Approximation and online algorithms: CS60023: Spring 2024

Instructor: Sudebkumar Prasant Pal

IIT Kharagpur

email: spp@cse.iitkgp.ac.in

April 14, 2024

Preliminary Background

- Suppose we have an optimization problem (i.e. computing a minimum sized vertex cover or a maximum cardinality stable set) where a certain parameter must be minimized or maximised.
- Say OPT is the value of the optimal solution 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.
- In a minimization problem $\frac{v}{OPT} \geq 1$

Minimization problems

- OPT is generally not known but we may know a lower bound m, where $m \leq OPT$, yielding an upper bound for the approximation ratio $\frac{v}{OPT} \leq \frac{v}{m}$.
- Thus we can estimate an upper bound i.e. $\frac{v}{m}$ on the approximation ration $\frac{v}{OPT}$ for Algorithm A if we know the lower bound m; this bound is better if we have tighter value for m.
- Note that in maximization problems, we can in a similar manner define the approximation ratio as $\frac{v}{OPT} \leq 1$.
- The smaller the upper bound estimate for OPT, the better would the lower bound on the approximation ratio.

Large DAG subgraphs of directed graphs

- Let us consider the problem of computing a large directed acyclic subgraph in a given *directed graph* $G(V, E)$.
- If OPT is number of edges in the maximum size DAG, then $OPT \leq e$.
- If we partition the set E of edges into two sets so that each set induces a DAG, then we can choose the bigger one, ensuring at least $\frac{e}{2} \geq \frac{OPT}{2}$ edges are selected in the large DAG.
- Naturally each set in the partition must induce a DAG. To achieve this requirement, we use the total ordering of integers after arbitrarily numbering the vertices from 1 through $n = |V|$.
- Then we take the *forward edges* $(i < j)$ in one set and the *backward edges* $(i > j)$ in the other set. Observe that both these sets of edges constitutes DAGS; so we can pick the larger one.[1]

[1]See Problem 1.9 in page 7 in [6].

Large cuts for undirected graphs

- The size of a cut in a graph $G(V, E)$ cannot exceed $e = |E|$. If OPT is the size of the max sized cut then $OPT \leq e$
- We can start with just about any cut and then keep improving it and stop this incremental step when we cannot increase the cut size anymore.
- 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 $\frac{2e}{2} = e$.

Large cuts for undirected graphs (cont.)

- 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 $\frac{OPT}{2}$.

Minimum maximal matchings

- See Problem 1.2 on page 8 in [6]
- 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; that is, the $2m$ vertices of M *hit* all edges in 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 $\frac{|M|}{|M'|} = \frac{m}{OPT} \geq \frac{m}{2m} = \frac{1}{2}$.

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 [6].
- 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. Why?
- So, the size OPT of the minimum vertex cover is no lesser than this lower bound. So, $m \leq 2 \times OPT$.

Vertex cover from matching: Ratio factor two

- In the next section 7, 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.

Vertex covering using a large cut

- The following problem is due to Vishnoi, given as Exercise 2.5 on page 23 of [6]. 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 therefore compute the exact vertex cover for this bipartite graph in polynomial time. How? (Recall the Konig-Egervary theorem for bipartite graphs.)
- 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 as follows.

Vertex covering using a large cut (cont.)

- 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 $OPT_H + (\log \frac{\Delta}{2})OPT_{G \setminus H} \leq OPT_G + (log \Delta - 1)OPT_G =$ $(\log \Delta)OPT_G$ for the whole of G.
- Section 2.4 Exercises 2.1, 2.2, 2.3 and 2.6 from pages 22-24 in [6]

The machines and jobs: A trivial lower bound

- We schedule n jobs in arbitrary arrival order on m identical machines.
- For the next arrival (in the arbitrarily selected sequence of arrivals of the n jobs), we assign the job to the least so far loaded machine.
- We know that the completion time or *makespan* can never be less than the processing time of the job with the largest processing time.
- So, OPT is at least $max_{i=1}^{n}\{p_i\}$, where p_i is the processing time of the job i, $1 \leq i \leq n$.

The second (non-trivial) lower bound for *OPT*

- Additionally, let us assume for the sake of contradiction that OPT is strictly less than the average processing time $\frac{1}{m}\sum_{i=1}^{n} p_i$.
- Then, we can complete all the n jobs in a sequential simulation on only one machine, spending a total time of $OPT \times m < \sum_{i=1}^{n} p_i$, which is less than that required to complete all the jobs sequentially on a single machine, a contradiction.
- So, we conclude that $OPT \geq \frac{1}{m}\sum_{i=1}^{n} p_i$.
- Therefore, OPT is at least the larger of $\frac{1}{m}\sum_{i=1}^{n} p_i$ and $max_{i=1}^{n}\{p_i\}$.

