CS 5644: Analysis of Algorithms

Homework 11 Solution

1. For part (a), let the first object have weight *k/*2, the second object has weight *k*, and the third object has weight *k/*2. The algorithm will put each of the objects on their own trucks, but an optimal solution would combine the first and third objects into a single truck.

For part (b), let *W* denote the sum of the weights of all of the objects. We need a good lower bound on the number of trucks used in an optimal solution *OPT* . We can get this lower bound by

realizing that an optimal solution must use at least *W* trucks (packing exactly *k* into each truck).

*k*

We will show that we use at most 2*W*

*k*

trucks. Consider the first two trucks we will pack. It must

be that the sum of the weight put on these two trucks is greater than *k* (if it was at most *k* then we

would have put it all onto one truck). Partition the objects into groups where the first two trucks are a group, the next two are the second group, etc. Each of these groups has weight at least *k*, and therefore there are at most *W/k* such groups. Our algorithm will use at most 2 trucks to pack

each of these sets, and therefore the algorithm will use at most 2*W* trucks. Since *OPT* is at least

*k*

*W* , it follows that the algorithm is a 2-approximation algorithm.

*k*

1. We compute the flow using the augmenting path method of Ford-Fulkerson, and the minimum cut is all of the nodes reachable from *s* in the final residual network.

Original flow network (and original residual network). We choose the bolded path.



u

2

4

s

6

t

4

2

v

After pushing flow along our selected augmenting path:



u

2/2

2/4

s

0/6

t

0/4

0/2

v

Our new residual network, and our next augmenting path:



u

2

2

s

6

2

t

4

2

v

After pushing flow along our selected augmenting path:



u

2/2

2/4

s

0/6

t

2/4

2/2

v

The final residual network. There is no path from *s* to *t* so we are done.



u

2

2

s

6

2

t

2

2

2

v

The min cut is *S* = {*s, v*} and *T* = {*u, t*}.

1. We will view each unit of flow as an assignment of a client to a base station. We can view the underlying graph as a bipartite graph where the clients are one set of vertices and the base stations are the other set of vertices. We add a source vertex *s* and a sink vertex *t*. We add an edge of capacity 1 from *s* to each of the client vertices (this captures the constraint that each client can be assigned to at most one base station). We add an edge of capacity 1 from a client to each of the base stations whose distance is at most *r*. We then add edges from each base station to *t* with capacity *k* (this captures the constraint that each base station can have at most *k* clients assigned to it). This completes the construction.

We will now show that each client can be connected to a base station if and only if the maximum flow of this flow network is *n* (the number of clients). First suppose that there is a flow of value *n*. Then there must be one unit of flow to each of the clients from the source, and therefore one unit of flow leaving the client and going through a particular base station. Since the capacity on the edge leaving a base station is at most *k*, then we know that there is at most *k* units of incoming flow. Therefore we can feasibly assign each client to the base station as indicated by the flow to get a valid assignment of the clients to the base stations.

Now suppose that there is a valid way of assigning each of the clients to base stations. We will show how to construct a flow of value *n*. For each “client/base station” pair, set the flow on the corresponding edge in the flow network to be 1. Let *bi* denote the number of clients assigned to

base station *i*. We know that *bi* ≤ *k*, so we can set the flow on the edge from *i* to *t* to be *bi*. We set the flow on all of the edges from *s* to a client to be 1. Clearly this is a feasible flow, and its value is *n*.