_{1,3}, ..., a_{n,1}, a_{n,2}, a_{n,3})$, where n is the number of variables and clauses, and clause i is given by

$$\bigwedge_{j=1}^{3} \begin{cases} x_{a_{i,j}} & \text{if } a_{i,j} > 0 \\ \neg x_{-a_{i,j}} & \text{if } a_{i,j} < 0 \end{cases}. \tag{58}$$

Now suppose we have a strongly NP-hard problem Π. For any given instance of MAX-3SAT, let $(m, b_1, ..., b_m)$ be the equivalent instance of Π through a pseudo-polynomial reduction. If there is a multinomial p such that Π can be solved in $p(m, b_1, ..., b_m)$ time, and $q, \{q_{i,j}\}$ are polynomials such that

$$m \leq q(n), \tag{59}$$

$$b_{3(i-1)+j} \leq q_{i,j}(a_{i,j}) \; \forall i \in [n], j \in [3], \tag{60}$$

then MAX-3SAT can be solved in

$$q(p(3n), q_{i,j}(a_{i,j})) \tag{61}$$

time. As $a_{i,j}$ is bounded between $-3n$ and $3n$, (61) is a polynomial in n. Obviously, this is impossible unless $P = NP$.

9 Bin Packing

9.1 Exercise 9.1

Give an example on which First-Fit does at least as bad as $\frac{5}{3} \cdot OPT$.

Solution

We first define large, medium, small items:

$$L_6 = (\frac{64}{126}, \dots, \frac{64}{126}),$$

$$M_6 = (\frac{43}{126}, \dots, \frac{43}{126}),$$

$$S_6 = (\frac{12}{126}, \dots, \frac{12}{126}),$$

with the input

$$I = S_6 \circ M_6 \circ L_6,$$

that is, 6 items of each cateogry in that order.
Then, the optimal uses 6 bin by packing (L, M, S) in each bin.
For the First-Fit algorithm, it first fits all S items in one bin, say

$$b_1 = (S, S, S, S, S, S) \quad , \quad \text{size} = 6 \cdot \frac{12}{126} = \frac{72}{126}.$$

It is clear that M, hence L also, can not fit.
The algorithm then fits M_6 into 3 new bins b_2, b_3,

$$b_2, b_3 = (M, M) \quad , \quad \text{size} = 2 \cdot \frac{43}{126}.$$

Note that any L item can not fit into these bins either. So, the algorithm fits L_6 into 6 new bins.
Therefore, the algorithm uses $1 + 3 + 6 = 10$ bins, while OPT uses 6,

$$\implies ALG = \frac{10}{6} \cdot OPT.$$

Remark : I am not sure how to show a tight 2-factor example.

9.2 Exercise 9.2

(Johnson [149]) Consider a more restricted algorithm than First-Fit, called Next-Fit, which tries to pack the next item only in the most recently started bin. If it does not fit, it is packed in a new bin. Show that this algorithm also achieves factor 2. Give a factor 2 tight example.

Solution

The factor guarantee is similar to load balancing. Observe that in alg,

$$\text{size}(b_{2k}) + \text{size}(b_{2k+1}) > 1.$$

For OPT, we must have $\text{size}(b'_i) \leq 1$. Since each item is in some unique bin,

$$\sum_{b \in B} \text{size}(b) = \sum_{b' \in B'} \text{size}(b')$$

where B are the bins used in ALG, while B' are the bins used in OPT.
It follows that

$$|B| \leq 2 \cdot |B'|.$$

To see a tight example, consider

$$I = \underbrace{(\frac{1}{2}, \frac{1}{2n}, \ldots, \frac{1}{2}, \frac{1}{2n})}_{4n}.$$

Then, $OPT = n + 1$ by pairing each $(\frac{1}{2}, \frac{1}{2})$ in $2n$ bins, and all the $\frac{1}{2n}$s into one bin.
The Next-Fit algorithm clearly needs n bins for each pair $(\frac{1}{2}, \frac{1}{2n})$, so

$$\frac{ALG}{OPT} = \frac{2n}{n+1} = 2 - \frac{2}{n+1},$$

which can be arbitrarily close to 2.

9.3 Exercise 9.4

Prove the bounds on R and P stated in Lemma 9.4.

