6.854 Advanced Algorithms **Petar Maymounkov**

Problem Set 9 (November 8, 2005)

With: Benjamin Rossman, Oren Weimann, and Pouya Kheradpour

Problem 1a. Let $v_1, \ldots, v_k$ be an independent set in G. The corresponding independent set in the product graph consists of the points $v_1, \ldots, v_k$ within each of $G_{v_1}, \ldots, G_{v_k}$ – a total of k^2 vertices.

Any two vertices (of the corresponding independent set) within a G_v are not connected with an edge, because they are not in the original graph. Also any pair of vertices where one is in G_v and the other is in G_w ($v \neq w$) are also not connected, because there are no edges between any of the vertices in G_v and G_w, since v and w are not connected in G (as they are part of the original independent set).

Problem 1b. Let m be the number of subgraphs G_v of the product graph that contain vertices of the independent set. Call these subgraphs $G_{v_1}, \ldots, G_{v_m}$. By construction there are no edges between the vertices of any pair G_{v_i} and G_{v_j} and hence $v_1, \ldots, v_m$ is an independent set in G of size m.

Also, let G_v be the subgraph that has the largest number of vertices of the independent set. Then by averaging we know that G_v must contain at least s/m vertices of the independent set. And hence by construction, these vertices form an independent set in G of size s/m.

So, G must have an independent set of size $w = \max\{m, s/m\}$. We claim that $w \geqq \sqrt{s}$. Assume not. Then $w < \sqrt{s}$, and hence both $m < \sqrt{s}$ and $s/m < \sqrt{s}$, which implies that the size of the original independent set $s < m(s/m) = s$ – contradiction.

Problem 1c. Begin with a graph G that has a maximal independent set of size OPT. Create the product graph, in $O(n^2)$ time, whose maximal independent set is at least of size OPT^2 (according to part 1a). Find an α-approximation in the product graph, which produces an independent set in the product graph of size at least OPT^2/α in $P(n^2) = \text{poly}(n^2)$ time, where $P(n)$ is the runtime of the α-approximation scheme. Project this independent set down to the original graph (using part 1b), to get an independent set of size $\mathrm{OPT}/\sqrt{\alpha}$ in the original graph. Hence we just obtained a $\sqrt{\alpha}$-aproximation algorithm, which runs in $n^2 + P(n^2)$ time.

Notice that the new runtime is polynomial and has double the degree of the polynomial $P(n)$. Hence every recursive application of this procedure doubles the degree of the runtime polynomial.

Given any desired approximation factor ϵ. Apply the above procedure $r = -\log_2 \log_\alpha \epsilon$ times to get an ϵ-approximation. This procedure runs in polynomial time, since $r = O(1)$ and our runtime is $O(n^{2^r \deg P(n)})$.

Problem 2a. This is a non-algorithmic argument:

1. Begin with a Steiner tree in G, with edge cost OPT
2. Double each edge of the Steiner tree to turn the Steiner tree into 1 big cycle, with edge cost 2OPT
3. On the cycle, replace the path between every pair of adjacent-on-the-cycle terminal vertices, with the shortest path between them. Get a cycle of cost $\leqq$ 2OPT
4. Project this cycle to G', where it preserves its cost, but now it consists only of terminal vertices
5. Remove some subset of edges from the cycle (in G') to turn it into a tree (which spans all terminal vertices of G, and hence all vertices of G'). This further reduces the cost, and maintains that cost $<$ 2OPT
6. Observe that the MST in G' is cheaper than that tree, hence the cost of MST in G' is no bigger than 2OPT

Problem 2b.

1. Compute all pairs shortest paths in G using Floyd-Warshall, and create G'. Runtime $O(n^3)$
2. Compute MST in G' in $O(n^2 \log n)$
3. Project MST in G' to a subgraph of G that contains all terminal vertices (by construction) and has cost no bigger than 2OPT
4. Remove cycles from this graph arbitrarily so it can become a tree. Do this using DFS e.g.

The total runtime of this algorithm is $O(n^3)$.

