1

Approximation and online algorithms: CS60023: Spring 2020

Sudebkumar Prasant Pal

March 9, 2024

Contents

*Unfinished version: Only to be used as class notes. Not for circulation. Course CS60023.

email:spp@cse.iitkgp.ernet.in
Department of Computer Science and Engineering and
Centre for Theoretical Studies
Indian Institute of Technology, Kharagpur 721302, India.

1 Upper/lower bounds for the optimal in maximization/minimization problems

Suppose we have an optimization problem (such as that of computing a minimum sized vertex cover or a maximum cardinality stable set) where a certain quantity or parameter must be minimized or maximised. If OPT is the value of the optimal soution and we can compute a solution of value v in polynomial time using an algorithm A, then we say that $\frac{v}{OPT}$ is the *approximation ratio* for the algorithm A. Assume that this is a minimization problem. This ratio is at least one for such minimization problems. If we do not know OPT but we know a *lower bound* $m \leq OPT$ for OPT, then we can say that the approximation ratio $\frac{v}{OPT}$ is at most $\frac{v}{m}$. So, even not knowing OPT, we can estimate an upper bound $\frac{v}{m}$ on the approximation ratio $\frac{v}{OPT}$ for algorithm A, if we know just any lower bound for OPT such as m. [Knowing OPT could be difficult because computing the value of OPT may require exponential time for NP-hard problems.] The tighter the lower bound m, the better is our approximation ratio estimate for the minimization problem. Therefore, establishing higher (tighter) lower bounds for OPT is crucial in developing approximation algorithms with smaller approximation ratios. Note that in the case of a maximization problem, we can in a similar manner define the approximation ratio as $\frac{v}{OPT}$, which can never exceed unity, and the lower the upper bound estimate for OPT, the better would be the lower bound on the approximation ratio for the algorithm A.

1.1 DAG subgraphs of directed graphs

We know that *DAGs (directed acyclic graphs)* do not have *directed cycles*. We wish compute a large DAG subgraph of a given *directed graph* G. If OPT is the number of edges in the maximum size DAG in G, then we know that $OPT \leq e$, where e is the number of directed edges in G. If we can partition the set of e edges into two sets and then choose the bigger one then we are ensured

at least $\frac{e}{2} \geq \frac{OPT}{2}$ edges. Naturally both the sets in the partition must induce DAGs. How do we achieve this requirement? We use the total ordering property of integers; we simply number the n vertices of G from 1 through n arbitrarily. Then we take all *forward* edges in one set and *backward* edges in the other set. An edge (i, j) is a *forward* edge if $i < j$ and *backward* if $j < i$. Clearly, both these sets are DAGS and we can select the larger one. See Problem 1.9 on page 7 in [8].

1.2 Large cuts for undirected graphs

We know that the size of a cut in a graph cannot exceed e, where e is the number of edges in the graph. If OPT is the size of the maximum sized cut in G then $OPT \leq e$. We can start with just about any cut and then keep improving it as much as we can. We can stop when our incremental step cannot increase the cut size. This incremental step could be moving one vertex across the cut only if it has a larger number of neighbours in its own current side of the cut compared to the number of neighbours on the other side of the cut. When we stop, we find that each vertex has more neighbours on the opposite side, that is, at least half its degree is *exhausted* across the cut. Therefore, we have total vertex degree across the cut at least half the sum of degrees of all vertices, which is at least $\frac{2e}{2} = e$. However, this count is just twice the number of edges across the cut as each edge counts once in the degree of its two vertices. So, we conclude that at least $\frac{e}{2}$ edges are across the cut, which is more than $\frac{OPT}{2}$.

1.3 Minimum maximal matchings

See Problem 1.2 on page 8 in [8]. Consider an undirected graph $G(V, E)$. Let M be a maximal matching of m edges and let OPT be the size of a maximum cardinality matching M'. We wish to show that $m \geq \frac{OPT}{2}$. To this effect we first observe that all the OPT edges of the maximum matching M' are incident on the $2m$ vertices of M because these vertices also cover all edges in M'. In other words, the $2m$ vertices of M *hit* all edges in M' and thus form a vertex cover for M. Since no two edges of M' can be incident on the same vertex, we have at most $2m$ edges in M', that is, $OPT \leq 2m$. The approximation ratio is therefore $\frac{|M|}{|M'|} = \frac{m}{OPT} \geq \frac{m}{2m} = \frac{1}{2}$.

1.4 Vertex cover using DFS tree: Ratio factor two

Note that internal vertices of any DFS tree of a connected undirected graph G form a vertex cover of G. Why? So, this vertex cover can be computed in polynomial time. See Problem 1.3 on page 8 in [8].

If the number of internal vertices is m then we can show that there is a matching of size $\lceil \frac{m}{2} \rceil$, a lower bound on vertex cover size. So, the size OPT of the minimum vertex cover is no lesser than this lower bound. So, $m \leq 2 \times OPT$.

1.5 Vertex cover from matching: Ratio factor two

In the next section 2, we observe that the size of any maximal matching in an undirected graph is a lower bound for the size of the minimum vertex cover.

1.6 Vertex covering using a large cut

The following problem is due to Vishnoi, given as Exercise 2.5 on page 23 of [8]. If we compute a large cutset H of cardinality at least half the size of the maximum cutset in an undirected graph G then $G \setminus H$ has maximum degree $\frac{\Delta}{2}$ if G has maximum degree Δ. Then, by induction we can show that a factor $\log \Delta$ algorithm for computing a vertex cover of G can be designed. The set of edges in H can be viewed as a bipartite graph on incident vertices across the large cut. We can also compute the exact vertex cover for this bipartite graph in polynomial time. The vertex cover for $G \setminus H$ is computed recursively to within a factor $\log \frac{\Delta}{2}$ of the size of its minimum vertex cover. So, a vertex cover for G of (i) size OPT_H for H, and (ii) (recursively) for $G \setminus H$ of size $(log\frac{\Delta}{2})OPT_{G\setminus H}$ gives a vertex cover of size at most $OPT_H + (\log \frac{\Delta}{2})OPT_{G\setminus H} \leq OPT_G + (log\Delta - 1)OPT_G = (\log \Delta)OPT_G$.

1.7 Exercises

Section 2.4 Exercises 2.1, 2.2, 2.3 and 2.6 from pages 22-24 in [8].

2 The approximation algorithm with ratio factor two for vertex covering

For an undirected *simple* graph $G(V,E)$, a set $W \subseteq V$ is a *vertex cover* if, for every edge $\{u,v\} \in E$, either u or v or both are in W. We wish to find a *minimum cardinality vertex cover* for the graph $G(V,E)$. This being an NP-hard problem, it is worthwhile searching for polynomial time approximation algorithms.

We can try several heuristics. We may select (and delete) an arbitrary vertex $v \in V$ for inclusion in vertex cover C and drop all edges incident on v. This step can be repeated until the graph becomes empty (of edges). Alternatively, we may use another rule, where we select an arbitrary edge $\{u,v\} \in E$ and include both u and v in C; we drop all edges incident on u and v and repeat the process until the graph becomes empty.

For a *straight line graph* (that is, a simple path of n vertices and $n-1$ edges), the first method finds a vertex cover of size $n-1$, which is within twice the size of the optimal vertex cover of size $\lfloor n/2 \rfloor$. Why? Does it work well also for other classes of graphs like trees, planar graphs, and general graphs?

The second one chooses both vertices of all edges in a *maximal matching* S to be included in the computed vertex cover C. [*A matching is a set S of edges where no two edges in M share any vertex. A matching S is called a maximal matching if we cannot add an additional edge to M to get a larger matching.*] We analyze the second heuristic, following the exposition in [2]. How well does this second heuristic work for straight line graphs? If C^* is any minimum vertex cover then $|S| \leq |C^*|$, where S is any maximal matching. Why? Vertices in C^* have to cover each edge in the *(maximal) matching S*. So, C^* must include at least one vertex from each of the S edges. The $2|S|$ vertices comprise the computed approximate vertex cover C. So, $|C| = 2|S| \leq 2|C^*|$, since $|S| \leq |C^*|$. This gives a polynomial time algorithm yielding a vertex cover that is certainly at most twice the size of the minimum vertex cover.

[*It is interesting to note that any matching in a graph would force at least as many vertices in the vertex cover as twice the number of edges in the matching. Thus, the cardinality of the*

maximum matching gives a lower bound on the cardinality of any vertex cover. Do these two cardinalities ever coincide for any classes of graphs?]

From the algorithmic angle, we have already noted that the vertices of edges forming a *maximal* matching cover all edges, and a maximal matching can be computed using the *greedy* approach as in the second heuristic stated above (see [8]). [As mentioned earler, the *maximal matching* is such that none of its supersets enjoys the same property.] So, observe that a vertex cover generated by our approximation algorithm might as well be smaller than twice the cardinality of the *maximum matching*. In such cases, where the discovered maximal matching is smaller than the maximum matching, we indeed have some saving.

3 The cardinality (unweighted) set cover problem

We now generalize the 'cover' problem to general sets of objects from a universal set U. Let $\mathcal{S}$ be a collection of subsets $S \subset U$ such that the collection $\mathcal{S}$ of sets covers the entire set of objects in U, that is $\cup_{S \in \mathcal{S}}(S) = U$. We wish to find the smallest cardinality collection $\mathcal{C}^* \subseteq \mathcal{S}$ of sets so that $\mathcal{C}^*$ covers all the elements in U, that is $\cup_{S \in \mathcal{C}^*}(S) = U$.

[For vertex covering, $\mathcal{S}$ corresponds to the set of all vertices in the graph; the set of all edges incident at a vertex forms a subset $S \in \mathcal{S}$. So, the cardinality of $\mathcal{S}$=$|V|$. The elements in U are the edges of the graph. The vertex cover problem is a special case of the set cover problem. Can we then conclude that the set cover problem is also NP-hard, given that the vetex cover problem is known to be NP-hard?]

For the general set cover problem, we wish to find a good (small) cover $\mathcal{C} \subseteq \mathcal{S}$ of U such that $\cup_{S \in \mathcal{C}}(S) = U$. We show that such a cover $\mathcal{C}$ can be found in polynomial time with ratio bound $O(\log |U|)$, that is, $|\mathcal{C}| = O(|\mathcal{C}^*| \log |U|)$. We follow the exposition as in Cormen et al. [2]. Surprisingly, a simple heuristic works; we choose sets $S \in \mathcal{S}$ in decreasing order of the cardinality of the set of new elements covered by the remaining sets, until all elements of U are covered. The sets thus selected constitute the collection $\mathcal{C} \subseteq \mathcal{S}$.

3.1 The logarithmic approximation ratio on the cardinality of the greedy set cover

In order to establish the approximation ratio, we need a *charging scheme* for elements of U. We add one set at a time to the set cover. Whenever we select the next set from $\mathcal{S}$ to be included in the the set cover $\mathcal{C}$ in our approximation algorithm, we assign some *prices* to (only) the new elements of S as follows. If the $i-1$ sets selected so far are S_1, S_2, ..., S_{i-1}, then we have already assigned some *prices* to the elements of these sets. When the subsequent set S_i is selected, some more elements from S_i are introduced in to the set cover that were not covered by the previous $i-1$ sets. The set S_i is selected because it has the largest number $|S_i \setminus (S_1 \cup S_2 \cup \cdots \cup S_{i-1})|$ of new elements amongst all the sets in $\mathcal{S} \setminus \{S_1, S_2, \cdots, S_{i-1}\}$. The *price* applied on each new element is

$$\frac{1}{|S_i \setminus (S_1 \cup S_2 \cup \cdots S_{i-1})|}$$

Each element is charged with a *price* only once; let the *price* assigned to an element $x \in U$ be c_x. Observe that the sum of all weights charged is

$$|C| = \sum_{x \in U} c_x \tag{1}$$

[Each new element is charged (only once) with a price that is the inverse of the number of new elements introduced by the new set containing them. So, the sum total of all weights is equal to the number $|C|$, of sets selected by the approximation algorithm.]

We now define a quantity

$$\sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x$$

for an (unknown) optimal set cover $\mathcal{C}^*$. Observe that

$$|C| = \sum_{x \in U} c_x \leq \sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x. \tag{2}$$

The reason is that the sets in the optimal cover $\mathcal{C}^*$ might intersect. So, an element of x may be counted several times on the right hand side of inequality 2, whereas each element is counted exactly once on the left hand side.

Now that we have an upper bound on the cardinality of the approximate set cover, we assume (as shown in [2]), that

$$\sum_{x \in S} c_x \leq H(|S|), S \in \mathcal{S} \tag{3}$$

Here, $H(n) = O(\log n)$ is the harmonic sum

$$\sum_{1 \leq i \leq n} \frac{1}{i}$$

The proof of the inequality 3 is postponed to Section 3.2.

We can now see from inequalities 2 and 3 that

$$|C| \leq \sum_{S \in C^*} H(|S|) \leq |C^*|.H(max\{|S| : S \in \mathcal{S}\}) \tag{4}$$

The right hand side of inequality 4 leads to the desired upper bound on the approximation ratio required. The technique used in this section was a scheme for charging the elements $x \in U$ with weights c_x as we execute the approximation algorithm and compute the approximate set cover C. These charges as assigned by the greedy algorithm to the elements of U are then used to sum the charges of elements $c_x \in S$, summed up over the sets S in the optimal set cover C^*. This sum turns out to be proportional to the cardinality of the optimal set cover; constant of proportionality exceeds unity and deterines the logarithmic approximation ratio. For the complete proof and derivation of the crucial inequality 3, as applicable to all sets S in the collection $\mathcal{S}$, see pages 1036-37 of Cormen et al. [2] and Section 3.2.

3.2 The crucial inequality

Supporting the top level analysis of Section 3.1, we now establish the crucial inequality 3 as follows. See details in pages 1036-37 in Cormen et al. [2].

3.3 A simpler alternative analysis

Given a collection $\mathcal{S}$ of subsets of U, we wish to find a minimum cardinality collection $\mathcal{C}$ such that, $\cup_{S\in\mathcal{C}} S = U$. We initizlize $\mathcal{C} = \varnothing$. All elements of U are initially *uncovered*. As long as *uncovered* elements exist, we repeatedly pick a set S that contains the maximum number of uncovered elements and we set $\mathcal{C} = \mathcal{C} \cup \{S\}$. When all elemets of U are covered, we declare the collection $\mathcal{C} \subseteq \mathcal{S}$ the *approximate* minimum set cover of U from the collection $\mathcal{S}$. We say that $|U|$=n=n_0. Let n_i be the number of elements in U *uncovered* after iteration i. Let OPT be the minimum number of sets from $\mathcal{S}$ required to cover all the elements in U. We establish the following claims.

Claim I: $n_{i+1} \leq n_i \left(1 - \frac{1}{OPT}\right)$
Claim II: $|\mathcal{C}| \leq OPT \ln n$

We establish the claims as follows. We know that OPT sets can cover all elements in U. Suppose $\mathcal{C}_i$ is the collection of sets already selected so far by the greedy algorithm in the first i iterations. Some sets (possibly none) of the optimal set cover C^* might have already been selected in the collection $\mathcal{C}_i$. Let the number of sets of C^* that do not belong to $\mathcal{C}_i$ be p. Clearly $p \neq 0$. If $p = 0$ then we would have covered all elements in U and the algorithm would have to be terminated. These p sets can however cover the $|U \setminus \cup_i \mathcal{C}_i|$ remaining elements where clearly $p \leq OPT$. So, these $p \leq OPT$ sets cover the remaining n_i elements, and also possibly some more elements from $\mathcal{C}_i$. Each of these p remaining sets of the optimal set cover also possibly cover some elements of $\mathcal{C}_i$. We claim that at least one of these p sets covers at least $\frac{n_i}{OPT}$ elements of $U \setminus \mathcal{C}_i$. If each of these sets had less than $\frac{n_i}{OPT}$ elements of the n_i remaining elements then it would not have be possible for these $p \leq OPT$ sets to cover all these remianing n_i elements. The number of elements remaining *uncovered* after the $(i+1)$th iteration is therefore $\leqslant n_i - \frac{n_i}{OPT} = n_i \left(1 - \frac{1}{OPT}\right)$; the greedy heuristic chooses a set with at least $\frac{n_i}{OPT}$ elements.

Suppose the algorithm runs for t iterations until all elements of U are covered. Then, $|\mathcal{C}| = t$. We have, $n_t \leqslant n_{t-1}\left(1 - \frac{1}{OPT}\right) \leqslant n_0 \left(1 - \frac{1}{OPT}\right)^t = n\left(1 - \frac{1}{OPT}\right)^t$. Now, $1 - x < e^{-x}$, for $x \neq 0$. Thus, $n_t < ne^{-\frac{t}{OPT}}$. Let $t = OPT \ln n$. Then, $n_t < ne^{-\frac{t}{OPT}} = ne^{\frac{-OPT \ln n}{OPT}} = ne^{-\ln n} = 1$, or $n_t = 0$. This means after t iterations all the elements of U have been covered. This happens if $t = OPT \ln n$ or $|\mathcal{C}| = OPT \ln n$. The exposition in this subsection is based on Prof. Abhiram Ranade's lectures.