Solution

Let the distinct sizes in the set of items be $w_1, ..., w_K$. Each bin configuration is then a subset of the multiset

$$\{w_1 : \infty, ..., w_K : \infty\}. \tag{62}$$

We thus have the bound

$$R \leq \binom{M + K - 1}{M - 1} \leq \binom{M + K}{M}. \tag{63}$$

The bound on P is obtained similarly.

9.4 Exercise 9.6

Prove the following statement made in Lemma 9.5, "A packing for instance J' yields a packing for all but the largest Q items of instance J."

Solution

Let $a_1, ..., a_n$ be the item sizes, where $a_1 \leq ... \leq a_n$. As described, we divide the items into K groups from smallest to largest, that is, if $A_1, ..., A_K$ is the grouping, for $i \in [K]$,

$$A_i = \{a_j \mid iQ \leq j < (i+1)Q, j \leq n\}. \tag{64}$$

For $i \in [K]$, define the following operations:

$$\lfloor A_i \rfloor = \{a_{(i-1)Q+1} : |A_i|\}, \ \lceil A_i \rceil = \{a_{iQ-1} : |A_i|\}. \tag{65}$$

Then J is the instance on $\cup_{i\in[K]} \lceil A_i \rceil$, and J' is the instance on $\cup_{i\in[K]} \lfloor A_i \rfloor$. If $n \leq Q$, the statement is trivially true. Discard the Q largest items of J and the Q smallest items of J', and name the respective result $\hat{J}$ and $\hat{J}'$. After this,

$$\hat{J} = \{\lceil A_1 \rceil, ..., \lceil A_{K-2} \rceil, B\}, \ \hat{J}' = \{\lfloor A_2 \rfloor, ..., \lfloor A_{K-1} \rfloor\}, \tag{66}$$

where $B \subseteq \lceil A_{K-1} \rceil$. As $A_i \leq A_{i+1}$ (i.e. $\max A_i \leq \min A_{i+1}$) for $i \in [K-1]$, we have $\lceil A_1 \rceil \leq \lfloor A_2 \rfloor, ..., B \leq \lfloor A_{K-1} \rfloor$. In other words, a packing for $\hat{J}'$ (and thus J') must be a packing for $\hat{J}$, proving the statement.

9.5 Exercise 9.9

(C. Kenyon) Consider the following problem.

(Bin covering) Given n items with sizes a $1, ..., a_n \in (0, 1]$, maximize the number of bins opened so that each bin has items summing to at least 1. Give an asymptotic PTAS for this problem when restricted to instances in which item sizes are bounded below by c, for a fixed constant $c > 0$.

Solution

We first provide a polynomial algorithm to a restricted version of the problem where the number of distinct item sizes is not greater than a constant $K \in \mathbb{N}$. Observe that there must exist an optimal solution such that by removing at most $\lfloor 1/c \rfloor$ "residual" items, each bin has at most $M = \lceil 1/c \rceil$ items. In this case, each bin has at most $R = \binom{M+K-1}{K}$ configurations. There is at most n bins in any solution. Therefore, we can obtain an optimal solution by enumerating at most

$$f(n, c) = \binom{n+R}{n} \in \text{poly}(n) \tag{67}$$

cases. (We are putting R configurations into $n+1$ bins, with the last bin for residual items.)

Next, given any instance I of bin covering, derive J and J' as in Exercise 9.6. We know that $\lceil A_1 \rceil \leq \lfloor A_2 \rfloor, ..., B \leq \lfloor A_{K-1} \rfloor$, so J must be a valid bin covering for $\hat{J}'$. In other words,

$$OPT(J') + Q \geq OPT(\hat{J}') \geq OPT(J) \geq OPT(I) \tag{68}$$

$$\implies OPT(J') \geq OPT(I) - Q \geq OPT(I) - \varepsilon OPT(I) = (1-\varepsilon)OPT(I). \tag{69}$$

We have thus obtained the required algorithm.

10 Minimum Makespan Scheduling

10.1 Exercise 10.1

(Graham [114]) The tight example for the factor 2 algorithm, Example 10.4, involves scheduling a very long job last. This suggests sorting the jobs by decreasing processing times before scheduling them. Show that this leads to a $\frac{4}{3}$ factor algorithm. Provide a tight example for this algorithm.