Problem 3a. Set up a dynamic program. Let the problem instance be identified by a vector $\mathbf{x} = x_1, \ldots, x_k$ describing number of items of each type. And set $B(\mathbf{x})$ to be the optimal number of bins for $\mathbf{x}$. Let also Q be the set of all possible single-bin packings. We can describe any element $\mathbf{y}$ of Q by giving a vector describing the number of items of each type in the bin, $\mathbf{y} = y_1, \ldots, y_k$, hence $|Q| = n^k$.

For any problem $\mathbf{x}$, the "last bin" is an element of Q. For each possibility for the last bin, we lookup the best allocation of the remaining items. So, in particular:

$$B(\mathbf{x}) = 1 + \min_{\mathbf{y} \in Q} B(\mathbf{x} - \mathbf{y})$$

We also have $B(\mathbf{0}) = 0$. Finally, we reconstruct the actual solution by tracing back through the dynamic program. There are a total of n^k nodes in the dynamic program, and computing each requires an enumerate of size n^k, so the total runtime is n^{2k}.

Problem 3b. Add the remaining small items one at a time:

1. Scan through the "usable" bins

2. Eeach time we encounter a bin that has $< \epsilon$ free space, set it aside as "unusable"
3. If we encounter a bin with $\geqq \epsilon$ free space, place the current item in it and stop scanning
4. If we reach the end of the bin list (they must have all been marked as unusable), create a new bin and place the item there
5. Repeat above procedure for each small item

Obvious amortized analysis shows that this runs in linear time (each bin is processed at most as many times as the number of small items that go in it plus 1).

If no new bins are created, then the packing uses B bins. Otherwise, assume that we end with $r > B$ bins. By construction $r - 1$ of them must have $< \epsilon$ free space each, hence there is at least $(r-1)(1-\epsilon)$ volume of items in them, hence $B^* > (r-1)(1-\epsilon)$. Hence:

$$r < 1 + \frac{B^*}{1-\epsilon}$$
$$r < 1 + (1+2\epsilon)B^*$$

Note: The problem set statement may have a typo. Achieving $r \leqq 1 + (1+\epsilon)B^*$ is not possible in all cases. Consider 1000 items of size 0.5, 1100 items of size 0.2, and 201 items of size 0.33. Optimum is 787 bins. Set $\epsilon = 0.33$. And consider a placement of the large items where each bin has 1 item of size 0.5 and 1 item of size 0.2. Packing the small items could entail 100 bins with 1 item of 0.2, and 2 items of 0.33, and 1 more bin with 1 item of 0.33. Total of 1101 bins, which is much bigger than (1 + 0.33) 787.

So we've shown that the number of bins we get is $\max\{B, 1 + (1+2\epsilon)B^*\}$.

Problem 3c. Rounding up by to the next power of $1+\epsilon$ introduces two problems. Frist, a an item may become larger than 1 rendering the whole problem infeasible. But even if we avoid rounding above 1. We can still have a situation where rounding with an arbitrarily small ϵ can increase the size of the optimal solution by a fixed constant factor. E.g. consider a problem with $n/2$ items of size $1/3$ and $n/2$ items of size $2/3$. The optimum here is n. After rounding with any $\epsilon > 0$, the optimum becomes $\geqq 1.5n$.

Problem 3d. Let the item sizes be $a_1, \ldots, a_n$ in decreasing order, and let $\tilde{a}_1, \ldots, \tilde{a}_n$ be the respective increased item sizes after grouping. Take an optimal packing, and replace each item a_i in the packing by $\tilde{a}_{i+k}$; also, throw away the smallest k items from the original packing (since we are not replacing them with anything). The result is a packing for all but the largest k items, using the items' increased sizes. This new packing still has B^* bins, and it is feasible because each original item is replaced by no bigger item (because of the definition of the grouping procedure). If we now accommodate each of the largest k items (with increased sizes) in a separate new bin each, we get a packing for all of the increased-size items with $B^* + n/k$ bins. By this we've shown that an optimum solution of the grouping problem is at most n/k bins larger than the optimum solution to the original problem.

Problem 3e.