4 The weighted set cover problem

In this case, each set $S \subseteq U$, has a positive and rational weight $c(S)$. Here, U is the universal set of n elements and the collection of m sets $\mathcal{S} = \{S_1, S_2, \cdots, S_m\}$. We need to find the minimum weighted subset of $\mathcal{S}$, that covers U, given that $\mathcal{S}$ covers U. We now show how the ratio approximation factor of $H(n)$ is attained following the exposition in [8].

4.1 The main analysis of the logarithmic ratio factor

The greedy selection rule for the next set S is similar to the rule in the unweighted set cover heuristic in Section 3. If the set of already covered elements is C, then $|S \setminus C|$ is the number of new elements. Let $c(S)$ be the total cost of the elements of S. Then the cost per element added afresh is $\alpha = \frac{c(S)}{|S \setminus C|}$. This is called the *cost effectiveness* of the set S. In each greedy step, we select that set S whose cost effectiveness is minimum; for each element $e \in S$, we assign $price(e) = \alpha$. Now, let $e_1, e_2, \cdots, e_n$ be the sequence in which the selected sets covered the n elements. We have the following non-trivial upper bound:

$$price(e_k) \leq \frac{OPT}{n-k+1}$$

We establish this bound below; first we show how to use this bound. We know that summing $price(e_k)$ over all $e_k \in U$ gives us the sum of weights of sets in the set cover computed by our greedy algorithm. This is clearly $H(n) \times OPT$, by the use of the above upper bound for $price(e_k)$.

We now proceed to establish the upper bound on $price(e_k)$ as given in [8]. At any *stage* during the computation of the greedy (approximate) set cover, the elements covered constitute the set C. Consider he set $U \setminus C$ of elements that remain to be covered. Each stage of the greedy algorithm selects a set S to be added to the current cover C, and adds elements of S to C, in one batch. This batch of elements $e \in S \setminus C$ are all assigned the same $price(e) = \frac{c(S)}{|S \setminus C|}$. By definition, all the n elements of U can be covered with OPT cost by (i) the already chosen sets of the optimal solution C^* in the cover C, and (ii) the leftover sets of the optimal solution C^*. Observe that it is therefore also possible for the leftover sets of the optimal solution C^* to cover the elements of $U \setminus C$ with cost at most OPT. Let the leftover sets of C^* be T_1, T_2, ..., T_p; these are not used to cover the elements in C in the greedy algorithm so far. We can show that there is at least one set amongst these p sets with cost effectiveness at most $\frac{OPT}{|U \setminus C|}$. This follows from the fact that the 'average' *cost* of these $|U \setminus C|$ elements in the cover of total cost at most OPT (with the sets T_1, ..., T_p) is no more than $\frac{OPT}{|U \setminus C|}$. So, there must be at least one of these p sets with cost effectiveness at most $\frac{OPT}{|U \setminus C|}$. [A more detailed argument for this claim is included in Section 4.2.] So, the greedy heuristic chooses a set S with no more cost effectiveness (may not be one of these p sets) to be added to C, assigning *prices* to elements of the chosen set. If $e_k \in S$ then $price(e_k)$ is at most this cost effectiveness. If $k-1$ elements were in C before this set S was chosen, then $price(e_k) \leq \frac{OPT}{|U \setminus C|} = \frac{OPT}{n-k+1}$. For any subsequent element of S, $prices(e_l) = price(e_k) = \frac{OPT}{n-k+1} \leq \frac{OPT}{n-l+1}$, where $l > k$. Our algorithm (for the whole problem) will therefore greedily select some set covering the kth element with

$$price(e_k) \leq \frac{OPT}{|U \setminus C|} \leq \frac{OPT}{n-k+1}$$

4.2 A few details in the proof of the upper bound on price

Suppose we have already selected $k-1$ elements in the set C and assigned $price(e_i) \leq \frac{OPT}{n-i+1}$ for elements e_i, $1 \leq i \leq k-1$. While selecting e_k, two cases arise. If the same set introduces both e_{k-1} and e_k into the set C, then $price(e_k) = price(e_{k-1}) \leq \frac{OPT}{n-(k-1)+1} \leq \frac{OPT}{n-k+1}$ and we are done. Otherwise, a new set is necessary for inclusion of e_k in C; we analyze as follows.

As mentioned in Section 4.1, the p leftover sets of the optimal cover C^* are T_1, T_2, ..., T_p, ordered arbitrarily. The cost effectivity of T_i is $\frac{c(T_i)}{|T_i \setminus C|}$. Using elementary algebra (see Problem 3 in Tutorial 1), we can show that there must be at least one set amongst these p sets with cost effectivity no more than

$$\frac{\sum_{1 \leq i \leq p} c(T_i)}{\sum_{1 \leq i \leq p} |T_i \setminus C|}$$

The numerator is clearly upper bounded by OPT. The denominator is at least $n-k+1$ because $n-k+1$ elements are left, which can be to be covered (starting with e_k) by the p sets of the optimal cover C^*; the p sets can share elements outside the set C. So, the set selected for introducing e_k into the set C must have cost effectivity no more than

$$price(e_k) \leq \frac{OPT}{n-k+1}$$

as claimed.

5 Problem 14.5 (J. Cheriyan) from Vazirani's text

Problem 14.5 from Vazirani's text (J. Cheriyan): Design a polynomial time algorithm for the following problem.

"Given a graph G with non-negative vertex weights and a valid, though not necessarily optimal, coloring of G, find a vertex cover of weight $\leq (2-\frac{2}{k}) \cdot OPT$, where k is the number of colors used."

6 The maximun coverage problem

Given a universal set U of n elements, a collection of subsets of U, say $C = \{S_1, \ldots, S_l\}$, and an integer k, we need to pick k sets from the collection C so as to maximize the number of elements covered. In order to obtain an approximate solution for this maximization problem, we apply a greedy algorithm where we keep picking in each iteration the set from C that contains the largest number of elements yet to be covered, until k sets have already been picked. We analyze the approximation ratio of this algorithm below.

For the ease of presentation of the analysis, we establish the following notations:

- $U^* \subseteq U$ denotes the subset of elements in U that are covered by an optimal solution to the maximum coverage problem where $C^* \subseteq C$ is thebibliographycollection of sets selected in that optimal solution,
- x_i denotes the number of *new* elements from U being covered by the set chosen in the i-th iteration of the greedy algorithm
- y_i denotes the total number of elements from U that have been covered by all the sets chosen from C up to (and including) the i-th iteration of the greedy algorithm. Observe that $y_0 = 0$, and $y_i = \sum_{j=1}^{i} x_j$.

- z_i denotes the target deficit with respect to the optimal solution, the targeted number of more elements yet to be covered by all the sets chosen from C up to (and including) the i-th iteration of the greedy algorithm. Observe that, for each $i \in \{1,2,\ldots,k\}$, $z_i = |U^*| - y_i$, and $z_{i+1} = z_i - x_{i+1}$.

Recall that, at each step, our greedy algorithm selects a subset whose inclusion covers the maximum number of uncovered elements. We also know that the optimal solution uses k sets to cover $|U^*|$ elements, where C^* is the optimal collection of k sets from C that cover the OPT elements of U in U^* (as already mentioned above). By the time we have selected i sets in our algorithm, we might have selected some sets from C^* or (may be) no set from C^*. Certainly, we would not have selected all sets from C^* already because in that case we would have covered $|U^*|$ elements already. So, some collection C' of the at most k sets of C^* remain unselected so far where we still have $|U^*| - y_i = z_i$ elements to be covered. Out of the sets in C' there must be a set that covers at least $\frac{z_i}{k}$ of the so far uncovered elements which C^* covers. So, our algorithm must pick such set from C' or a set from outside C', in the $(i+1)st$ selection where

$$x_{i+1} \geq \frac{z_i}{k}$$

Claim 1. *For each $i \in \{1,2,\ldots,k\}$, we have $z_i \leq (1-\frac{1}{k})^i \cdot |U^*|$.*

Proof We prove the above claim by induction. Observe that $z_0 = |U^*|$, since $y_0 = 0$.
So, for the base case of $i = 0$, we have $y_1 = x_1 \geq \frac{z_0}{k} = \frac{|U^*|}{k}$, which implies $z_1 \leq |U*| - y_1 \leq (1-\frac{1}{k}) \cdot |U^*|$.
Now let us assume inductively that $z_j \leq (1-\frac{1}{k})^j \cdot |U^*|$.Then, we have:

$$z_{j+1} = z_j - x_{j+1} \leq z_j - \frac{z_j}{k} \leq (1-\frac{1}{k}) \cdot z_j \leq (1-\frac{1}{k})^{j+1} \cdot |U^*|$$

Thus, the cost of the solution from our greedy algorithm is given by:

$$y_k = |U^*| - z_k \geq |U^*| - (1-\frac{1}{k})^k \cdot |U^*| \geq (1-(1-\frac{1}{k})^k) \cdot |U^*| \geq (1-\frac{1}{e}) \cdot |U^*|$$

7 Improvement of the approximation guarantee

A natural question about improving the approximation guarantee is whether a better analysis of the algorithm being considered, can improve the approximation guarantee any further. Essentially, we must show that the analysis already provided for the algorithm is tight.

7.1 Tight example for the vertex cover algorithm

In the case of the factor two algorithm using maximal matchings for vertex covering (in Section 2), we note that on $K_{n,n}$ the algorithm produces a solution that is twice the optimal in cardinality. Since $K_{n,n}$ is the complete bipartite graph on $2n$ vertices with n^2 edges, our algorithm would certainly choose a matching of size n, and therefore a vertex cover of size $2n$, thereby showing that we cannot get a factor better than 2 for such graphs for any integer n (see [8]), for this algorithm.

[This is despite the fact that the lower bound of the size of a maximal matching is n as well as the size of an optimal vertex cover is n.] So, this family of infinite graphs provides what we call a *tight (asymptotic) example* for the specific algorithm. Tight examples often give critical insight into the functioning of an algorithm and often lead to ideas for the design of other algorithms that can achieve improved guarantees.

7.2 Maximal matchings lower bound cannot yield better approximation guarantees for vertex covering

Now consider another question: can a better approximation algorithm be designed that achieves a better guarantee but still uses the the same lower bounding scheme as our current algorithm of Section 2. For addressing this second question, consider the complete graph K_n of n vertices where n is an odd integer. Note that it has a minimum vertex cover of size $n-1$; dropping any two vertices would leave an edge uncovered. Also, n being odd, we observe the maximum matching has cardinality $\frac{n-1}{2}$. For this example, no algorithm can achieve a ratio factor of approximation better than 2 for any odd integer n (see [8]).

[*So, we observe that by simply using the lower bounding scheme of maximal matchings, we cannot improve the approximation ratio.*]

7.3 Total weight of all edges cannot yield better approximation guarantees for weighted cut

As in the case of vertex cover in Section 2, we can also make a similar observation for the *maximum weighted cut* problem. A polynomial time algorithm exists that ensures a cut of weighted capacity at least $\frac{1}{2}w(E)$, where $w(E)$ is the sum of weights of the edges. We now show here that for all n, we cannot have a better ratio factor for graphs K_{2n}, if we use the (obvious) upper bound of $w(E)$ for the maximum cut. The graph K_{2n} has exactly $n(2n-1)$ edges and a maximum cut of size n^2, giving an approximation ratio at most $\frac{1}{2}$ for such graphs for all integers n. [Observe that the cut is maximised when it separates any set of n vertices form the rest of the n vertices.]

[*So, we observe that by simply using the upper bounding scheme provided by $w(E)$, we cannot improve the approximation ratio.*]

7.4 Tight example for the greedy weighted set cover algorithm

Suppose, n sets each have a singleton element and the set weights are respectively, $\frac{1}{n}, \frac{1}{n-1}, \cdots, 1$, and the last set has all these n elements with set weight $1+\varepsilon$. The optimal cover has weight $1+\varepsilon$ and the algorithm in Section 4 computes a set cover with weight $H(n)$. In each iteration, the cost effectiveness of the last set is higher than those of the other sets. This is an example where the $H(n)$ upper bound is approached as ε approaches zero.

In Section 13.1 of [8] we can see the Example 13.4 which reveals that the H_n bound is essentially tight, irrespective of the algorithm used. For this purpose, we must see the LP relaxation 13.2 on page 109 of [8], as discussed here in Section 9.4.

See Sections 29.7 and 29.9 of [8] for seeing why the obvious greedy algorithm is the best one can hope for.

8 Rounding linear programs for designing approximation algorithms

We can develop approximation algorithms based on *linear programming*. The example to begin with is the *weighted version* of the *vertex cover* problem on *(vertex) weighted undirected graphs* as illustrated in [4]. A *linear program* has a system $Ax \geq b$ of inequalities called *constraints*, and an *objective* function $c^T x$. We need to minimize $c^T x$ over all *positive vectors* $x \geq 0$, satisfying the given set of constraints. The set of constraints represents the intersection of half-spaces, which is a convex region of multi-dimensional space. It therefore represents a convex region called the *feasible* region. Optima of linear objective functions like $c^T x$ can occur only at vertices of this convex feasible region. However, the number of vertices can grow exponentially in the number of inequalities. So, we need efficient algorithms for linear programming that run in polynomial time. Being more precise, the problem we define is as follows.

Given an $m \times n$ matrix A, and vectors $b \in \mathcal{R}^m$ and $c \in \mathcal{R}^n$, find a vector $x \in \mathcal{R}^n$ solving the optimization problem $min\{c^T x$ such that $x \geq 0$ and $Ax \geq b\}$.

For the *weigthed vertex cover* problem each vertex i has a positive weight w_i, we say that the weight of a set of vertices is the sum of weights of its vertices. We wish to show that a vertex cover with at most twice the weight of the *optimal (minimum) weighted vertex cover* can be computed in polynomial time. We use an *indicator* or *decision* variable x_i for inclusion of the ith vertex in the vertex cover. Such an *integral* variable can take values 1 or 0, based on whether it is present or absent in the vertex cover. The minimum weighted vertex cover will minimize

$$\sum_{i \in V} w_i x_i$$

such that

$$x_i + x_j \geq 1, (i,j) \in E$$

and

$$x_i \in \{0,1\}, i \in V$$

We can rewrite the problem formally, relaxing the integrality restriction on x_i, as

$$Ax \geq 1$$

$$1 \geq x \geq 0$$

where the integer 0-1 matrix A has one row for each edge and one column for each vertex and $A[e,i] = 1$ whenever vertex i is in edge e and 0, otherwise. We need to solve the optimization problem $\min\{w^T x$ such that $1 \geq x \geq 0\}$ and $Ax \geq 1$. Observe that we have reduced the *discrete (integral) optimization version* of the minimum weighted vertex cover problem to the *linear programming problem* where the solutions (for x_i) can be *real rational numbers*. [The discrete optimization problem for minimum weighted vertex cover is called the *0-1 integer programming* problem. Why is the *decision version* of this 0-1 integer programming problem also in the class NP? Is it also NP-complete?]

Let w_{LP} be the optimal weight for the relaxed linear programming optimization problem where the (optimal) solution for the variables x_i can be rationals. For such an optimal solution vector

x^*, the components x_i^* may very well be non-integral. So, we cannot directly get a solution to the integer programming problem stated above, which was the actual problem to be solved.

One way to get an integer solution is to *round off* the fractional solutions to 0's and 1's. Note that the fractional solutions are in the range [0,1], as we can see in the formulation of the constraints. We include i in S if and only if $x_i^* \geq \frac{1}{2}$, for all $1 \leq i \leq n$. This way we indeed get an (approximate) vertex cover, whose total weight will now be shown to be at most twice that of the optimal weighted vertex cover.

Why is this set S a vertex cover? Each edge of the graph is covered by at least one vertex in S because the linear program solution satisfies the constraint for each edge; this is because at least one vertex in each equation must be having its fractional solution equal to or greater than $\frac{1}{2}$. We thus make sure that we select at least one vertex from each edge in the set S, a vertex which has fractional solution equal to or more than $\frac{1}{2}$.

First observe that $w_{LP} \leq w(S^*)$, where S^* is any optimal weighted vertex cover. This is because the optimal vertex cover is a special case where the solutions are integral and relaxing this restriction cannot worsen the solution. Also, $w(S^*) \geq w_{LP} = w^T x^* = \sum_i w_i x_i^* \geq \sum_{i \in S} w_i x_i^* \geq \frac{1}{2} \sum_{i \in S} w_i = \frac{1}{2} w(S)$. So, we have $w(S) \leq 2 w_{LP} \leq 2 w(S^*)$.