Upperbounding the makespan by bounding the start time of the last ending job.

- We certainly use both these lower bound for OPT in our analysis of the factor two algorithm.
- We must focus on the job p_j that ends last but may have started quite early.
- However, when p_j was scheduled on (say) machine M_i, all the other $m-1$ machines must have been busy.
- This is due to the assignment rule that we must assign the next arrival to a machine that is least loaded at the time $start_j$ of arrival (and assignment to machine M_i).
- So, as we mentioned already, $start_j$ must be quite small, and we now argue that it is indeed not too large.

The ratio factor two bound for makespan: Bounding the start time (contd.)

- Suppose, for the sake of contradiction, we assume that $start_j$ is strictly greater than the average processing time $\frac{1}{m}\sum_{i=1}^{n} p_i$.
- So, since all the machines were busy just until $start_j$, and therefore fully utilized, we have $m \times start_j > \sum_{i=1}^{n} p_i$, and so a simulation on a single machine would complete all jobs, a contradiction because we have a p_j-sized job yet to be processed, a contradiction.
- Therefore, $start_j \leq \frac{1}{m}\sum_{i=1}^{n} p_i \leq OPT$.
- We have $p_j \leq OPT$ as well.
- Therefore, the makespan computed by Algorithm 10.2 in [6] is at most $2.OPT$. Why?

The “local search” offline version of makespan

- Assume that we have all the n jobs' data p_j, and that we have already made an arbitrary assignment of the jobs to the m processors.
- We can improve by reassigning jobs to processors carefully.
- One such reassignment strategy is to reassign the currently last ending job b to a machine M if that helps finish job b earlier than time $C_b = start_b + p_b$.
- This improvement is therefore possible if the “total duration” of the machine M in the current schedule is smaller than $start_b = C_b - p_b$.

Offline makespan (contd.)

- The two lower bounds for OPT hold just as they hold for the online version or the incremental version.
- Also, if no improvement is possible, then we take the current $C_b = start_b + p_b$ as our upper estimate for OPT, yielding again the same upper bound of at most $2.OPT$ for C_b.
- How can we improve this approximation ratio to $2 - \frac{1}{m}$?

The notion of prices in the primal integral 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 dual solution objective function value as a lower bound

- 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 paid* 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 no set is *overpacked* (all constraints of the dual LP are satisfied).

- As in Section 15.1 of [6], 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 [6].
- Primal complementary slackness conditions: Let $\alpha \geq 1$. For each $1 \leq j \leq n$: either $x_j = 0$ or $\frac{c_j}{\alpha} \leq \sum_{i=1}^{m} a_{ij} y_i \leq c_j$.
- Dual complementary slackness conditions: Let $\beta \geq 1$. For each $1 \leq i \leq m$: either $y_i = 0$ or $b_i \leq \sum_{j=1}^{n} a_{ij} x_j \leq \beta b_i$.
- The design of Algorithm 15.2 is based on Proposition 15.1 leading to Theorem 15.3.
- Proposition 15.1: If x and y are primal and dual feasible solutions satisfying the conditions stated above then $\sum_{j=1}^{n} c_j x_j \leq \alpha\beta \sum_{i=1}^{m} b_i y_i$.
- 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 [6] (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. $\alpha\beta \sum_{i=1}^{m} b_i y_i$ 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 collection at all the primal variables is at least the l.h.s $\sum_{i=1}^{n} c_j x_j$ of Proposition 15.1.
- Note that the total amount $\alpha y^T A x$ sent by the dual and the total amount $\alpha x^T A^T y$ received by the primal is the same quantity.

- The interesting example when $\alpha = 1$ and $\beta = f$ is that of the weighted set cover problem where each element is in at most f sets.

- We are guaranteed an f-factor algorithm if the post-condition satisfied by the execution of the algorithm satisfies the relaxed complementary slackness conditions, due to Proposition 15.1.

- The algorithm picks sets in the set cover one by one by satisfying the equality constraint due to the unit value of α in the dual inequalities.

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).

The approximation algorithm with ratio factor two for vertex covering (cont.)

- 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.

The approximation algorithm with ratio factor two for vertex covering (cont.)

- *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.

The approximation algorithm with ratio factor two for vertex covering (cont.)

- 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.
- So, 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 [6]).
- As mentioned earlier, the *maximal matching* is such that none of its supersets enjoys the same property.

The approximation algorithm with ratio factor two for vertex covering (cont.)