Solution

The key observation is that now instead of $p_j < OPT$, we have a tighter bound that $p_j < \frac{1}{3}OPT$.
To see this, we first claim that that if M_i less than two jobs, then the makespan is optimal.
Proof: If M_i contains only one job, then it is trivial.
Otherwise M_i contains two jobs, so all other machines M_j also contain at least one job. Additionally, amongst the first jobs M_{j_1} of each M_j,

$$M_{i_1} \leq M_{j_1}.$$

Clearly, OPT must also assign machine M_i two jobs. If it does not assign p_j to machine M_i, this can only increase the makespan. $\square$
Now, assume M_i contains more than two jobs. In particular, we have

$$OPT \geq M_{i_1} + M_{i_2} + \dots M_{i_k}, \quad k \geq 2,$$
$$\implies OPT \geq kp.$$

If $k \geq 3$, we are done.
Otherwise assume $k = 2$ and $p_j > \frac{OPT}{3}$. Consider the set of jobs upto p_j, then OPT assigns all these jobs to m processors such that each processor has at most 2 jobs - in particular, each machine has load more than $M_{i_1} + M_{i_2} \geq 2p_j$. This is a contradiction since we would have a machine with load

$$L = 3p_j > 3 \cdot \frac{OPT}{3} = OPT.$$

Thus, we have that $p_j \leq \frac{OPT}{3}$ hence it follows

$$ALG = M_{i_1} + \dots M_{i_k} + p_j \leq OPT + \frac{OPT}{3} = \frac{4}{3} \cdot OPT.$$

To see a tight example, consider the input of $2m+1$ processes

$$I = \{2m-1, 2m-1, 2m-2, 2m-2, \dots m+1, m+1, m, m, m\}.$$

Then, the greedy algorithm returns

$$\{2m-1, m, m\}, \{2m-1, m\}, \dots, \{\frac{3}{2}m, \frac{3}{2}m-1\},$$

with maximum makespan $4m-1$. We can instead assign as follows:

$$\{m, m, m\}, \{2m-1, m+1\}, \dots, \{\frac{3}{2}m, \frac{3}{2}m\}$$

which achieves a maximum makespan of $3m$. In particular,

$$\frac{ALG}{OPT} \geq \frac{4}{3} - \frac{1}{3m}.$$

10.2 Exercise 10.2

(Horowitz and Sahni) Give an FPTAS for the variant of the minimum makespan scheduling problem in which the number of machines, m, is a fixed constant.

Solution

We first present a pseudo-polynomial dynamic programming algorithm for minimum makespan scheduling on a given upper bound M on the minimum makespan. We then round each job in a similar manner as in knapsack.

Let $P_i = \{p_1, ..., p_i\}$. Each entry in the dynamic programming table dp is defined as follows:

$$\text{dp}[i, \ell_1, ..., \ell_m] = \mathbb{1}[(\ell_1, ..., \ell_m) \text{ is a valid scheduling of } P_i], \tag{70}$$

for $i \in \{1, ..., n\}$ and $0 \leq \ell_1, ..., \ell_m \leq M$. To fill dp, initialize all dp entries to 0. Set $\text{dp}[1, p_1, ..., 0], ..., \text{dp}[1, 0, ..., p_1]$ to 1. After that, for $i \in \{2, ..., n\}$, we fill dp for $\ell_1, ..., \ell_m \leq M$ as follows:

$$\text{dp}[i, \ell_1, ..., \ell_m] = \text{dp}[i-1, \ell_1 - p_i, ..., \ell_m] \vee ... \vee \text{dp}[i-1, \ell_1, ..., \ell_m - p_i]. \tag{71}$$

After the dynamic programming is complete, scan through $\text{dp}[n]$ and obtain an 1-entry with minimal makespan. The correctness of this algorithm is trivial and its running time is $O(n \cdot M^m)$.

Next, round up each job to the nearest r, the exact value of which will be derived later. Let $\hat{p}_i = \lceil p_i/r \rceil$ and $\bar{p}_i = r \cdot \hat{p}_i$. Denote by S and $\bar{S}$ the optimal schedules of $P = \{p_1, .., p_n\}$ and $\bar{P} = \{\bar{p}_1, ..., \bar{p}_n\}$ respectively, with $OPT = \max S$ and $\overline{OPT} = \max \bar{S}$. It is easy to see that

$$\bar{p}_i \leq p_i + r \implies \overline{OPT} \leq OPT + nr. \tag{72}$$

Therefore, if we fix $r = \varepsilon \cdot LB/n$ and $M = 2LB/r$, we get an error bound of

$$nr = \varepsilon \cdot LB \leq \varepsilon \cdot OPT, \tag{73}$$

and a running time of

$$O(n \cdot M^m) = O(n \cdot (2LB/r)^m) = O(2^m \cdot n \cdot (n/\varepsilon)^m) = O(2^m \cdot n^{m+1}/\varepsilon^m). \tag{74}$$

11 Euclidean TSP

11.1 Exercise 11.1

Show that we may assume that the length of the bounding square can be taken to be $L = 4n^2$ and that there is a unit grid defined on the square such that each point lies on a gridpoint.