9 Linear programming duality and analysis of greedy approximation algorithms

The linear program with *m linear inequalities* representing *constraints* for minimizing a *linear objective function* for an *n*-dimensional *non-negative vector* is as follows.

$$minimize \sum_{j=1}^{n} c_j x_j [c^T x]$$

$$given \sum_{j=1}^{n} a_{ij} x_j \geq b_i, i = 1, \cdots, m [Ax \geq b]$$

$$x_j \geq 0, j = 1, \cdots, n [x \geq 0]$$

where a_{ij}, b_i, c_j are given rational numbers. Here, A is an $m \times n$ matrix, b is an $m \times 1$ matrix, and x and c are an $n \times 1$ matrices. Note that b is a lower bound on Ax, whereas we cannot indefinitely inflate x since we wish to minimize $c^T x$.

The optimization (minimization) problem yields an optimal solution x^*. If we wish to address the question of membership in P or NP, it helps to formulate decision versions of the *linear programming* problem. Instead of computing x^*, we may ask whether $z^* = c^T x^*$ is at most α, where α is a real number. [Note that we do not know z^* when we are given the decision version instance, denoted by matrices A, b, c and α. Nevertheless, we pose the decision version question "whether $z^* \leq \alpha$".] In other words, we may ask whether the optimal value of the objective function is upper bounded by some not too large constant. Suppose we have a 'yes' instance. Then, we are assured that indeed $z^* \leq \alpha$. In that case, there must be some $x = a > 0$ that satisfies $Aa \geq b$, and $z^* \leq c^T a = d \leq \alpha$. Why? Such an a is a feasible (possibly non-optimal) solution which is a 'witness' that this is a 'yes' instance. The moment we know such a 'witness' a, we set $d = c^T a$, and we

can easily check whether $Aa \geq b$ and $c^T a \leq \alpha$, confirming and verifying that z^* is also at most α, that is, $z^* \leq c^T a = d \leq \alpha$. In other words, we can *verify* efficiently that α is indeed an upper bound on z^*, even though we do not know z^*, simply by checking a 'witness' for the given 'yes' instance. This means that the decision problem at hand is indeed in the class *NP*. We simply can check efficiently for a 'yes' instance given such a certificate a, that the instance is indeed a 'yes' instance. This is however applicable only if the witness is 'succinct'. The witness a must be bounded in description length by a polynomial in the size of the input to the decision problem. How we do establish the existence of such 'succinct' witnesses? In the classical text by Hopcroft and Ullman [3], we find the elaborate proof of the fact that feasible solutions of polynomial description lengths can indeed be shown to exist. The techniques used are from linear algebra, including Cramer's rule.

Is this decision question also in the class co-NP? We will soon answer this question after we define what is known as the *dual* problem of a given (*primal*) linear program.

The *dual* linear program for the *primal* program given above is

$$maxmize \sum_{i=1}^{m} b_i y_i [b^T y]$$

$$given \sum_{i=1}^{m} a_{ij} y_i \leq c_j, j = 1, \cdots, n [A^T y \leq c]$$

$$y_i \geq 0, i = 1, \cdots, m [y \geq 0]$$

Here, the lower bounds b_i in the constraints in the primal program define the objective function for maximization in the dual program. Symmetrically, the upper bounds in the constraints of the dual program define the objective function in the primal program. Observe further that the 'variables' in $y \geq 0$, in the dual linear program are multipliers of the lower bounds in b of the primal linear program. Even though we maximize the objective function in the dual, which is the dot or inner product of b with the weight- or price- of the variables- the vector y, we are well guarded by the upper bounds in $c \geq A^T y$. Suppose we ensure that the coefficients of each primal variable x_i (in all the m inequalities of the primal), when weighted by the m multipliers or variables in y of the dual, do not exceed the corresponding cost c_i of the primal. This ensures that the objective function value in the dual is always below that in the primal, for any pair of feasible solution pairs x and y of the primal and dual, respectively. With this intuition, we now proceed to formally establish the 'weak duality' result below.

9.1 Weak duality

For feasible solutions x and y to the primal and dual programs respectively, we know that the following inequality holds.

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} b_i y_i [c^T x \geq b^T y]$$

This *Weak LP-Duality result* is easily establsihed; we replace upper bounds c_j from the inequalities of the dual problem getting

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{j=1}^{n} (\sum_{i=1}^{m} a_{ij} y_i) x_j [c^T x \geq x^T A^T y]$$

and symmetrically replace the lower bounds b_i from the inequalities in the primal problem getting

$$\sum_{i=1}^{m}(\sum_{j=1}^{n} a_{ij}x_j)y_i \geq \sum_{i=1}^{m} b_i y_i [y^T Ax \geq b^T y]$$

finally observing that

$$x^T A^T y = y^T Ax$$

9.2 Optimality and complementary slackness

It is known that the *feasible* solutions x^* and y^* for the primal and dual respectively, are both *optimal* if and only if

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^* [c^T x^* = b^T y^*]$$

Now we state the complementary slackness conditions as

1. For each $1 \leq j \leq n$ either $x_j^* = 0$ or $\sum_{i=1}^{m} a_{ij} y_i^* = c_j$
2. For each $1 \leq i \leq m$ either $y_i^* = 0$ or $\sum_{j=1}^{n} a_{ij} x_j^* = b_i$

These two conditions hold if and only if x^* and y^* are respective optima in the primal and dual problems. Subsequently, in Section 7.7 we consider *relaxed complementary slackness conditions* for non-optimal integral primal solutions and correspnding fractional dual solutions for the primal-dual method of designing approximation algorithms.

9.3 Membership in the class co-NP

We are now in a position to show that linear programming belongs to the class co-NP. The question we now ask given a linear programming primal instance A, b, c and α is whether $z^* \geq \alpha$, that is whether z^* is at least α. This question being complementary to the question in the original problem, establishing the membership in *NP* for this question would place the original problem in the class co-NP, as much as we have already shown that the original problem is in the class NP. This is easy to show using a similar argument applied to suitable feasible solutions of the dual linear program that have lower bounded objective function values; using such solutions of the dual as 'succinct' 'certificates' or 'witnesses', 'yes' instances of this new problem can be shown to be checkable in polynomial time for feasibility and for being above the lower bound α, thereby verifying that indeed $z^* \geq \alpha$ and therefore checking that the instance is indeed a 'yes' instance.

9.4 The dual fitting technique for the greedy algorithm of Section 4 for the weighted set covering problem

Now we revisit the *weighted set cover* problem and use a linear programing *dual fitting* approach, in order to analyze the same asymptotic approximation ratio bound of $H(n)$. This exposition is based on Chapter 13, pages 108-113 of [8]. We show the existence of a feasible dual solution y, that can be used to get an upper bound on the cost of the greedy set cover.

9.4.1 The integer program, its relaxation LP and the dual LP

The problem of minimum set cover is as follows.

$$\textit{minimize} \sum_{S \in \mathcal{S}} c(S) x_S$$

subject to

$$\sum_{S: e \in S} x_S \geq 1, e \in U$$

$$x_S \in \{0,1\}, S \in \mathcal{S}$$

This is a 0-1 integer program.

The *LP-relaxation* of this integer program is the following *primal* linear program.

$$\textit{minimize} \sum_{S \in \mathcal{S}} c(S) x_S$$

subject to

$$\sum_{S: e \in S} x_S \geq 1, e \in U$$

$$x_S \geq 0, S \in \mathcal{S}$$

In this formulation of the relaxation LP, we have one constraint for each element $e \in U$ and one term in the objective function for each set. One very important point to note here is that the variables x_S are only required to be non-negative. We do not need any restriction/constraint upperbounding x_S because all these variables are non-negative, the costs $c(S)$ are non-negative and we have an objective function which is the dot product of such two non-negative component vectors. The constraint $x_S \leq 1$ does not change the optimum, because a solution in which some x_S are bigger than 1 can be converted to a solution in which all variables are at most 1 while decreasing the objective function, and so no variable is larger than 1 in an optimal solution, even if we do not have the (explicit) constraint $x_S \leq 1$.

[Further, note that such a primal relaxation is also a *covering linear program*. We say that min $c^T.x$, subject to $Ax \geq b$, where $x \geq 0$, and all entries of A, b, c are non-negative, is called a *covering linear program*.]

[An *LP-relaxation* of an integer linear program must satisfy two conditions: (i) every feasible solution for the original integer program must be a feasible solution for the linear program, and (ii) the value of any feaible solution of the integer program must have the same value in the linear program.]

The *dual* linear program has one term y_e for each element in the objective function and one constraint for each set S, as follows.

$$\textit{maximize} \sum_{e \in U} y_e$$

subject to

$$\sum_{e: e \in S} y_e \leq c(S), S \in \mathcal{S}$$

$$y_e \geq 0, e \in U$$

We know that the optimal cost OPT of the set cover is at least the optimal cost OPT_f of the primal linear program in the LP relaxation. We also know that the cost of any feasible solution to the dual linear program is no more than OPT_f, which in turn is no more than OPT. [The optimal costs of the primal and dual linear programs are both OPT_f.]

9.4.2 The greedy algorithm

When we choose the next ist element $e_i \in S = \{e_1, e_2, \cdots, e_k\}$ of the k elements of a set S in the greedy set cover heuristic, the $price(e_i)$ is no more than $\frac{c(S)}{k-i+1}$, as we now demonstrate. [This upper bound is a straightforward ratio, unlike the analysis we did earlier in Section 4.] The main argument is that the element e_i may be incorporated due to the inclusion of either the set S itself or some other set. If S is itself chosen then there are $k-i+1$ new elements $e_i, ..., e_k$ to be included with cost effectivity $\frac{c(S)}{k-i+1}$, the assigned value of $price(e_j)$, $i \leq j \leq k$. Clearly, $price(e_j) = \frac{c(S)}{k-i+1} \leq \frac{c(S)}{k-j+1}$, $i \leq j \leq k$. If not the set S but some other set includes e_i with cost effectivity no more than $\frac{c(S)}{k-i+1}$, as per the greedy algorithm, then we again have $price(e_i) \leq \frac{c(S)}{k-i+1}$.

9.4.3 Scaling for dual fitting

Now setting the variable y_e of the dual linear program for each $e \in U$ to $\frac{price(e)}{H(n)}$, where $H(n) = \sum_{i=1}^{n} \frac{1}{i}$, we observe that

$$y_{e_i} \leq \frac{1}{H(n)} . \frac{c(S)}{k-i+1}$$

for each of the k elements $e_i \in S$. So,

$$\sum_{i=1}^{k} y_{e_i} \leq \frac{c(S)}{H(n)} . (\frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{1}) = \frac{H(k)}{H(n)} . c(S) \leq c(S)$$

So, the constraints in the dual linear program are satisfied establishing the feasibility of the solution with y_e values as assigned above. Now we further observe that

$$\sum_{e \in U} price(e) = H(n)(\sum_{e \in U} y_e) \leq H(n).OPT_f \leq H(n).OPT$$

Recall here that the costs of the sets selected in the set cover are distributed over the elements of U as $price(e)$; all new elements e covered when a certain set S is picked up by the greedy algorithm are assigned the cost-effectivity value $price(e)$ of the seleted set S. Therefore, $\sum_{e \in U} price(e)$ is the sum of costs of all selected sets, which as seen above is at most $H(n)$ times OPT.

9.4.4 The notion of prices for the primal integral solution being fully and payed up by the dual solution

Observe that using the linear programming relaxation of the integer program and the dual LP of the (primal) LP relaxation, we saw how the (primal) integral solution computed by the algorithm is *fully paid for* by the computed dual variables. The objective function value of the primal integral

solution is matched by the objective function value of the dual variables computed. However, further in the analysis, we divide the dual variables by a suitable factor and show that the scaled down dual solution is feasible. The scaling factor is the approximation guarantee of the algorithm since the dual gives a lower bound on the optimal value of the linear primal and dual linear programs, thereby giving a lower bound on also the optimal objective function value of the integer linear program.

Indeed, the greedy algorithm defines dual variable values $price(e)$, for each element e. Observe that the cost of the selected sets in the set cover picked by the algorithm is *fully payed* for (in this case exactly equalled) by the dual solution. However, this dual solution is not feasible. We therefore needed to shrink the values by a factor of $H(n)$, so that they fit into the given set cover instance, i.e., no set is *overpacked* (in other words, all constraints of the dual LP are satisfied).

9.5 Dual fitting for the constrained set multicover problem

The discussion here on the *constrained set multicover* problem is from Section 13.2.1 in [8]. Here, each element e needs to be covered a specific integer number r_e of times. We also use the constraint that each set can be picked up at most once. [A set S if picked up k times yields cost $kc(S)$. Such permissible picking of a set multiple times is allowed in the less constrained problem *set multicover*.]

9.5.1 Integer program and the primal relaxation LP for the constrained problem

Now we propose the integer programming formulation as min $\sum_{S \in \mathcal{S}} c(S)x_S$ subject to $\sum_{S:e \in S} x_S \geq r_e$, for all $e \in U$, given that $x_S \in \{0,1\}$, for all $S \in \mathcal{S}$. Here, $r_e \in Z^+$. The LP-relaxation is tricky because we must now constrain each set to be selected at most once. So, we need to realize the constraint $x_S \leq 1$ as well. Therefore, replacing the integer program constraint on x_S taking on values 0 and 1 only, we now use the following constraints in the LP-relaxation: $-x_S \geq -1$ and $x_S \geq 0$, for all $S \in \mathcal{S}$.

9.5.2 The dual LP for the primal LP relaxation of the integer program

The dual linear program for the LP-relaxation is therefore complex, with a few more variables because we do not have a primal covering linear program (there are some negative elements in the matrices and vectors in the primal-dual linear program formulation). The additional constraints have new variables z_S in the dual. The dual LP is no more a packing program. The primal LP has one constraint for each element in U as well as a constraint for each set in $\mathcal{S}$; there are as many variables in the dual LP, the y_e variables as well as the z_S variables. Now, a set S can be overpacked with the y_e variables. This can be done only provided we raise z_S to ensure feasibility. The objective function value can then decrease. However, overall, overpacking may still be advantageous, since the y_e appear with coefficients of r_e in the objective function.

max $\sum_{e \in U} r_e y_e - \sum_{S \in \mathcal{S}} z_S$
subject to
$(\sum_{e:e \in S} y_e) - z_S \leq c(S)$, for all $S \in S$
$y_e \geq 0$, for all $e \in U$
$z_S \geq 0$, for all $S \in \mathcal{S}$

9.5.3 The greedy set cover algorithm for choosing alive elements repeatedly

The greedy algorithm is as follows. We say that element e is *alive* if it occurs in less than r_e of the sets already selected. The algorithm next picks an unpicked set which is the most cost-effective set; the *cost-effectiveness* of a set is defined as the average cost at which the set covers its currently alive elements. The algorithm halts when there are no more alive elements.

9.5.4 Multiple prices for elements in different selections

On picking a set S, its cost $c(S)$ is distributed equally amongst the alive elements it covers. If S covers e for the jth time, $price(e, j)$ is set to the current cost-effectiveness of S as defined above. [It is easy to see that the cost-effectiveness of sets picked is nondecreasing.] Since cost-effectiveness is non-decreasing over iterations of selection of sets in the set cover, we have, for each element e, $price(e,1) \leq price(e,2)... \leq price(e,r_e)$.

9.5.5 The dual solution

The variables of the dual are set as follows at the end of the algorithm's execution. For each $e \in U$, we set (after scaling by H_n) $y(e) = \frac{\alpha_e}{H_n} = \frac{1}{H_n}.price(e,r_e)$. For each $S \in \mathcal{S}$ picked up by the algorithm in the set cover, we set (after scaling down by H_n) $z_S = \frac{\beta_S}{H_n} = \frac{1}{H_n}.[\sum_{e-covered-by-S}(price(e,r_e) - price(e,j_e))]$, where j_e is the copy of e covered by S. Note that since $price(e,j_e) \leq price(e,r_e)$, so β_S is non-negative. If S is not picked by the algorithm, then β_S is defined to be 0.

Now observe that the objective value of the primal is $\sum_{e \in U} \sum_{j=1}^{r_e} price(e,j)$. Indeed, this is identical to the objective function value of the dual variables (α, β) since $\sum_{e \in U} r_e \alpha_e - \sum_{S \in \mathcal{S}} \beta_S = \sum_{e \in U} \sum_{j=1}^{r_e} price(e,j)$. After scaling down (α, β) by a factor of H_n we get the *scaled* dual LP feasible solution (y,z), where $y_e = \frac{\alpha_e}{H_n}$ and $z_S = \frac{\beta_S}{H_n}$.