1. Take all items $> \epsilon/2$, the large items, n_{LARGE} in count
2. Note that for the optimum of the large items, $\text{OPT}_{\text{LARGE}}$, we have $n \geqq \text{OPT}_{\text{LARGE}} \geqq n_{\text{LARGE}}\epsilon/2$ and $\text{OPT}_{\text{LARGE}} \leqq \text{OPT}$
3. Set $k = 2/\epsilon^2 = O(1)$
4. Group large items and round up (as described in part 3d)
5. Solve optimally, using part 3a
6. Get a solution for large items w_{LARGE}, where:

$$\begin{aligned} w_{\text{LARGE}} &\leqq \text{OPT}_{\text{LARGE}} + n_{\text{LARGE}}\epsilon^2/2 \\ &\leqq (1+\epsilon)\text{OPT}_{\text{LARGE}} \\ &\leqq (1+\epsilon)\text{OPT} \end{aligned}$$

7. Add small items into packing of large items using part 3b

8. Get a packing of all items with w bins. We have that:

$$\begin{aligned} w &= \max\{w_{\text{LARGE}}, 1+(1+\epsilon)\text{OPT}\} \\ &\leqq \max\{(1+\epsilon)\text{OPT}, 1+(1+\epsilon)\text{OPT}\} \\ &= 1+(1+\epsilon)\text{OPT} \end{aligned}$$

Total runtime is dominated by the time to solve for the grouped large items optimally, i.e. $O(n^{2k}) = O(n^{4/\epsilon^2})$.

Problem 4a. Maximum completion time is $T = \sum p_j = \text{poly}(n)$. The ILP is:

$$\min \sum_j w_j \sum_{t=1}^{T} t x_{jt} \tag{9.1}$$

$$x_{jt} = 0 \quad \text{or} \quad x_{jt} = 1 \tag{9.2}$$

$$\sum_{t=1}^{T} x_{jt} = 1, \quad \text{for all } j \tag{9.3}$$

$$\sum_{t=1}^{t_0} x_{it} \geqq \sum_{t=1}^{t_0} x_{jt}, \quad \text{for every "}i\text{ must precede }j\text{", and every } t_0 \tag{9.4}$$

$$\sum_{t=1}^{t_0} \sum_j x_{jt} p_j \leqq t_0, \quad \text{for all } 1 \leqq t_0 \leqq T \tag{9.5}$$

Problem 4b. Define h_j to be the smallest integer number such that:

$$\sum_{t \leqq h_j} x_{tj} \geqq \sum_{t > h_j} x_{tj}$$

Then we have:

$$\begin{aligned}
\bar{c}_j &= \sum_t t x_{jt} \\
&= \sum_{t \leqq h_j - 1} t x_{jt} + \sum_{t > h_j - 1} t x_{jt} \\
&\geqq \sum_{t \leqq h_j - 1} t x_{jt} + h_j \sum_{t > h_j - 1} x_{jt} \\
&\geqq h_j \sum_{t > h_j - 1} x_{jt} \\
&\geqq h_j / 2
\end{aligned}$$

Problem 4c. Let "i must precede j", and assume on the contrary that $h_i > h_j$. That implies that $\sum_{t \leqq h_j} x_{jt} > \sum_{t \leqq h_j} x_{it}$, which violates constraint 9.4.

Problem 4d. WLOG, $h_1 \leqq \cdots \leqq h_n$. By 9.5 (with $t_0 = h_j$) we have that:

$$\begin{aligned}
h_j &\geqq \sum_{t=1}^{h_j} \sum_{i \leqq j} x_{it} p_i \\
&\geqq \sum_{i \leqq j} \sum_{t=1}^{h_i} x_{it} p_i \\
&\geqq \sum_{i \leqq j} \frac{p_i}{2}, \quad \text{by definition of } h_i
\end{aligned}$$

Therefore $2h_j \geqq p_1 + \cdots + p_j = c_j$. From part 4b we know that $4\bar{c}_j \geqq 2h_j$, and hence conclude that $4\bar{c}_j \geqq c_j$.

Problem 4e.

1. Solve LP relaxation (where $0 \leqq x_{jt} \leqq 1$)
2. Get relaxed optimum $w = \sum_j w_j \hat{c}_j$
3. Compute h_j's

4. Order job's according to h_j's (as in part 4c)
5. Resulting schedule is feasible according to part 4c
6. Resulting schedule cost is $\sum_j w_j c_j \leqq 4 \sum_j w_j \bar{c}_j$ (according to part 4d), i.e. within a factor 4 of optimum
7. Hence we get a 4-approximation