Solution

Scale all coordinates by the factor $\alpha = \frac{4n^2}{S}$. Distances and OPT scale by the same factor, so it suffices to prove the statement for the scaled instance. From now on the bounding square is fixed and its side length equals $L = 4n^2$.
Next, we choose a fine grid (spacing δ) inside the fixed square. Let

$$\delta := \frac{L}{2\sqrt{2}n^3}.$$

Note that $\delta > 0$ and this choice does not change the bounding square - it only defines a grid of spacing δ inside the (fixed) square of side L.
Then, we "snap" each point to the nearest grid point and bound the error. Each original point is moved by at most half a cell diagonal, so the Euclidean displacement of any point is at most $\frac{\sqrt{2}}{2}\delta$. We have

$$\text{increase per point} \leq 2 \cdot \frac{\sqrt{2}}{2}\delta = \sqrt{2}\delta.$$

Thus, for all points,

$$\Delta \leq n \cdot \sqrt{2}\delta.$$

Substitute the chosen δ:

$$\Delta \leq n\sqrt{2} \cdot \frac{L}{2\sqrt{2}n^3} = \frac{L}{2n^2}.$$

It clear that $OPT \geq L$. So,

$$\Delta \leq \frac{L}{2n^2} \leq \frac{OPT}{2n^2} \leq \frac{OPT}{n^2}.$$

Thus snapping to the δ-grid increases the optimal tour length by at most $\frac{OPT}{n^2}$.
Finally, multiply all coordinates by the factor $\frac{1}{\delta}$. This transforms the δ-grid into

a unit (integer) grid, and it multiplies all lengths (including OPT and the error Δ) by the same factor.

Combining the steps above, we obtain a scaled instance with a unit integer grid containing every point, and moving all points to grid points increases the optimal tour length by at most $\frac{OPT}{n^2}$, as required.

11.2 Exercise 11.2

Provide the missing details in the proof of Lemma 11.3.

Solution

Below we provide a rigorous and detailed dynamic programming formulation that finds the optimal well-behaved tour w.r.t. the basic dissection with limited crossings. Call a tour (on all nodes and a subset of the portals) "good" if it matches the above description. Consider a good tour τ. For the sake of convenience, assuming that τ never "walks along" the boundary of s (denoted ∂s), i.e. the set $\tau \cap \partial s$ is finite. We can ensure this by perturbing τ by an infinitesimal length towards either side of a gridline.

Given any square s, $\tau \cap s$ is a collection W of paths, and for each path w, it has an entry portal $\text{entry}(w)$ and exit portal $\text{exit}(w)$. Exclude the cases when w only touches s but does not enter it, i.e. w is a point, because they are irrelevant analysis. We say a portal p is used once for every instance where there is a path w such that $\text{entry}(w) = p$ or $\text{exit}(w) = p$. As τ is well-behaved, paths in W do not self-intersect in the interior of s, which means W is planar. As a result, if we walk along ∂s in any orientation starting from any point, the pairings of the portals never interleave. That is, there are no two distinct pairs p_1, p_2 and q_1, q_2 such that p_1, q_1, p_2, q_2 is the order they appear. Due to limited crossings, each portal can be used 0, 1 or 2 times. This gives 3^{4m} combinations. For each combination where total usage r is even, the number of non-interleaving pairings is given by the $(r/2)^{\text{th}}$ Catalan number, $C_{r/2}$. To restore W from pairings, we would also need the directions of its paths, totaling $2^{r/2}$ possibilities. Therefore, the number of good W-s (up to the directions and endpoints of paths) is given by

$$\sum_{\text{usage}|r \text{ is even}} C_{r/2} \cdot 2^{r/2} \leq 3^{4m} \cdot C_{4m} \cdot 2^{4m} = 6^{4m} \cdot C_{4m} \in n^{O(1/\varepsilon)}, \tag{75}$$

as $r \leq 2 \cdot 4m = 8m$.