9.5.6 Scaling and dual-fitting for satisfying the dual constraints

For ascertaining that the (y,z) solution is a scaled but feasible dual solution, we need to look at each set S. Consider a set $S \in \mathcal{S}$ consisting of k elements. Order and enumerate its elements in the order in which their r_e occurance requirements were fulfilled. This is the order in which they stopped being alive. Let the ordered elements be $e_1, ..., e_k$. Suppose S is not picked by the algorithm. When the algorithm is about to cover the last copy of e_i, S contains at least $k-i+1$ alive elements, so $price(e_i, r_{e_i}) \leq \frac{c(S)}{k-i+1}$. Since z_S is zero, we get $\sum_{i=1}^{k} y_{e_i} - z_S = \frac{1}{H_n} \sum_{i=1}^{k} price(e_i, r_{e_i})$ $\leq \frac{c(S)}{H_n}.(\frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{1}) \leq c(S)$.

Next, we assume that S is picked by the algorithm. Also assume that just before S is picked up $k' \geq 0$ elements of S are already completely covered. Then, $(\sum_{i=1}^{k} y_{e_i}) - z_S = \frac{1}{H_n}[\sum_{i=1}^{k} price(e_i, r_{e_i}) - \sum_{i=k'+1}^{k}(price(e_i, r_{e_i}) - price(e_i, j_i))]=$

$\frac{1}{H_n}[\sum_{i=1}^{k'} price(e_i, r_{e_i}) + \sum_{i=k'+1}^{k} price(e_i, j_i)]$, where S covers the j_ith copy of e_i, for each $i \in \{k'+1, ..., k\}$. But $\sum_{i=k'+1}^{k} price(e_i, j_i) = c(S)$, since the cost of S is equally distributed among the copies it covers. Finally consider elements e_i, $i \in \{1, ..., k'\}$. When the last copy of e_i is being covered, S is not yet picked and covers at least $k-i+1$ alive elements. Thus, $price(e_i, r_{e_i}) \leq \frac{c(S)}{k-i+1}$. Therefore, $(\sum_{i=1}^{k} y_{e_i}) - z_S \leq \frac{c(S)}{H_n}(\frac{1}{k} + \cdots + \frac{1}{k-k'+1} + 1) \leq c(S))$.

9.5.7 The final analysis of the factor H_k ratio bound

The actual aprroximation ratio is as good as H_k, where k is the cardinaity of the largest set in $\mathcal{S}$. This fact is easily seen in the derivations above.

9.6 Primal-dual analysis for the edge-charging algorithm for the weighted vertex covering problem

In this section we again use the Primal-Dual technique where an (alternative) algorithm is used for *edge-based charging* as in the Exercise 2.11 on page 24 of [8]. Observe that the factor two vertex covering algorithm using maximal matching computation (as presented in the Section 2) was based on a lower bound on the size of any vertex cover provided by the size of any maximal matching. Here we use lower bounding in a differet way by working on LP dual variables; the primal LP uses variables $x(v)$ for vertices v and the dual LP uses varibles $y(e)$ for edges e. An edge e is written as (u,v), or better as $\{u,v\}$ where u,v are vertices. Vertices are assigned weights $w(v)$ as in the input. The primal objective function is $\sum_{v\in V} w(v)x(v)$, which is to be minimized s.t. $x(u)+x(v)\geq 1$, for all $\{u,v\}\in E$. The integer programming version is when $x(v)$ is 0 or 1 for all $v\in V$. The primal LP relaxation says that $x(v)\geq 0$, for all $v\in V$. The dual LP maximizes $\sum_{e\in E} y(e)$ s.t. $\sum_{e=(u,v),e\in E} y(e)\leq w(v)$, for all $v\in V$. Here $y(e)\geq 0$, for all $e\in E$. The constraint for each vertex in the dual LP for each vertex v says that the prices $y(e)$ of all edges $e=(u,v)$, for vertex v add up to at most $w(v)$.

Supppose some algorithm computes a vertex cover C with *prices* $y(e)$ for edges $e\in E$. Then, the sum of these prices over edges incident on a vertex v would add up to at most $w(v)$ if the $y(e)$ values for all $e\in E$ constitute a feasible solution for the dual LP. However, suppose we could algorithmically assign prices to edges $e\in E$ s.t. for every $v\in C$, we have $\sum_{e=(u,v)\in E} y(e)=w(v)$. So, the price of an edge incident on v, say $e=(u,v)$ must be such that $y(e)\leq w(v)$ and $\sum_{e=(u,v)\in E} y(e)=w(v)$. We can do this by selecting an edge $e=(u,v)$ and subtracting the smaller of (remaining) minimum of weights of u and v from both u and v. Then if u (or v) become zero in weights, we force u (or v) into C. [If they simultaneously become zero then naturally both are included in C.] The algorithm proposed is thus as follows. Set C to an empty set and $t(v)$ to $w(v)$ for each $v\in V$. These $t(v)$ are varying (reducing) weights of vertices; picking an edge $e=(u,v)$, we subtract the minimum of $t(v)$ and $t(u)$ from each of them. For u or v (or both) we select u (v) in C if $t(u)$ ($t(v)$) is zero after subtraction. Since each time an edge of v is processed, we deduct $t(v)$ till we select v when $t(v)$ becomes zero, the prices of edges $e=(u,v)$ where $v\in C$, are indeed summed up to $w(v)$.

Summing up weights of all vertices in C we have $\sum_{v\in C} w(v)=\sum_{v\in C}\sum_{e=(u,v)} y(e)=$
$\sum_{e=(u,v),e\in E}\sum_{w\in e=(u,v),w\in C} y(e)\leq \sum_{e=(u,v)\in E} 2.y(e)$.

We know that $OPT=\sum_{v\in C^*} w(v)\geq \sum_{e\in E} y(e)$, since the dual LP objective function value can never exceed the cost OPT of the optimal weighted vertex cover, which is no lesser than OPT^*, the optimal objective function value shared by both the (fractional) primal and dual LPs. The 2-approximation result follows.

9.7 Improvements beyond the factor 2 for vertex covering

Strictly less than factor 2 approximation ratio can be achieved for the vertex covering problem (VC) when certain conditions are enforced. Firstly, it is possible to show that the primal LP is

half-integral. More particularly and precisely, it is possible to show that extreme point solutions to the primal LP are half-integral (see Lemma 14.4 and Theorem 14.5 of [8]).

A half-integral solution x^* to the primal VC LP may not be an optimal solution for VC. However, it is known due to Neemhauser and Trotter (see [9]), that there is some half-integral solution x^* for the primal VC which agrees in all its integral components with an optimal (integer) solution for VC. Guessing the optimal solution by rounding such a half-integral solution may not be possible in polynomial time in the number of variables. However, it is possible to approximate the VC problem with approximation ratio 2 by working non-trivially on a subgraph induced by a subset of the set of vertices of the graph, as follows.

9.7.1 Partitioning graph G into vertex sets P, Q and R

We partition the set V of vertices into sets P, Q and R, corresponding to vertices i, where x_i^* is 1, $\frac{1}{2}$, or 0, respectively, and where x^* is an optimal half-integral solution for VC; this optimal half-integral solution for VC can in turn be computed in polynomial time, as we show below, starting from any half-integral solution for VC.

9.7.2 Optimal half-integral solution from any half-integral solution

We develop the various ideas and partial results as follows. We show that given a half-integral solution HI to VC, there exists an optimal LP solution $OPT \subseteq V$ for VC such that OPT comprises of all the 1-components of HI and a subset of the $\frac{1}{2}$-components of HI.

First we show that all 1-components lie in OPT. .... Then we show that the other elements of OPT are from the $\frac{1}{2}$-components. ....

Next we will have to find out the OPT solution given the initial half-integral solution. ....

9.7.3 Using combinatorial methods and algorithms like network flow computations for computing optimal LP solutions

. . .

.

Once the OPT solution is obtained, we can then call the three sets with 1, 1/2 and 0 values as P, Q, R, and proceed as follows.

9.7.4 The vertex cover and the lower bound

We can consider $P \cup Q$ as a vertex cover. Why? The weight of this cover is $w(P) + w(Q)$. The optimal LP solution objective funcion value of $w(P) + \frac{1}{2}w(Q) \leq OPT$ leads to a 2-factor approximation ratio. A smaller vertex cover results by dropping an independent set S from Q with weight at least $\frac{w(Q)}{k}$, which we can get from a proper k-coloring of Q; note that $R \cup S$ is an independent set and $P \cup Q \setminus S$ is a vertex cover. So we have a vertex cover with weight $w(P) + \frac{k-1}{k}w(Q) \leq 2(\frac{k-1}{k})(w(P) + \frac{w(Q)}{2}) \leq (2 - \frac{2}{k})OPT$.

9.8 The generic primal-dual scheme for covering-packing programs written in the standard form

As in Section 15.1 of [8], we focus on the standard form primal and dual LPs, and define *relaxed complementary slackness* conditions with parameters α and β, leading to the crucial Proposition 15.1 of [8]. The design of Algorithm 15.2 is based on Proposition 15.1 leading to Theorem 15.3.

Algorithm 15.2 starts with a primal *infeasible* solution and a dual feasible solution; these are usually the trivial solutions $x = 0$ and $y = 0$. It iteratively *improves the feasibility of the primal solution*, and the *optimality of the dual solution*, ensuring that in the end *a primal feasible* solution is obtained and all conditions, with a suitable choice of α and β, are satisfied. The primal solution is *always extended integrally*, thus ensuring that the final solution is integral. The current primal solution is used to determine the improvement to the dual, and vice versa. Finally, the cost of the dual solution is used as a lower bound on OPT, and by Proposition 15.1, the approximation guarantee of the algorithm is $\alpha\beta$.

[For exercise 12.4 in the print version of [8] (absent in the e-version), we use the paying mechanism for showing the equality of the objective function values for the primal and dual LPs provided the solutions for the primal and dual LPs obey the complememtary slackness conditions.]

To prove Proposition 15.1, we show that given a sufficient amount of balance, the dual can pay enough from this balance through dual variables y_i for the primal variabes x_j, so that the total payment done by the y_i's, and collected by the x_j's, is sufficient to meet the objective function cost of the primal solution. The upper bound (balance) for the total payment by the y_i's is the r.h.s. of Proposition 15.1 and the collection made by the primal x_j's is at least the lower bound l.h.s. of Proposition 15.1. The details as are follows.

The payment from y_i to x_j is $\alpha y_i a_{ij} x_j$. The total payment to all the primal variables from y_i is therefore $\alpha y_i \sum_{j=1}^{n} a_{ij} x_j \leq \alpha\beta b_i y_i$ (due to the upper bounds in the relaxed dual complementary slackness conditions). So, the total payment from all dual variables to all primal variables is at most the balance r.h.s. of Proposition 15.1. The total collection in x_j is $\alpha x_j \sum_{i=1}^{m} a_{ij} y_i \geq c_j x_j$ (due to the lower bounds in the relaxed primal complementary slackness conditions.) So, the total payment is at least the l.h.s of Proposition 15.1.

10 The travelling salesman problem

In the *metric TSP* problem (see Section 3.2 of [8]), the approximation ratio is two, as we argue now. Note that the input graph is a complete graph $G(V,E)$ with n vertices, with non-negative edge weights, where the edge weights satisfy the *triangle inequality*.

10.1 The lower bound for the cost of the minimum cost tour

The minimum tour has cost at least that of an MST. Why? We can get a spanning tree T from a minimum cost tour by dropping any edge of the tour. The MST has cost m, where m is at most the cost of T. So, an MST has cost $m \leq OPT$, where OPT is the cost of the minimum metric TSP tour over $G(V,E)$. This lower bound m (of the MST cost) for OPT (the cost of the minimum metric TSP tour), is used in establishing the approximation ratio bound of two.

10.2 Computing the 2-approximate tour

For computing a tour that is at most twice the cost of the minimum cost metric TSP tour, we *double* an MST, by duplicating each of the $n-1$ weighted edges of the MST. Clearly, in the resulting doubled (multi-)graph G', all vertices have even degree, thereby admitting an Euler tour, say $\mathcal{T}$ of exactly $2n-2$ edges. From the Euler tour $\mathcal{T}$, a Hamiltonian cycle C is finally constructed by visiting the n vertices in the order of their first appearance in $\mathcal{T}$ in a DFS traversal on the Euler tour T. To skip the repetition of vertices (as we convert the Euler tour into a Hamiltonian tour), we add "short-circuited edges" from the set E of $G(V,E)$. Clearly, by the triangle inequality, such short-circuting does not increase the cost; the resulting Hamiltonian tour has cost at most twice that of the MST, and therefore at most twice that of the minimum metric TSP tour.

10.3 Using minimum cost perfect matchings for lower cost Euler tours

The above factor two algorithm doubles the whole MST, only to construct a new graph G' with all vertices of even degree, thus ensuring the existence of an Euler tour in G'; a DFS on the Euler tour is then used to generate a Hamiltonian metric TSP tour by short-circuiting. Instead, we could avoid *doubling* all the $n-1$ edges of the MST and bother only about the set $V' \subseteq V$ of $|V'|$ vertices in the MST T, where V' is the subset of vertices of V with only the odd degree vertices in the MST T. Each pendant vertex in T is of odd degree one. Some internal vertices of T could also be of odd degree in the tree T. (The even degree vertices in $V \setminus V'$ do not bother us.) Christofedes used the technique of Euler tour construction by appending the MST T of $n-1$ edges with a set of $\frac{|V'|}{2}$ edges, where we note that $|V'|$ is an even number. Why is $|V'|$ even?

[We use induction on the number of vertices to show that any tree has an even number of odd-degree vertices. We can grow a tree by adding a pendant edge (u,v) where u is already in the tree and v is the new vertex; this increases the number of odd-degree vertices as v has degree 1, and (i) if u had even degree then u now becomes odd-degree as well (making the total number of odd-degree vertices again even), and (ii) if u had odd-degree then there is no change in the number of odd-degree vertices overall.]

The edges added comprise the *minimum cost perfect matching* in $G'(V',E')$, where the graph G' is the induced subgraph of G over the vertex subset V' of V with edges $E' \subseteq E$. The MST T we start with is already a connected graph. Adding new edges to T does not destroy the connectedness property of the resulting graph G' with edge set $E' = T \cup M$, where M is the *minimum cost perfect matching* in $G'(V',E')$. These edges in M, added to T are however special edges– to reiterate, they are the $\frac{|V'|}{2}$ edges of the *minimum cost perfect matching* in $G'(V',E')$, the induced subgraph of G over vertex set V'. Such a minimum cost perfect matching can be computed in polynomial time $O(n^3)$ time by an algorithm of Gabow.. How do we do this computation?

Now, consider an Euler tour $\mathcal{T}$ of $T \cup M$. Let C be the tour that visits the n vertices in G in the order of their first appearance in the Euler tour $\mathcal{T}$ of $M \cup T$. This C is the output TSP tour of the algorithm by Christofedes. Since C is found from $\mathcal{T}$ by DFS, we have $cost(C) \leq cost(\mathcal{T})$ by triangle inequality. So, $cost(C) \leq cost(\mathcal{T}) \leq cost(T) + cost(M) \leq OPT + \frac{OPT}{2} = \frac{3}{2}OPT$, because $cost(M) \leq \frac{OPT}{2}$, as we will soon show below; we already know that $cost(T) \leq OPT$.

To see that $cost(M) \leq \frac{OPT}{2}$, consider the minimum metric TSP tour $\mathcal{C}^*$ of cost OPT for $G(V,E)$. Let $\mathcal{C}'$ be the tour on V' obtained by *shortcutting* $\mathcal{C}^*$, that is by doing a DFS on $\mathcal{C}^*$ and picking

only the vertices of V' in the order of their first appearance in $\mathcal{C}^*$. By triangle inequality, we have $cost(\mathcal{C}') \leq cost(\mathcal{C}^*) = OPT$. Observe that $\mathcal{C}'$ has edges alternating on the even number of vertices in V', and therefore it has two perfect matchings. The lower cost perfect matching of these two matchings has cost at most $\frac{cost(\mathcal{C}')}{2} \leq \frac{OPT}{2}$. Also, $cost(M) \leq \frac{cost(\mathcal{C}')}{2} \leq \frac{OPT}{2}$, since M is the minimum cost perfect matching in $G(V', E')$.

11 The k-centre and the k-suppliers problems

The k-center problem is formally stated as follows. Let $G = (V, E)$ be a complete graph having a non-negative cost d_{ij} associated with each edge (v_i, v_j) of E. We assume that for every triple of vertices v_i, v_j , $v_l \in V$, the distances satisfy the triangle inequality, i.e., $d_{ij} \leq d_{il} + d_{lj}$. Given a positive integer k, (i) chose a set (called cluster centers) $S \subseteq V$ of $|S| = k$, and (ii) assign each of the remaining vertices $V \subseteq S$ to its nearest cluster center. The objective is to minimize the *maximum distance* of a vertex to its own *cluster center*.

11.1 Further geometric interpretations

Geometrically, the goal is to find k different *balls* centred at vertices of V, covering all the vertices in V, so that the radius of the largest ball is as small as possible. Why? In other words, the goal is to find a set $S \subseteq V$ of the centers of k different balls of the *same radius* r that cover all points in $V \setminus S$, so that this radius r is as small as possible.

First, we define the distance of a vertex i from a set $S \subseteq V$ of vertices to be $d(i, S) = min_{j \in S} d_{ij}$. Then the corresponding radius for S is equal to $max_{i \in V} d(i, S)$, and the goal of the k-center problem is to find a subset S of V of size k of *minimum radius*.

Again in this problem, we will use the triangle inequality. The n points of a set V of points with pairwise distances obeying the triangle inequality are given. We study the specific *clustering problem* of choosing a set S of k out of n points as *the centres of clusters* or *cluster centres*, so that points of $V \setminus S$ closer to a specific *cluster centre* in S than any other *cluster centres* in S are grouped into a *cluster*. The *cluster radius* is the radius of the smallest *ball* (circle) centred at each *cluster centre* and enclosing all points of that cluster. The maximum of the k cluster radii has to be minimised. This is an NP-hard problem; we present a factor two approximation algorithm as given in [7].

11.2 The algorithm

The approximation algorithm is simple: it selects an arbitrary vertex of V initially as one cluster centre in the set S of cluster centres. Then, it repeatedly chooses cluster centres for newer clusters till all k centres are selected in S. Every subsequent cluster centre is chosen by selecting a vertex $i \in V \setminus S$ whose distance $d(i, S)$ to the points in the current set S of cluster centres, is maximized.

For proving the factor two approximation bound, we again choose an arbitrary optimal solution S^* with r^* denoting the radius of the largest cluster in the optimal solution S^*. Due to triangle inequality, the distance between any two vertices within any cluster of the optimal solution S^* is bounded by $2r^*$. The solution S of k cluster centres, as identified by our approximation algorithm

may be different from S^*. Assume that the algorithm has chosen only one vertex v_i in S from a cluster B of the optimal solution S^*. If only one vertex is selected in S from each cluster of the optimal solution S^*, then any vertex v_l within the cluster with v_i as the center (in cluster B) is within a distance of $2r^*$ because the distance from each of v_i and v_l to the center of B is r^* (in the optimal solution S^*).

Now consider the other situation where one vertex v_i in a cluster B of S^* is already selected in S and the algorithm again chooses another vertex v_j in B as a cluster center in S. Again, the distance between v_i and v_j is bounded by $2r^*$. Moreover, v_j must be the furthest point from all points in S including v_i by the choice of the algorithm, and therefore, all the given points are within a distance of $2r^*$ of some center point already selected in S. Why? This argument holds even if the algorithm adds more points of B to S subsequently. Why?

Now consider the problem where we need "centres" to be placed in specialized "supplier" nodes, whereas the other nodes will simply act as "customer" or "consumer" nodes. See problem 2.1 from [7].

11.3 The modified k-suppliers problem

Given a set V of vertices on a metric space, and a set $F \subseteq V$ of suppliers where $D = V$ is the set of clients, we define for each $v_i \in V$, $d(v_i, S) = \min_{v_j \in S} d(v_i, v_j)$. Find $S \subseteq F$ with $|S| = k$ such that $\max_{v_i \in V} d(v_i, S)$ is minimized. So, the nearest supplier in S to $v_i \in V$ defines $d(v_i, S)$. The subset S has to be chosen minimizing $\max_{v_i \in V} d(v_i, S)$.

Initialize $S = \emptyset$
1. Run the k-center 2-approximation algorithm on V. Let it produce the k-center set $T \subseteq V$.
2. Let $TF = T \cap F$, and $S = S \cup TF$; We include the whole of TF in S.
3. For each $u \in T \setminus T_F$
(i) find a vertex $u' \in F$, where $d(u, u')$ is minimized $\forall u' \in F$.
(ii) $S = S \cup \{u'\}$; Include u' in S.
4. Return S

Analysis

Claim: The above algorithm is a 3-approximation algorithm for the k-suppliers problem.

Proof: Let OPT be the actual optimal value i.e.,

$$OPT = \min_{\substack{S \subseteq F \\ |S|=k}} \max_{v_i \in V} d(v_i, S)$$

Let OPT^* be the optimal value for the k-center problem i.e.,

$$OPT^* = \min_{\substack{S \subseteq V \\ |S|=k}} \max_{v_i \in V} d(v_i, S)$$

As the condition $S \subseteq F$ is relaxed for OPT^*, we have

$$OPT^* \leq OPT \tag{5}$$

The k-center algorithm ensures $d(v,S) \leq 2 * OPT^* \leq 2 \times OPT$, $\forall v \in V$ such that, $d(v,S) = d(v,u)$ where $u \in TF$ (i.e., the vertices having their supplier as one of the original k-center vertices). See Step 2 of the algorithm.

Let v be such that its nearest k-center vertex is u where $u \notin TF$. Let $u' \in F$ be the nearest vertex from u to F. Therefore, by Step 3(ii) of the algorithm, $u' \in S$. Also, as u' is nearest to u amongst all vertices in F, we have $d(u,u') \leq OPT$.

Now, $d(u',v) \leq d(u',u) + d(u,v)$ [By Triangular inequality]

$\leq OPT + 2 \times OPT^*$

$\leq OPT + 2 \times OPT = 3 \times OPT$

So, any supplier that gets allotted to v must be within $3 \times OPT$ distance from it.

11.4 The k-suppliers problem as in Williamson-Shmoys, Exercise 2.1

In the modified problem of the previous section, we considered the suppliers also to be behaving as clients. So, although suppliers supply to clients, suppliers also supply to suppliers. In this section however, we do not permit suppliers to be clients. The total number of suppliers will again be exactly k.

Treating all elements in D uniformly, we can apply the k-center algorithm to D, and then find the nearest supplier $s(u) \in F$ to each center $u \in D$.

Let OPT be the optimal value for the k-supplier radius. For any $d_i \in D$ let $f_i \in F$, be its nearest supplier. Note that $dist(d_i, f_i) \leq OPT$ for any i. If $|d| \leq k$, then $S = \{f_i | d_i \in D\}$ is an optimal solution. So from now onwards assume that $|D| > k$.

The k-center algorithm computes the subset D' of D as a set of k centres. Let $D' = \{d_1, d_2, ..., d_k\}$.

Take $S = \{f_i | d_i \in D'\}$ as a solution. We prove that this gives a 3-approximation.

Consider an arbitrary customer $v \in D$. If there is a customer $d_i \in D'$ at a distance at most $2 \times OPT$ from v, then by the triangle inequality $dist(v, f_i) \leq dist(v, d_i) + dist(d_i, f_i) \leq 2 \times OPT + OPT = 3 \times OPT$.

Now assume that there is no customer $d_i \in D'$ at a distance at most $2 \times OPT$ from v. Then, by the algorithm, for any two customers in $d_i, dj \in D'$ we have $dist(d_i, d_j) > 2 \times OPT$. However, in that case, the set $D' \cup \{v\}$ consists of $k+1$ customers such that the distance between any pair of vertices is more than $2 \times OPT$. This is impossible since in that case at least $k+1$ suppliers are needed for a solution with radius OPT.

12 Facilities location

The *uncapacitated facility location* problem is a combinatorial optimization problem. It has applications in setting up facility distribution centres. A 4-approximation primal rounding algorithm is first analyzed. A 3-approximation primal-dual algorithm is also considered here.

In the uncapacitated facility location problem, we have a set of *clients* or *demands* D and a set of *facilities* F. For each client $j \in D$ and facility $i \in F$, there is a cost c_{ij} of assigning client j to facility i. Furthermore, there is a cost f_i associated with each facility $i \in F$. The aim is to choose a subset $F' \subseteq F$ so as to minimize the total cost of the facilities in F' and the cost of assigning each client $j \in D$ to the nearest facility in F'. In other words, we wish to find $F' \subseteq F$ such that,

$$Minimize \quad \sum_{i \in F'} f_i + \sum_{j \in D, i \in F'} c_{ij}$$

where the first part is called *facility cost* and the second part is called *assignment cost* or *service cost*. This is an NP-hard problem.

The integer programming formulation for this problem has decision variables $y_i \in \{0,1\}$ for each facility $f_i \in F$. If we decide to open facility i, then $y_i = 1$, and $y_i = 0$ otherwise. We also introduce decision variables $x_{ij} \in \{0,1\}$ for all $i \in F$ and all $j \in D$. If we assign client j to facility i, then $x_{ij} = 1$ while $x_{ij} = 0$ otherwise.

The objective function becomes,

$$Minimize \quad \sum_{i \in F} f_i y_i + \sum_{i \in F, j \in D} c_{ij} x_{ij}$$

We need to make sure that each client $j \in D$ is assigned to exactly one facility. This can be done by,

$$\sum_{i \in F} x_{ij} = 1$$

We also need to make sure that the client is assigned to a facility that is open. This can be done by,

$$x_{ij} \leq y_i$$

Thus, the integer linear programming formulation of the facility location problem can be summarized as follows:

$$\begin{aligned}
\textit{minimize} \quad & \sum_{i \in F} f_i y_i + \sum_{i \in F, j \in D} c_{ij} x_{ij} & \\
\textit{subject to} \quad & \sum_{i \in F} x_{ij} = 1, & \forall j \in D, \\
& x_{ij} \leq y_i, & \forall i \in F, j \in D, \\
& x_{ij} \in \{0,1\}, & \forall i \in F, j \in D, \\
& y_i \in \{0,1\}, & i \in F.
\end{aligned}$$

Linear programming relaxation from the integer linear programming can be obtained by replacing the constraint $x_{ij} \in \{0,1\}$ and $y_i \in \{0,1\}$ with $x_{ij} \geq 0$ and $y_i \geq 0$. Thus, the relaxed linear programming can be summarized as follows:

$$\begin{aligned}
\textit{minimize} \quad & \sum_{i \in F} f_i y_i + \sum_{i \in F, j \in D} c_{ij} x_{ij} & \\
\textit{subject to} \quad & \sum_{i \in F} x_{ij} = 1, & \forall j \in D, \\
& x_{ij} \leq y_i, & \forall i \in F, j \in D, \\
& x_{ij} \geq 0, & \forall i \in F, j \in D, \\
& y_i \geq 0, & i \in F.
\end{aligned}$$

In order to formulate the dual of the relaxed linear programming, we argue as follows. Ignoring the facility cost, by setting $f_i = 0$ for all $i \in F$, the optimal solution is to open all the facilities and assign each client to its nearest facility. We introduce a dual variable v_j and set it as $v_j = \min_{i \in F} c_{ij}$; a modest lower bound for the primal's objective function cost in an integral solution is $\sum_{j \in D} v_j$ therefore. We can better the lower bound by considering non-zero facility costs. Each facility takes its cost f_i and shares it apportioned among the clients it provides service to: $f_i = \sum_{j \in D} w_{ij}$, where each $w_{ij} \geq 0$. A client j needs to pay this share only if it uses facility i. Hence, the optimal solution is to assign all the clients to the nearest facility, now we can set $v_j = \min_{i \in F}(c_{ij} + w_{ij})$, and if we allow $v_j \leq c_{ij} + w_{ij}$, the objective function maximizing $\sum_{j \in D} v_j$ forces v_j to be equal to the smallest right-hand side overall facilities $i \in F$, then any feasible solution to dual is lower

bound on cost of optimal value on facility location problem. Thus, the dual linear program for the primal linear program can be summarized as:

$$
\begin{aligned}
\textit{maximize} \quad & \sum_{j \in D} v_j \\
\textit{subject to} \quad & \sum_{j \in D} w_{ij} \leq f_i, && \forall i, \in F \\
& v_j - w_{ij} \leq c_{ij}, && \forall i \in F, j \in, D \\
& w_{ij} \geq 0, && \forall i \in F, j \in D.
\end{aligned}
$$

12.1 4-factor algorithm

We need to determine a subset $F' \subseteq F$ of facilities to be opened to serve customers in D. So, facilities indicator variables f_i and connection indicator variables x_{ij} appear in the primal ILP. We connect each $j \in D$ to exactly one $i \in F$. Also, clients are assigned to only the opened facilities. These two conditions appear as constraints in the primal ILP.

If we allow opening facilities without any restriction then we may use $v_j = min_{i \in F} c_{ij}$ as the connection cost. This will make $\sum_{j \in D} v_j$ a lower bound for the primal objective function. Since such unrestricted action might shoot up facilities installation costs, we may enhance v_j by apportioning cost f_i as $f_i = \sum_{j \in D} w_{ij}$, $w_{ij} \leq 0$.

....

12.2 3-factor algorithm

We set $S = D$, the set of clients. We raise v_j's and w_{ij}'s uniformly until either (i) some client $j \in D$ *neighbours* some facility $i \in F$, or (ii) some facility $i \in F$ becomes *saturated*. Such saturated facilities are kept in a set T. All clients neighbouring a facility discovered above are moved out of the set S. A subset $T' \subseteq T$ of facilities is opened by selecting one facility at a time to cover a number of clients; whenever any such facility i is moved into T', all other facilities $h \in T$ are removed from T if h and i are *contributed* to by some client j. Please recall the definitions of a client *neighbouring* a facility ($v_j^* \geq c_{ij}$), a *saturated* dual constraint (obeying equality), and when it is said that a client *contributes* to a facility ($w_{ij} > 0$).

More precisely, v_j are increased uniformly for all $j \in S$. Once $v_j = c_{ij}$ for some i, we increase w_{ij} and v_j uniformly. It may be the case that some dual inequality becomes tight (saturated) in the process. It may be the case that v_j grows along with some w_{ij} till j neighbours i. Removal of j from S and addition of i in T are mentioned above. Recall that the neighbours of a facility i are in the set $N(i)$ of clients, and the neighbours of a client j are in the set $N(j)$ of facilities. From the set T we compute a subset T' of T as mentioned above. Whenever a facility i is added to T, we remove all $N(i)$ from S.

Once the whole set S is exhausted, we assign facilities from T' to the clients such that each client is assigned to its closest facility in T'.

If a client j has a neightbour i in T' then the client j is assigned to i and has connection cost $c_{ij} \leq v_j^*$ to i. Otherwise, we see due to Lemma 7.13 in WS that although j does not have a neighbour in T', there is a facility $i \in T'$ such that $c_{ij} \leq 3v_j^*$.

Assuming Lemma 7.3, we see Lemma 7.4 that proves the 3-factor approximation.

Note one important point that each client has a neighbouring facility in T. This is due to the maximality of (v^*, w^*). We refer to the proof of this fact to the section in WS. With the above discussion, we conclude the 3-factor claim.

13 Computing spanning trees of low maximum vertex degree

Starting with some initial structure and gradually modifying it with smart local changes, we can sometimes get provably good results. So, even if we start with an arbitrary spanning tree of the given graph $G(V,E)$ of n vertices, and keep reducing degrees of some large degree vertices in the tree, we may eventually get an spanning tree with a resonably low value for the maximum vertex degree in the final spanning tree, after a sufficiently large number of steps. Reduction of the large degree of a vertex (in a spanning tree) can also reduce the degree of an earlier neighbour of this vertex (in the modified spanning tree), thereby possibly reducing the largest degree too, after several such steps.

13.1 Local action

The mechanism for reducing a vertex degree by adding an edge $\{v,w\}$ to a spanning tree T is as follows. If the cycle generated by adding the new edge has the vertex u in it then we can drop an edge ending on u in that cycle to reduce the degree of u. The degrees of v and w rise by one each in the resulting spanning tree T. So it is prudent and necessary to choose $\{v,w\}$ so that the larger of the degrees of these two vertices is at least two less than that of u. Then, the modified degrees of v and w remain less than the modified (reduced) degree of u.

13.2 The overall algorithm

The algorithm starts with an arbitrary spanning tree and ends with what we call the locally optimal spanning tree. In the locally optimal spanning tree (say) T, we do not have a sufficiently high degree vertex u whose degree can be further reduced by adding an edge $\{v,w\}$ to the current tree T as explained above. However, we also have to determine how long this algorithm runs. The chosen vertex u must have degree at least $\Delta(T) - \log n$. Henceforth, let $l = \log n$.