The dynamic programming operates on a table dp. Each entry

$$\text{dp}[s, U, M, \Phi] \tag{76}$$

is defined to be the minimum length of a good W on s where usages of portals follow U, the matchings are given by M, and their directions given by Φ. The table is built bottom-up, with the base case being when s is a unit square. There are two cases:

1. There is no node in s. (After initial perturbation, all nodes lie on a gridpoint. Arbitrarily assign which square it belongs to.) Then simply connect each pair in M with a straight line and calculate the cost.

2. Otherwise, still connect pairs in M with straight lines. s is now divided into $|M|+1$ regions. Let the (p_1, p_2) and (q_1, q_2) be the two pairs of portals bounding the node. Observe that as s lives on ∂s, other portals trying to connect with the node would necessarily cross (p_1, p_2) or (q_1, q_2), regardless of the selection of paths. Therefore, simply try breaking the line between the two pairs to connect the node and report the shorter path.

For any non-unit s, let $s_1, ..., s_4$ be its children. We say that a combination of configurations $\{(s_i, U_i, M_i, \Phi_i)\}_{i\in[4]}$ is compatible with (s, U, M, Φ) if their portal usage and matching directions match, and the configurations do not induce an internal cycle. The latter can be prevented by making sure that no portal on the inner lines is an entry and an exit simultaneously. Its dp entries are then given by:

$$\mathrm{dp}[s, U, M, \Phi] = \min_{\{(s_i, U_i, M_i, \Phi_i\}_{i\in[4]} \text{ are compatible}} \sum_{i=1}^{4} \mathrm{dp}[s_i, U_i, M_i, \Phi_i]. \tag{77}$$

The correctness of the algorithm can easily be shown by induction. We now analyze its running time. Calculating each entry in the base case obviously takes constant time. As for the inductive case, note that the two level $(i+1)$ lines contained in s (the inner lines) have at most $4m$ portals on them. Note that once portal usages have been fixed, setting the matchings and their directions in one child square automatically fixes the others. Therefore, the number of cases we need to consider is bounded by

$$3^{8m} \cdot C_{r/2} \cdot 2^{r/2} = 3^{8m} \cdot C_{4m} \cdot 2^{4m} \in n^{O(1/\varepsilon)}. \tag{78}$$

Putting all of the above together, the running time is therefore in

$$\left(O(n^{O(1/\varepsilon)})\right)^2 = O(n^{O(1/\varepsilon)}). \tag{79}$$

11.3 Exercise 11.4

Prove Lemma 11.4.

Solution

As nodes have been snapped to gridpoints, LHS exactly measures the ℓ_1 length of π, except when a segment of π runs horizontally or vertically. In the usual case, the bounding factor of $\sqrt{2}$ applies. In the special case, π touches (once) a grid intersection for every vertical (resp. vertical) gridline it crosses, so the bounding factor is 2. This proves the lemma.

11.4 Exercise 11.6

Generalize the algorithm to norms other than the Euclidean norm.

Solution

The algorithm generalizes to L_p norms for $p \geq 1$ naturally.
The correctness and analysis follows analogously.

```
Scale and translate points so bounding box is $[0, L] \times [0, L]$ with
  $L = 4n^2$ ;
Snap each point to nearest integer grid point (Exercise 11.1) ;
Let $k = 2 + \lceil \log_2 n \rceil$, $L \leftarrow 2^k$ (Adjust to power of 2) ;
$m \leftarrow$ smallest power of 2 in $[\frac{k}{\epsilon}, \frac{2k}{\epsilon}]$ ;
Build basic dissection tree $T$ of depth $k$;
For each line in dissection, place portals $\frac{L}{2^i m}$ apart for level $i$ lines;
Initialize DP table for all useful squares;
for level $i$ from $k$ down to 0 do
    for each useful square $S$ at level $i$ do
        for each valid visit $V$ of $S$'s portals do
            if $S$ is leaf (unit square) then
                Compute optimal paths for points inside $S$ with portal
                  constraints;
            end
            else
                Consider all portal usage patterns on $S$'s four internal
                  sides;
                Consider all valid pairings consistent with $V$;
                Compute minimum cost from children's DP entries;
            end
        end
    end
end
return Tour from root's DP entry with all points covered
```

Remark: In fact, the algorithm extends to any norm that is Lipschitz equivalent to the Euclidian norm (within a constant factor of the Eucldien norm). Particularly, we need convexity so that short cutting works, and translation invariance so that we can move the points. This means for example L_p norms for $0 < p < 1$ will fail.