13.2.1 A lower bound for the maximum vertex degree in any MST

Let $S \subseteq V$ be a set of vertices with the following property. All edges in $G(V,E)$ that jump across the $k+1$ connected components of any spanning tree T (components created due to the removal of some k edges of T), have at least one vertex in S. Now any spanning tree T of $G(V,E)$ must have at least k edges that connect across the $k+1$ components defined above due to the removal of k edges from the spanning tree T. All these (at least) k edges must be covered by the vertex set S. So, the average vertex degree in S in the spanning tree T must be at least $k|S|$. If $T = T^*$ where T is a spanning tree with the minimum value of maximum vertex degree then we have $OPT \geq k|S|$.

14 Scheduling jobs on a single machine

Scheduling jobs with specified deadlines on a single machine is a simply stated and fundamental combinatorial optimization problem. We wish to schedule n jobs on a single machine. The given single machine can process at most one job at a time. Also, each job beng processed, must be completed fully once it has been started. Let p_j units of time be the processing time for job j, $1 \leq j \leq n$. The processing of job j must start only on or after a specified release time r_j, $j = 1, ..., n$. We assume that all release times are non-negative. Since each job j has a specified due time d_j, its *lateness* L_j is $C_j - d_j$, where C_j is the actual completion time for job j. Note that the optimal (maximum lateness) can be zero or even negative, when due times for jobs can be large positive quantities. So, we simplify the problem by assumimg that all due times are negative.

We wish to schedule the jobs minimizing the maximum lateness $L_{max} = max_{j=1}^{n} L_j$. Consider the example where $p_1 = 2, r_1 = 0, p_2 = 1, r_2 = 2, p_3 = 4, r_3 = 1$. Let $C_1 = 2, C_2 = 3, C_3 = 7$. If the deadlines for the jobs are such that $d_1 = -1, d_2 = 1, d_3 = 10$, then $L_1 = 3, L_2 = 2, L_3 = -3$. Hence, $L_{max} = L_1 = 3$ for the schedule of jobs 1, 2 and 3 in that order. We are interested in the optimal schedule that minimizes maximum lateless. We may compute a schedule in polynomial time that has maximum lateness at most as bad as twice the maximum lateness of the optimal solution. The exposition here is from the text by Williamson and Shmoys [7].

14.1 A lower bound for maximum lateness

Let L^*_{max} denote the optimal value for maximum lateness. We show that for each subset S of jobs, $L^*_{max} \geq r_{(S)} + p_{(S)} - d_{(S)}$, where $r_{(S)} = min_{j \in S} r_j$, $p_{(S)} = \sum_{j \in S} p_j$, and $d_{(S)} = max_{j \in S} d_j$. Consider the optimal schedule, viewing it as a schedule only for the jobs in the subset S. Let job j be the last job in S to be processed. Since none of the jobs in S can be processed before $r_{(S)}$, and in total they require $p_{(S)}$ time units of processing, it follows that the job j cannot complete earlier than $r_{(S)} + p_{(S)}$. The due date of job j is $d_{(S)}$ or earlier. The lateness of job j therefore is at least $r_{(S)} + p_{(S)} - d_{(S)}$. So, the maximum lateness $L^*_{max} \geq r_{(S)} + p_{(S)} - d_{(S)}$. In particular, with the singleton set $S = j$ having only one job j, we have $L^*_{max} \geq r_j + p_j - d_j \geq -d_j$.

14.2 An algorithmic (polynomial time) upper bound for maximum lateness

Now we use the EDD (earliest due date) method for computing a schedule whose maximum lateness is at most twice the maximum lateness L^*_{max} in the optimal schedule. Since a job j is available at time t if its release date $r_j \leq t$, we consider the following natural step: at each moment that the machine is idle, start processing next *an available job with the earliest due date*. This is known as the *earliest due date (EDD)* rule. We show that the EDD rule yields a 2-approximation algorithm for the problem of minimizing the maximum lateness on a single machine, subject to release dates with *negative* due dates.

Consider the schedule produced by our algorithm using the EDD rule, and let job j be a job of maximum lateness in this schedule; that is, $L_{max} = C_j - d_j$. Focus on the time C_j in this schedule and find the earliest point in time $t \leq C_j$ such that the machine was processing without any idle time for the entire period $[t, C_j)$. Let S be the set of jobs that are processed in the interval $[t, C_j)$. By our choice of t, we know that just prior to t, none of the jobs in S were available, and clearly at least one job in S is available at time t; hence, $r_{(S)} = t$. [The jobs selected before time t were available

to be picked up before time t, but none of the jobs in S were available before time t.] Furthermore, since only jobs in S are processed throughout this time interval, $p_{(S)} = C_j - t = C_j - r_{(S)}$. Thus, $C_j \leq r_{(S)} + p_{(S)}$. Since $d(S) < 0$ by assumption, $L^*_{max} \geq r_{(S)} + p_{(S)} - d_{(S)} \geq r_{(S)} + p_{(S)} \geq C_j$. On the other hand, with the singleton set $S = j$, we have $L^*_{max} \geq r_j + p_j - d_j \geq -d_j$. Therefore, by addition of the two inequalities, we dedcue that $2L^*_{max} \geq C_j - d_j = L_{max}$.

15 Multiway cut

Given a set $S = \{s_1, s_2, ..., s_k\}$ of *terminals* where $S \subseteq V$, a *multiway cut* is a set of edges whose removal disconnects the specified terminals from each other. The *multiway cut problem* asks for such a minimum weighted cut. This presentation is from Section 4.1 of [8]. The problem of finding a minimum weight multiway cut is NP-hard for any fixed $k \geq 3$. Observe that the case $k = 2$ is precisely the minimum (s,t)-cut problem, which is solvable in polynomial time using newtwork flows. We study a $2 - \frac{2}{k}$ approximation algorithm for this problem as follows. For each $i = 1, ..., k$ do (i) identify the terminals in $S \setminus \{s_i\}$ into a single vertex, (ii) compute a minimum weight cut C_i for $(s_i, S - \{s_i\})$ using a network flow algorithm, and (iii) discard the heaviest of these k cuts. The output answer is the union of the rest, say C.

Let A be an optimal multiway cut in G. A can be viewed as the union of k cuts as follows. The removal of A from G creates k connected components, each having one terminal. Let $A_i \subseteq A$ be the cut separating the component containing s_i from the rest of the graph. So, $A = \cup_{i=1}^{k} A_i$. Since each edge of A is incident at two of these components, each edge belongs to two of the cuts. So, $\sum_{i=1}^{k} w(A_i) = 2w(A)$.

15.1 The computational lower bounds on the sizes of cuts in the optimal solution

Now the main lower bound argument is that C_i being a minimum weight cut for s_i, we have $w(C_i) \leq w(A_i)$. [A similar lower bound argument is used also in the much more complex proof of an approximation bound for the minimum weight k-cut problem in Section 4.2 of [8].] Note that this already gives a 2-approximation algorithm, by taking the union of all k cuts C_i. [This union step in the alorithm is reminiscent of the vertex cover algorithm where for each matching edge we include vertices at both ends of the edge.] Finally, since C is obtained by discarding the heaviest of the cuts C_i, we have $w(C) \leq (1 - \frac{1}{k}) \sum_{i=1}^{k} w(C_i) \leq (1 - \frac{1}{k}) \sum_{i=1}^{k} w(A_i) = 2(1 - \frac{1}{k}) w(A)$.

16 The k-cut problem

The k-cut problem is similar to the multiway cut problem but in this case we do not provide any set of k terminals. This exposition is based on Section 4.2 of [8]. This k-cut problem is a more general problem. The nice approimation bound in this problem requires a complicated analysis using Gomory-Hu trees. A k-cut is is a set of edges whose removal leaves k connected components for a connected graph. For positive edge weights, we wish to find a minimum weighted k-cut. We will address the well-known result about the factor $2 - \frac{2}{k}$ approximation algorithm.

16.1 The Gomory-Hu tree and minimum weight cuts

We use the Gomory-Hu tree T defined on the same vertex set V as that of the graph $G(V,E)$ with positive edge weights for edges in E. The edges of T may not belong to E. Suppose the removal of an edge e_T of T gives two components of the vertex set namely, S and $V \setminus S$. Let $E' \subseteq E$ be the edges of G whose removal from G partitions vertex set of G into S and $V \setminus S$. In other words, E' is the cut-set for S and $V \setminus S$ in G. Assign the sum of weights on edges of E' on edge e_T of T. So, edges of T have weights corresponding to the minimum weight cut-sets in G. Thus, out of $\binom{n}{2}$ minimum weight $u-v$ cuts, only $n-1$ minimum weight cut sets in G are used as weights on edges of T. Thus, the min-cut tree T (called the Gomory-Hu Tree) has the property that the minimum cut between any two nodes v_i and v_j in G is the smallest weight edge in the unique path that connects v_i and v_j in T.

16.2 Properties of any optimal k-cut A and the approximation algorithm for computing a k-cut

Let S be the union of minimum weights cuts in G associated with l edges of T. Then, the removal of S from G leaves a graph with at least $l+1$ components, as it leaves T with $l+1$ components. The k-cut approximation algorithm we analyze is simple; the algorithm first constructs the Gomory-Hu tree T for G in polynomial time, and then constructs a k-cut set C by taking the union of cut edges of G corresponding to the *lightest* $k-1$ edges in T. If more than k connected components result then we keep throwing back cut edges till there are exactly k components. So, much for the polynomial time approximation algorithm.

Let A be an optimal k-cut in G, which can be viewed as the union of k cuts. Let the removal of A from G create k connected components, $V_1, V_2, ..., V_k$. Let $A_i \subseteq A$ be the cut separating V_i from the rest of the graph. Then, $A = \cup_{i=1}^{k} A_i$. Each edge of A is incident at two of these components. So, each edge of A is in two of the cuts. So, $\sum_{i=1}^{k} w(A_i = 2w(A)$.

16.3 Establishing the novel lower bound

Now we have to connect the properties of the output C of the approximation algorithm with the properties of the arbitrary minimum k-cut A, in order to establish the factor $2-\frac{2}{k}$ ratio bound. The main idea is to *identify* (show the existence of) $k-1$ cuts defined by the edges of T whose weights are *dominated* by the weight of the cuts $A_1, A_2, ...A_{k-1}$ of the optimal k-cut A, where without loss of generality we assume that A_k is the heaviest cut in A. This lower bound argument is crucial. These $k-1$ cuts are identified as follows.

Let B be the set of edges of T that connect across two of the sets $V_1, V_2, ..., V_k$, as partitioned by A. Consider the graph on the vertex set V and the edge set B. We shrink each of the sets $V_1, V_2, ..., V_k$ to respective k single super-vertices. So, we have essentially superimposed the tree T over the k connected components of an optimal k-cut A giving a possibly non-tree but connected graph with edge set B on the k super-vertices. Observe that this graph must be connected since T is itself connected. Throw edges away from this graph until a tree T' survives. Let $B' \subseteq B$ be the leftover edges in T'. Clearly, $|B'| = k-1$ as T' is a tree of k vertices. The edges of B' define the required $k-1$ cuts which are dominated by $k-1$ cuts from A. Assuming that A_k is the heaviest cut amongst the cuts of A. Imagine rooting tree T' at V_k. We now define a correspondence between the

edges in B and the sets $V_1, V_2, ..., V_{k-1}$: each edge corresponding to the set it comes out of in the rooted tree, going towards the parent.

Suppose edge $(u,v) \in B'$ corresponds to a set V_i in this manner where V_j is the parent of V_i in T', $u \in V_j$ and $v \in V_i$. The weight of a minimum $u-v$ cut in G is $w'(u,v)$, as represented in the Gomory-Hu tree T, by the edge (u,v) of T in edge set B'. Observe that A_i is also a $u-v$ cut in G (but may not be the minimum such cut!); therefore we have $w(A_i) \geq w'(u,v)$ for all i, $1 \leq i \leq k-1$. Since, the union of the lightest $k-1$ cuts defined by T is C in our approximation algorithm, we have $w(C) \leq \sum_{e \in B'} w'(e) \leq \sum_{i=1}^{k-1} w(A_i) \leq \sum_{i=1}^{k} (1-\frac{1}{k}) A_i = 2(1-\frac{1}{k}) w(A)$.

17 Using set covering for the shortest superstring problem

The $2H_n$ factor approximation algorithm uses a reduction to the more general set cover problem and then uses the greedy weighted set cover heuristic. The set cover instance is designed so that the solution computed yields a superstring which is at most $2H_n$ times the length of the shortest superstring. The weights of the sets in the set cover instance are lengths of certain "covering" strings. Given n strings as input, the set cover universe comprises of precisely the given n strings which need to be substrings of the computed superstring.

18 Online deterministic paging algorithms and their competitive ratio bounds

Theorem 1 from [6] claims that LRU and FIFO are k-competitive. For arbitrary requests streams σ, we must show that $C_{LRU}(\sigma) \leq k.C_{OPT}(\sigma)$. The phases $P(0)$, $P(1)$, ... in σ are substrings of page requests, where in each substring some page requests raise page faults in the online algorithm LRU, or in the deterministic offline algorithm OPT. All we need to prove is that there is at least one fault in each of these phases of page requests for OPT, whereas in LRU we already have at most k faults in $P(0)$ and exactly k faults in each phase $P(i)$ for $i > 0$.

Since LRU and OPT start with the same set of k pages in their respective fast memories, OPT has a page fault on the first page request on which LRU has a fault. So, $P(0)$ has at least one page request on which OPT has a page fault.

To show that the other phases too have at least one page fault for OPT, we use Lemma 1.

Lemma 1. *Let page p be the last requested page in phase $P(i-1)$ (at time $t_i - 1$). Then $P(i)$ must contain requests to k distinct pages that are different from p.*

Before we prove Lemma 1, we show how this result is used to show that OPT must have a page fault in $P(i)$. Since $P(i)$ has requests to k distinct pages other than p but has p in its fast memory at the end of $P(i-1)$, $P(i)$ cannot have all the k distinct pages in its fast memory. So, $P(i)$ must have a page fault for OPT.

(Page requests may lead to page faults. Lemma 1 states that there are k page requests in $P(i)$ for pages different from p, and all these k requests are for distinct pages. Here p is the last requested page in $P(i-1)$, just before time t_i when $P(i)$ starts.)

The proof of Lemma 1 goes case by case. LRU has k faults in $P(i)$ by construction. If all these page requests are for distinct pages and these requests are not for page p then Lemma 1 holds. So, we assume that LRU faults *twice* on some page q in $P(i)$. Page q is served and brought into fast memory at time s_1 and then evicted from fast memory at time t before time s_2, when it is again served in the phase $P(i)$. When q is evicted at time t, it is the *least recently requested* page in the fast memory. So, the sequence $\sigma(s_1)$, ... , $\sigma(t)$ must contain requests to $k+1$ distinct pages, at least k of which must be different from p. *(After coming into fast memory at time s_1, q remains in fast memory till it is evicted at time t. So there must be k requests (after the time s_1 requests for page q and before time t when q is evicted) for non-q pages, and at most one of these k requests can be for page p. If two (or more) of these k requests (after the time s_1 request for page q) are for page p, then these k requests cannot be for $k-1$ distinct non-q pages leading to eviction of q at time t. So p can be requested only at most once in the k distinct requests from time s_1+1 till time t, that witness the eviction of q at time t. The request at time s_1 is for q.)*

(Page p is requested just before phase $P(i)$ started and this phase has exactly k faults in LRU.)

The only other case is when LRU faults (i) not on all distinct pages different from p, or (ii) not twice on some page q, but (iii) faults once for a page request to page p. Let $t \leq t_i$ be the time when page p is evicted so that page p was there in fast memory right before time t_i in the beginning of the phase $P(i)$ as it was the last requested page in phase $P(i-1)$. So, the sequence $\sigma(t_i-1)=p$, $\sigma(t_i)$, ..., $\sigma(t)$ has a request to the page p in the beginning of the sequence. Since p is evicted at time t, there must a sequence of k distinct page requests that cannot be requests for p from time t_i till t. So, we have a set of k distinct non-p page requests in $P(i)$. This complete all the cases for the proof of Lemma 1.

19 Amortized bound for the competitive ratio for paging using a potential function

As usual let $\sigma = \sigma(1), \sigma(2), ..., \sigma(m)$ be an arbitrary request sequence. At any time let S_{LRU} be the set of pages contained in LRUs fast memory, and let S_{OPT} be the set of pages contained in the fast memory of OPT. Set $S = S_{LRU} \setminus S_{OPT}$.

Assign integer weights from the range $[1:k]$ to the pages in S_{LRU} such that, for any two pages $p, q \in S_{LRU}$, $w(p) < w(q)$ if and only if the last request to p occurred earlier than the last request to q.

Let the potential function $\phi = \sum_{p \in S} w(p)$. Consider an arbitrary request $\sigma(t) = p$ and assume without loss of generality that OPT serves the request first and LRU serves second. If OPT does not have a page fault on $\sigma(t)$, then its cost is 0 and the potential does not change immediately. On the other hand, if OPT has a page fault, then its cost is 1. In that case, OPT might evict a page (say, r) that is in LRUs fast memory, in which case the potential increases by $w(r)$ as r now becomes the member of S. Since $w(r)$ can be at most k, the potential can increase by at most k for evicting r from OPT's fast memory. Note that when LRU does not have a fault on $\sigma(t)$, the cost is 0 and the potential cannot change.

If LRU has a page fault, its cost on the request is 1. In that case, the potential decreases by at least 1 as follows if OPT has not had a page fault on this request. Immediately before LRU serves $\sigma(t)$, page p is only in OPTs fast memory but not in LRU's fast memory, so that that there

is no page fault of OPT. By symmetry, there must be page(s) q only in LRUs fast memory, i.e, $q \in S = S_{LRU} \setminus S_{OPT}$. If q is evicted by LRU during the operation, then the potential decreases by $w(q) \geq 1$. Otherwise, since p is loaded into fast memory, the weight of q must decrease (why? because p gets weight k and the rest of the pages in the fast memory of LRU reduce in weights by unity), and thus the potential must decrease by at least 1.

In summary, every time OPT has a fault, the potential increases by at most k. Every time LRU has a fault, the potential decreases by at least 1.

20 Online coloring for complements of bipartite graphs

Naturally, the first fit method produces a maximal stable sequence partition of $V = S_1 \cup ..., \cup S_k$ where S_i is a maximal non-empty stable set in the subgraph induced by $S_i \cup ..., \cup S_k$, for $1 \leq i \leq k$. Also, every maximal stable sequence partition of V can be reproduced by a first fit process, if an appropriate ordering of the vertices is taken.

20.1 The formulation using maximal stable sets partitioning

We show that the vertex set of the complement of a bipartite graph may be partitioned into a number k of maximal stable sets, where we may assign one unique colour to all vertices of each of such k stable sets. Suppose we have already used $i-1$ colors and colored $i-1$ such stable sets S_1, S_2, ..., S_{i-1}. Let S_i be the maximal stable set in $S_i \cup S_{i+1} \cup ... \cup S_k$. It is easy to see that we can determine a sequence in which we can supply the vertices of the graph so that first-fit coloring will use exactly k colors, one for each S_i, $1 \leq i \leq k$.

20.2 The stable sets and the online competitive ratio bound

For a maximal matching M in a bipartite graph $G(U \cup V, E)$, we have $|M| + |X| \leq \alpha(G)$ and $|M| + |Y| \leq \alpha(G)$, where $X \subseteq U$ and $Y \subseteq V$ are the unmatched vertices. Also, unmatched vertex pairs cannot be connected in G; any such pair forming an edge would add to the maximal matching M, yielding a bigger matching. So. $|X| + |Y| \leq \alpha(G)$. So, $2|M| + 2|X| + 2|Y| \leq 3\alpha(G)$ or, $2|M| + 2\alpha(G) \leq 3\alpha(G)$, or, $|M| \leq \frac{1}{2}\alpha(G)$. Observe that vertices of U (or, V) form a clique in G' (the complement graph of G), vertices of M in U (or, V) form a clique in G', and vertices of $X \cup Y$ form a clique in G'. Since both vertices of a matched edge (in M) can be coloured in G' with the same colour, and every vertex of $X \cup Y$ needs a different colour as $X \cup Y$ is a clique in G', $\zeta_{FF}(G) = \chi_{FF}(G') \leq |M| + |X \cup Y| \leq \frac{3}{2}\alpha(G) = \frac{3}{2}\chi(G')$.

20.3 The tight example

Let $V = A \cup B \cup C \cup D$ be the vertex set in a graph $G(V,E)$ where A, B, C and D are pairwise disjoint independent sets of k vertices each, and $A \cup B$, $B \cup C$, and $C \cup D$ induce complete bipartite subgraphs, totalling to $3k^2$ edges in all, in G. Now G is a bipartite graph $G(A \cup C, B \cup D, E)$. Edges run from A to B and from C to D. Also, edges run between B and C. Take a maximum matching M between B and C of size k in the bipartite graph $G(A \cup C, B \cup D, E)$. The unmatched sets are A and D. The maximum independet set is of size $2k$. This is a tight example therefore.

21 Online coloring for complements of chordal graphs

We show that if $G(V,E)$ is chordal, then $\zeta_{FF}(G) \leq 2\alpha(G)-1$ (or, $\chi_{FF}(G') \leq 2\chi(G')-1$), and there are graphs that attain equality. The result holds if G is a complete graph (G' is empty), as $\alpha(G)(=\chi(G'))$ is one. We use induction on V to prove the upper bound. Let $C_1, C_2, ..., C_k$ be a *first-fit clique partitioning* of G where C_i is a maximal clique in the subgraph of G induced by $C_i \cup C_{i+1} \cup ... \cup C_k$, for every i, $1 \leq i \leq k$.

To prove the result, it is enough to show that $\alpha(G) \geq \frac{k+1}{2}$ as $\zeta_{FF}(G) \leq k$. Here, C_1 induces a maximal independent set in G' and we can therefore use just one color, the first color. So the first-fit algorithm simply identifies a maximal clique in the chordal graph G. This is done using the "simplicial vertex" s in G from C_1, which is adjacent to the maximal clique induced by $C_1 \setminus \{s\}$. A chordal graph always has at least two such simplicial vertices. The removal of C_1 from G again results in a residual chordal graph. This process repeats with C_2, C_3 and so on so that we get a first-fit coloring of G' with k colors. So, $G_1 = G \setminus C_1$, and $C_2, C_3, ..., C_k$ is a first-fit clique partitioning of G_1. If G_1 has more components than G, then the claim easily follows by using the inductive hypothesis, as each additional component contributes one vertex to $\alpha(G)$. Otherwise, G has a simplicial vertex (a vertex whose neighbourhoods induce a clique) in C_1 as G is a perfect graph. This simplical vertex can be added to maximum independent set of G_1, that is $\alpha(G)$ is at least $\alpha(G_1)+1$, and again the proof follows by applying induction as $\alpha(G) \geq \alpha(G_1)+1 \geq \frac{(k-1)+1}{2}+1 > \frac{k+1}{2}$, that is $2\chi(G')-1 = 2\alpha(G)-1 > k \geq \chi_{FF}(G') = \zeta_{FF}(G)$.

22 The K-server problem

Online algorithms for the K-server problem are considered. K servers need to be moved around to service requests appearing online at points of a metric space.The total distance travelled by the K servers must be minimised, where any request arising at a point of the metric space must be serviced on site by moving a server to that site.$d(a_1,a_2)$ is defined as the distance between a_1 and a_2. M represents the metric space where d is the metric which satisfies the triangle inequality. M^K represents the set of configurations of the K points of M. Given configurations C_1 and C_2, $d(C_1,C_2)$ is the minimum possible distance travelled by K servers that change configuration from C_1 to C_2. $C_0 \in M^K$ is the initial configuration. Let $r = (r_1, r_2, ...r_m)$ be the sequence of request points in M. The solution $C_1, C_2, ..., C_m \in M^K$ is such that $r_t \in C_t, \forall t = 1....m$. Serving $r_1, r_2, ...r_m$ by moving through $C_1, C_2, ..., C_m$ entails solution cost $\sum_{t=1}^{m} d(C_{t-1}, C_t)$.

The online algorithm uses only $r_1, r_2 ... r_t$ and $C_0, .., C_{t-1}$ to compute C_t. The offline algorithm uses also $r_{t+1}, r_{t+2} ... r_m$. Given C_0, $r = (r_1, r_2, ...r_m)$, $cost_A(C_0, r)$ is the cost of the online algorithm A, and $opt(C_0, r)$ is the cost of the optimal algorithm. ρ is the competitive ratio. Competitive ratio is used as $cost_A(C_0,r) < \rho * opt(C_0,r) + \phi(C_0)$ for some ρ. $\phi(C_0)$ is independent of r. ρ_M may be used for metric space M. Conjecture: For every metric space with more than K distinct points the competetive ratio for the K-server problem is exactly K. $\rho = \inf_A \sup_r \frac{cost_A(C_0,r)}{opt(C_0,r)}$, modulo a constant term.

Theorem 1. *In every metric space with at least $K+1$ points, no online algorithm for the K-server problem can have competitive ratio less than K.*

We wish to show there are request sequences of arbitrary high cost for A for which the online algorithm A has cost K times that of the optimal offline algorithm. We prove this lower bound result later below.

22.1 The upper bound

Now we consider the *double coverage* strategy, used for the online algorithm A to achieve the competitive ratio K. Let a_1, a_2, a_3 be ordered left to right on a horizontal line with a_2 closer to a_1 than a_2. Let servers s_1 and s_2 be at a_1 and a_3 respectively, initially, with no server at a_2.

Let serving requests come repeatedly alternating between a_2 and a_1. If we move only the (closest) server s_1 (which was intitally stationed at a_1) up and down between a_1 and a_2 for the sequence $a_2, a_1, a_2, a_1, \cdots$, we incur unbounded competive ratio for asympopically large strings of requests a_2, a_1. This is so because the offline algorithm would place servers s_1 and s_2 at a_1 and a_2 respectively, permanently, instead of fixing s_2 at a_3. In the *double coverage* strategy instead, we move both s_1 and s_2 towards a_2 by amount $d(a_1, a_2)$ on serving request a_2, and then move s_1 back to a_1 on serving request a_1. So we use travel cost at most 3 times of that used by the optimal offline algorithm, which moves s_2 only once to a_2 on the first serving request for a_2.

We continue to analyse the double coverage strategy whose suggested ratio is 3 for $K = 2$ servers. Note that consecutive configurations C_t, C_{t-1} differ only in r_t i.e., $C_t = C_{t-1} \cup \{r_t\}$.

The scenario where servers are moved only to service requests directly is called *lazy*. The double coverage algorithm in that sense is not lazy. A non-lazy algorithm can however be *memory-less* like the double coverage algorithm since decisions are based only on the current configuration. Let us use potential $\Phi(C_t, C'_t)$ where C stands for the online algorithm and C' for the offline algorithm. Let $cost(t)$ and $opt(t)$ be the costs to service r_t by online and offline methods at the instant t. We need to show that

$$cost(t) - K * opt(t) \leq \Phi(C_{t-1}, C'_{t-1}) - \Phi(C_t, C'_t) \tag{6}$$

Adding for m steps we have

$$\sum_{t=1}^{m} cost(t) - K * \sum_{t=1}^{m} opt(t) \leq \Phi(C_0, C'_0) - \Phi(C_m, C'_m) \tag{7}$$

We can drop $\Phi(C_m, C'_m)$ without disturbing upper bounding so that we have

$$\sum_{t=1}^{m} cost(t) - K * \sum_{t=1}^{m} opt(t) \leq \Phi(C_0, C'_0) \tag{8}$$

which gives the competitive ratio of at most K.

Let us use the offline algorithm to respond to r_t first and then the online algorithm.

1. $C'_{t-1} \leftarrow C'_t$, whereas C_{t-1} is unchanged.
2. $C_{t-1} \leftarrow C_t$ where C'_t has already reached a server to location r_t.

We define the potential function as

$$\Phi(C_t, C'_t) = K * d(C_t, C'_t) + \sum_{a_i, a_j \in C_t} d(a_i, a_j) \tag{9}$$

$d(C_t, C'_t) \leftarrow$ weight of the minimum weight bipartite matching in K_{C_t,C'_t}, the complete bipartite graph where servers of the offline and online algorithm form the two vertex sets C_t and C'_t.

To prove inequality (1) we do the two transitions of [1], the offline algorithm, and then [2], the online algorithm.

$cost(t) - K * opt(t) \leq \Phi(C_{t-1}, C'_{t-1}) - \Phi(C_t, C'_t)$

Wherever $cost(t)$ is more than $K * opt(t)$, there is a *balancing payment* from fall in the potential function. Observe that $d(C_t, C'_t)$ is simply $\sum_{i=1}^{K} d(s_i, a_i)$ for the scenario of straightline geometry. For the offline algorithm movement of servers for the request r_t, we have the following equations for potential functions for transition [1].

$$\Phi(C_{t-1}, C'_t) = K * d(C_{t-1}, C'_t) + \sum_{a_i, a_j \in C_{t-1}} d(a_i, a_j) \tag{10}$$

$$\Phi(C_{t-1}, C'_{t-1}) = K * d(C_{t-1}, C'_{t-1}) + \sum_{a_i, a_j \in C_{t-1}} d(a_i, a_j) \tag{11}$$

By the definition of Φ in Equation 9 and from Equations 10 and 11 we deduce

$$\Phi(C_{t-1}, C'_t) - \Phi(C_{t-1}, C'_{t-1}) = K * [d(C_{t-1}, C'_t) - d(C_{t-1}, C'_{t-1})] \tag{12}$$

Now by the triangle inequality
$d(C_{t-1}, C'_t) \leq d(C_{t-1}, C'_{t-1}) + d(C'_{t-1}, C'_t)$
and inequality 12 we have

$$\Phi(C_{t-1}, C'_t) \leq \Phi(C_{t-1}, C'_{t-1}) + K * d(C'_{t-1}, C'_t) \tag{13}$$

We will remember Inequality 13 for future use to prove Inequality 6.

Suppose we show for the online movement that

$$\Phi(C_t, C'_t) \leq \Phi(C_{t-1}, C'_t) - d(C_{t-1}, C_t) \tag{14}$$

Combining Equation 13 and 14 we get
$\Phi(C_t, C'_t) + d(C_{t-1}, C_t) \leq \Phi(C_{t-1}, C'_t) \leq \Phi(C_{t-1}, C'_t)$ or
$d(C_{t-1}, C_t) - K * d(C'_{t-1}, C'_t) \leq \Phi(C_{t-1}, C'_{t-1}) - \Phi(C_t, C'_t)$

However, $d(C_{t-1}, C_t) = cost(t)$ and $d(C'_{t-1}, C'_t) = opt(t)$. So we have established Inequality 6. Therefore, Inequality 7 follows and we are done. Finally, to show Equation 14 for movement of online steps, we do as follows.

Let us account the cost of the online algorithm for moving survers at the request r_t. Again, by the definition of Φ, we have $\Phi(C_t, C'_t) = K \times d(C_t, C'_t) + \sum_{a_i, a_j \in C_t} d(a_i, a_j)$ and $\Phi(C_{t-1}, C'_t) = k \times d(C_{t-1}, C'_t) + \sum_{a_i, a_j \in C_{t-1}} d(a_i, a_j)$. Observe that if r_t is a point between two online servers s_i and s_{i+1}, then one of them moves towards its matching point of the offline configuration and the other server may move away from its matching offine server an equal distance. So, their total contribution does not increase the matching, i.e., $d(C_t, C'_t) - d(C_{t-1}, C'_t) \leq 0$. Without loss of generality assume that $d(s_i, r_t) \leq d(s_{i+1}, r_t)$. Since s_i and s_{i+1} move towards r_t by the same distance, $\sum_{a_i, a_j \in C_t} d(a_i, a_j) - \sum_{a_i, a_j \in C_{t-1}} d(a_i, a_j)$ is reduced by $2d(s_i, r_t)$. So, $\Phi(C_t, C'_t) - \Phi(C_{t-1}, C'_t) \leq 2d(s_i, r_t)$, or $\Phi(C_t, C'_t) \leq \Phi(C_{t-1}, C'_t) - 2d(s_i, r_t)$, or $\Phi(C_t, C'_t) \leq \Phi(C_{t-1}, C'_t) - d(C_{t-1}, C_t)$, where $d(C_{t-1}, C_t)$ represents the change in the distance between online servers. This is the very Inequality 14 for this case.

Now consider the other case where r_t lies outside the interval of the K servers, and only one server (say, s_1) moves to r_t. Here, the first term of the potential decreases by $K \times d(s_1, r_t)$, because s_1 moves closer to its matching point a_1. The second term of the potential increases by $(K-1) \times d(s_1, r_t)$ as the distance to s_1 from s_2, s_3, ... , s_k increases by $d(s_1, r_t)$. The difference of these two terms is $d(s_1, r_t)$, which is equal to $d(C_{t-1}, C_t)$. So, in this second case too, $\Phi(C_t, C'_t) \leq \Phi(C_{t-1}, C'_t) - d(C_{t-1}, C_t)$, that is, Inequality 14. This completes the proof.

22.2 The lower bound

23 Minimum Knapsack Problem

We wish to solve the ILP.

$$\min \sum_{i \in I} s_i x_i \text{ such that } \sum_{i \in I} v_i x_i \geq D,\ x_i \in \{0,1\} \qquad \forall i \in I$$

View each $A \subseteq I$, such that $v(A) < D$, where we may achieve an additional value of $D_A = D - v(A)$. So, from v_i, $i \notin A$, we seek only $min(v_i, D_A)$. So, if $v(A) < D$ and $v(X) \geq D$ then we observe that

$$\sum_{i \in X \setminus A} v_i^A \geq D_A$$

We may therefore write

$$\sum_{i \in I} x_i s_i$$

$$\text{such that } \sum_{i \in I \setminus A} v_i x_i \geq D_A, \forall A \subseteq I$$
$$x_i \in \{0,1\}, \forall i \in I$$

LP relaxation

$$\sum_{i \in I} x_i s_i$$

$$\text{such that } \sum_{i \in I \setminus A} v_i^A x_i \geq D_A, \forall A \subseteq I,\ x_i \geq 0, \forall i \in I$$

So, for any $A \subseteq I$, the deficiency, if any can be satisfied by $I \setminus A$. The dual LP is

$$\max \sum_{A : A \subseteq I} D_A y_A$$

$$\text{such that } \sum_{A \subseteq I, i \notin A} v_i^A y_A \leq s_i, y_A \geq 0, \forall A \subseteq I, \forall i \in I$$

On having collected a set $A \subseteq I$ already, we need to pick up the next item $i \in I$ and adding i to A, i.e., we update $A \leftarrow A \cup \{i\}$. This goes on until $v(A) \geq D$, when we output $X \leftarrow A$.

Initially $A = \phi$, and therefore any $i \in I$ can be selected. We show that the approximation ratio is 2.

So, when X is returned, $X \subseteq I$, and l was the last item selected, then

$$v(X) \geq D \text{ and } v(X \setminus \{l\}) < D$$

So,

$$\sum_{i\in X} s_i = \sum_{i\in X}(\sum_{A\subseteq I, i\notin A} y_A v_i^A)$$

$$= \sum_{A\subseteq I} y_A \sum_{i\in XnA} v_i^A \tag{15}$$

Now $v_i < D - v(A) = D_A$ in all but the last iteration, i.e., $v_i^A = min(v_i, D_A) = v_i$ when A was the set of items.

So,

$$\sum_{i\in XnA} v_i^A = v_l^A + \sum_{i\in XnA, i\neq l} v_i^A = v_l^A + v(X\setminus\{l\}) - v(A) \tag{16}$$

But $v_l^A \leq D_A$ and $v(X\setminus l) < D$

So that

$$v(X\setminus\{l\}) - v(A) < D - v(A) = D_A \tag{17}$$

So,

$$v_l^A + v(X\setminus l) - v(A) < 2D_A \tag{18}$$

$$\sum_{i\in X} s_i = \sum_{A\subseteq I} y_A \sum_{i\in X\setminus A} v_i^A < 2\sum_{A\subseteq I} D_A y_A \leq 2OPT \tag{19}$$

24 Hardness of approximation

The following are MAXSNP-hard problems.

MAXE3SAT: Given a set of clauses with 3 literals each, find a truth assignment that satisfies the maximum number of clauses.

MAX2SAT: Differs from MAXE3SAT with at most 2 literals per clause.

24.1 Hardness of k-centre and TSP approximation

24.1.1 Definitions and notation

For a minimization problem an α-approximation algorithm is a polynomial time algorithm with a performance guarantee of α. This guarantee is that of delivery of a solution whose value is at most α times the optimal value for the problem. We know that the vertex cover problem has a performance gurantee of $\alpha = 2$.

24.1.2 The k-centre problem

From Theorem 2.4 in [7] we know that the k-centre problem has no α-approximation algorithm for $\alpha < 2$ unless P=NP. The dominating set problem is used for proving this result. A reduction from the dominating set problem shows that we can find a dominating set of size at most k if and only if an instance of the k-center problem (in which all distances are either 1 or 2), has optimal value 1.

The graph whose dominating set we wish to compute is translated into a graph where the missing edges are edges now with weight 2 and the original edges have weight 1. Such a construction obeys the triangle inequality.

24.1.3 The travelling salesman problem (TSP)

Theorem 2.9 from [7] is interesting. For any $\alpha > 1$, there does not exist an α-approximation algorithm for the traveling salesman problem (TSP) on n cities, provided P is not the same as NP. In fact, the existence of an $O(2^n)$- approximation algorithm for the TSP would similarly imply P = NP. This result follows from the hardness of the Hamiltonian cycle problem. If we had an α-approximation algorithm for the TSP problem for some $\alpha > 1$ then we could invoke this algorithm with weights 1 for all edges and get a yes answer for such an input if and only if the graph had a Hamiltonian cycle.

24.1.4 The bin-packing problem

Given an integer T and integral weights in the range 0 through T with a total sum of weights $2T$, we wish to partition these weights into two sets so that each set has a sum exactly T. This partition problem is a well known NP-hard problem. We may view this problem as a 2 bins bin-packing problem. Is there an α-approximation algorithm for this bon-packing problem for any $\alpha < \frac{3}{2}$? Since the optimal value is 2 bins for a yes instance of this partition problem, any such approximation algorithm would give a yes answer for the number of required bins as strictly less than 3, that is, with 2 bins, solving the partition problem in polynomial time.

24.2 Algorithms for maximum satisfiability [1]

24.3 Hardness of approximation for MAXE3SAT and MAX2SAT

24.3.1 L-reductions

Now we consider reductions from a problem Π to another problem Π' such that if there is an approximation algorithm with performance guarantee α for problem Π', then there is an approximation algorithm with performance guarantee $f(\alpha)$ for problem Π, where f is some function. These results yield hardness theorems via the contrapositive, that is, if there is no $f(\alpha)$-approximation algorithm for problem Π unless P = NP, then there is no α-approximation algorithm for problem Π' unless P = NP.

24.3.2 L-reduction from MAXE3SAT to MAX2SAT

From Theorem 5.2 of [7] we know that there can be no ρ-approximation algorithm for MAXE3SAT for constant $\rho > \frac{7}{8}$, unless P = NP. Suppose we show that given an α-approximation algorithm for the MAX2SAT problem, we have a $(\frac{55}{7}\alpha - \frac{48}{7})$-approximation algorithm for the MAXE3SAT problem, then we can solve for α satisfying $\frac{55}{7}\alpha - \frac{48}{7} > \frac{7}{8}$ to conclude Theorem 16.3 in [7] that there exists no α-approximation algorithm for the MAX2SAT problem for constant $\alpha > \frac{433}{440}$ unless P = NP.

24.3.3 The L-reduction approximation bound

See L-reduction definition in Chapter 16 of [7]. Such a reduction from a problem Π to Π' requires defining an instance I' of Π' from an instance I of Π, so that (i) $OPT(I') \leq a \times OPT(I)$, and (ii) $|OPT(I) - V| \leq b \times |OPT(I') - V'|$, where given a solution V' of I' we can compute in polynomial time a solution V of I. Assuming I' is α-approximate and we have maximization to be done, we can say $|V'| \geq \alpha \times OPT(I')$. Using these three inequalities involving a, b and α, we see that $|V| \geq OPT(I) - b(OPT(I') - |V'|) \geq OPT(I) - b(1-\alpha)OPT(I') \geq OPT(I)(1 - ab(1-\alpha))$. This means we have an $(1 - ab(1-\alpha))$-approximate algorithm for Π. Verify that $a = 55/7$ and $b = 1$ give approximation ratio $(1 - 55/7(1 - 433/440)) = (440 - 55)/440 = 385/440 \geq 7/8$, as in Theorem 16.3 of [7].

For an α-approximate minimization problem Π, we say $|V'| \leq \alpha OPT(I')$ and we can write $|V| - OPT(I) \leq b(|V'| - OPT(I'))$, so that $|V| \leq OPT(I) + b(\alpha OPT(I') - OPT(I')) \leq OPT(I) + ab(\alpha - 1)OPT(I)) = OPT(I)(1 + ab(\alpha - 1))$, rendering Π' to be $(ab(\alpha - 1) + 1)$-approximate. We will use this bound for the L-reduction from vertex covering to the steiner tree problem.

24.4 L-reduction for the independent set problem as in [7]

We discuss an L-reduction from MAXE3SAT to the maximum independent set problem as in [7]. Given an E3SAT instance I with m clauses, we create a graph with $3m$ nodes, one for each literal in the E3SAT instance. For any clause, we add edges connecting the nodes corresponding to the three literals in the clause. Furthermore, for each node corresponding to a literal x_i we add an edge connecting this node to each node corresponding to the literal x_i' . Call this instance I' for the maximum independent set problem.

Observe that given any solution V' to the independent set instance, we can obtain a solution V to the E3SAT instance by setting to true any x_i whose corresponding literal node is in the independent set and to false any x_i' whose corresponding literal node is in the independent set. This leads to a *consistent* assignment to the variables since there is an edge between each x_i node and each x_i' node, so that only one of the two kinds of nodes can be in the independent set. If for some variable x_j neither type of node is in the independent set, then we set x_j arbitrarily to true or false truth value. Notice that at most one literal node can be in the independent set for each clause as each clause gives a triangle in the graph; thus we satisfy at least as many clauses as there are nodes in the independent set, yielding the inequality $|V| \geq |V'|$.

Similarly, given any solution to the E3SAT instance, for *each satisfied clause* we can pick one of the literals that satisfies the clause (that is, a positive literal x_i set true or a negative literal x_i' set false) and put the corresponding node in the independent set for an independent set of size at least the number of satisfied clauses. Thus $OPT(I) = OPT(I')$, and for any solution V' to the independent set instance we can get a solution V to the E3SAT instance with $|V| \geq |V'|$. This is an L-reduction with parameters $a = b = 1$, so that any α-approximation algorithm for the maximum independent set problem results in an α-approximation algorithm for the MAXE3SAT problem. Here, as defined in Section 24.3.3, $a = 1$ and $b = 1$ since $OPT(I) - |V| \leq OPT(I') - |V'|$.

24.5 Gap preserving reductions

An example of a minimization problem: Approximating vertex covering

A formula ϕ of SAT is mapped by a *(gap) reduction* to a graph $G = (V, E)$ such that ϕ is satisfiable implies G has a vertex cover of size at most $\frac{2}{3}|V|$, and ϕ is not satisfiable implies the *smallest vertex cover* in G is of size more than $\alpha\frac{2}{3}|V|$, where $\alpha > 1$ is a fixed constant. (Recall earlier we studied the usual NP-hardness proof of vertex covering where $\alpha = 1$.) We now assume the existance of a polynomial time α-approximation vertex covering algorithm. In the above setting of the given gap reduction, if the algorithm finds a vertex cover of size at most $\alpha\frac{2}{3}|V|$ in G then we can deduce that ϕ is indeed satisfiable, and unsatisfiable otherwise. In such a scenario we would be settling satisfiability and thus be having P=NP.

The generalization for minimization problems

If Π is a minimization problem, we want the reduction to satisfy (i) ϕ is satisfiable implies $OPT(x) \leq f(x)$, and (ii) ϕ is not satisfiable implies $OPT(x) > \alpha(|x|)f(x)$. Here, x is an instance of Π, f is a function of the instance x and α is a function of the size $|x|$ of the instance x. In this case, $\alpha(|x|) \geq 1$. The gap, $\alpha(|x|)$, is the hardness factor in the gap-introducing reduction for the NP-hard problem. The instance x of Π is created from the formula ϕ for SAT in polynomial time.

The generalization for maximization problems

If Π is a maximization problem, we want the reduction to satisfy (i) ϕ is satisfiable implies $OPT(x) \geq f(x)$, and (ii) ϕ is not satisfiable implies $OPT(x) < \alpha(|x|)f(x)$. Here, x is an instance of Π, f is a function of the instance x and α is a function of the size $|x|$ of the instance x. In this case, $\alpha(|x|) \leq 1$. The gap, $\alpha(|x|)$, is the hardness factor in the gap-introducing reduction for the NP-hard optimization problem.

24.6 Hardness of art gallery problems

The papers of Lee and Lin (1986) [12] and that of Eidenbenz [11] are relevant.

24.6.1 NP-hardness of the vertex guarding problem

In the Lee and Lin work [12], we have a special vertex W that sees regions defined by the $3m$ literal patterns and the $2n$ rectangles in the n variable patterns. So, W has to be one of the $K = 3m+n+1$ guards of a guard set for the polygon. Thus we now only need to account for $3m+n$ gaurds, which should help us work out a truth assignment for the formula C. The $3m+n$ distinguished points are such that any vertex that covers one of them cannot cover any other distinguished point. So, at least $3m+n$ vertex guards are necessary. We show that $3m+n$ guards are sufficient along with W to guard the entire polygon. The distinguished points in literal patterns of clause $C = A+B+D$ in clause pattern C_h, will require (in literal pattern A), either a_{h1} or a_{h3} to be included in the guard set T. Similarly, one of two guards must be in T for literals B and D for each clause C. Which vertices will be the $3m$ guards in L from the clause patterns will certainly depend on the construction of variable patterns, with their spikes. Furthermore, the g_{h1}, g_{h2}, g_{h3} vertices, for all $h = 1, 2, ..., m$,

will need to be guarded by the $3m$ clause pattern guards. Here, Lemma 2 of [12] is used; the seven triples called "three-vertex covers" correspond to the seven possible ways a 3-literal cluase can be satisfied.

24.6.2 APX-hardness of the vertex guarding problem

The promise problem

The *promise* problem of 5-OCCURENCE-3-SAT (5-O-3-S henceforth) is where we are given an instance I of m clauses in n variables so that I is promised to be either *satisfiable* ($OPT(I) = m$) or at most $m(1-4\varepsilon)$ clauses are satisfied [11]. So, if we have a polynomial time $(1-4\varepsilon)$-approximation algorithm for this promise version of 5-O-3-S and we find an approximate solution with more than $m(1-4\varepsilon)$ clauses satisfied then we can conclude that $OPT(I)$ is indeed m, and less than m otherwise. Since 5-O-3-S MAXSNP-complete, this discimination between $OPT(I)$ being m or less than m would imply P=NP [11].

The gap preserving reduction from 5-OCCURENCE-3-SAT to PG [11, 13]

Now let f be a reduction from the *promise* problem 5-O-3-S to PG (point guards) such that for $I' = f(I)$ (i) $OPT(I) = m$ imples $OPT(I') \leq 3m+n+1$, and (ii) $OPT(I) \leq m(1-4\varepsilon)$ implies $OPT(I') \geq 3m+n+1+\varepsilon m$.

Assume that we can achieve an approximation ratio of $\frac{3m+n+1+\varepsilon m}{3m+n+1}$ for the minimization problem PG.

Suppose $OPT(I) = m$ then $OPT(I') \leq 3m+n+1$ by (i). The approximation algorithm will compute a guard set for I' with at most $3m+n+1+\varepsilon m$ guards, which, by the contrapositive of (ii) (for small enough ε), would tell the algorithm that $OPT(I) > m(1-4\varepsilon)$; by the *promise* for the 5-O-3-S problem, this would mean $OPT(I) = m$.

Suppose $OPT(I) \leq m(1-4\varepsilon)$. Then (ii) implies $OPT(I') \geq 3m+n+1+\varepsilon m$. Here the algorithm will find a solution which is at least as large as $OPT(I')$ and thus using the contrapositive of (i), we have $OPT(I) < m$.

The approximation ratio is $\frac{3m+n+1+\varepsilon m}{3m+n+1} \geq 1+\frac{\varepsilon}{7}$, as shown in [11], implying APX-hardness of PG.

References

[1] C. H. Papadimitriou, Computational Complexity, Addison-Wesley, 1994.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, Introduction to algorithms, Second Edition, Prentice-Hall India, 2003.

[3] J. Hopcroft and J. D. Ullman, Introduction to Automata, Languages and Computation, Addison-Wesley, 1979.

[4] J. Kleinberg and E. Tardos, Algorithm design, Pearson education, 2006.

[5] J. H. van Lint and R. M. Wilson, A course in combinatorics, Cambridge University Press, 2002.

[6] Susanne Albers, BRICS Mini-course on Competitive Online Algorithms, Aarhus University, August 27-29, 1996.

[7] David P. Williamson and David B. Shmoys, The design of approximation algorithms, Cambridge University Press, 2010.

[8] V. Vazirani, Approximation algorithms, Springer, 2003.

[9] Dorit S. Hochbaum (Editor), Approximation algorithms for NP-hard problems, Thomson, 1995.

[10] Robin J. Wilson, Introduction to Graph Theory, 4th Edition, Pearson, 2007.

[11] Stephan Eidenbenz, Inapproximability results for guarding polygons without holes, ISAAC'98, LNCS 1533, pp. 427-437, 1998.

[12] D. T. Lee and A. K. Lin, Computational Complexity of Art Gallery Problems, in IEEE trans. Info. Th, pp. 276-282, IT-32 (1986).

[13] S. Eidenbenz, C. Stamm, and P. Widmayer, Inapproximability Results for Guarding Polygons and Terrains, Algorithmica (2001) 31: pp. 79113.