- 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 can indeed have some savings.

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.

Tight example for the vertex cover algorithm

- In the case of the factor two algorithm using maximal matchings for vertex covering (in Section 7), 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 [6]), 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.

Tight example for the vertex cover algorithm (cont.)

- So, this family of infinite graphs provides what we call a *tight (asymptotic) example* for the specific algorithm. Tight examples often give critical insights into the functioning of an algorithm and often lead to ideas for the design of other algorithms that can achieve improved guarantees.
- However, we do have vertex covers of size n in $K_{n,n}$!!! A smarter algorithm might be able to tackle special cases where smaller than $2n$-sized vertex covers can be discovered.

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 7. 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 [6]).
- *So, we observe that by simply using the lower bounding scheme of maximal matchings, we cannot improve the approximation ratio.*

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

- As in the case of vertex cover in Section 7, 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.*

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 $1 + \epsilon$.
- The optimal cover has weight $1 + \epsilon$ but the greedy algorithm computes a set cover with weight $H(n)$; in each iteration, the cost effectiveness of the last set is higher than those of the previous sets and lower than those of the sets not yet selected. This is an example where the $H(n)$ upper bound is approached as ϵ approaches zero.
- In Section 13.1 of [6] 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 [6].
- See Sections 29.7 and 29.9 of [6] for seeing why the obvious greedy algorithm is the best one can hope for.

Dual fitting for the constrained set multicover problem

- The discussion here on the *constrained set multicover* problem is from Section 13.2.1 in [6]. 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 $k \times c(S)$. Such permissible picking of a set multiple times is allowed in the less constrained problem *set multicover*.

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 $S \in \mathcal{S}$.

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 what we call *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 dual LP for the primal LP relaxation of the integer program (cont.)

- 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 $S \in \mathcal{S}$

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 picks a hitherto 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.

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 non-decreasing.
- Since cost-effectiveness is non-decreasing over iterations of selection of sets in the set cover, we have, for each element e, $price(e, 1) \leq price(e, 2) ... \leq price(e, r_e)$.

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 $\sum_{e \in U} r_e \alpha_e - \sum_{S \in \mathcal{S}} \beta_S = \sum_{e \in U} \sum_{j=1}^{r_e} price(e, j)$.

The dual solution (cont.)

- 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 $z_S = \frac{\beta_S}{H_n}$.

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 multiple 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.

Scaling and dual-fitting for satisfying the dual constraints (cont.)

- 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, $(\sum_{i=1}^{k} y_{e_i}) - z_S \leq \frac{c(S)}{H_n}(\frac{1}{k} + \cdots + \frac{1}{k-k'+1} + 1) \leq c(S))$.

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.

The k-centre problem

- 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 cluster center.
- Geometrically, the goal is to find k different *balls* covering all points so that the radius of the largest ball is as small as possible.
- In other words, the goal is to find a set S of the centers of k different balls of the same radius that cover all points in $V \setminus S$ so that the radius is as small as possible.

The k-centre problem (cont.)

- 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 set 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 *centres of clusters*, so that points closer to a centre in S than any other 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]

The k-centre problem (cont.)

- The approximation algorithm is simple: it selects an arbitrary point 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 point $i \in V \setminus S$ whose distance $d(i, S)$ to the points in S 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^*.

The k-centre problem (cont.)

- Assume that the algorithm has chosen only one vertex $u(B) \in S$ from a cluster B with centre $v(B) \in S^*$ of the optimal solution S^*.
- Furthermore, suppose only one vertex is selected in S from each cluster of the optimal solution S^*.
- Then vertices of S^* are in any case within a distance r of some cluster centre in S ! Why?
- Now consider any vertex $u \in V \setminus S^*$.
- Any $u \in V \setminus S^*$, within the radius r cluster B centred at its cluster centre vertex $v(B) \in S^*$, is within a distance of $2r$ from the cluster centre vertex $u(B) \in S$, due to triangle inequality.
- Now consider the other situation where one vertex $u(B)$ inside the cluster B is already selected in S and the algorithm still chooses another vertex $w(B)$ inside the cluster B as a center point in S.

The k-centre problem (cont.)

- Again, the distance between $u(B)$ and $w(B)$ is bounded by $2r$. Moreover, $w(B)$ must have been the furthest point from all points in S at the time it was selected in S, including $u(B)$, by the choice of the algorithm, and therefore, all the points are within a distance of $2r$ of some center point already selected in S.
- This argument holds even if the algorithm adds more points of B to S subsequently.

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 terminals from each other.
- The multiway cut problem asks for such a minimum weight cut. This presentation is from Section 4.1 of [6]. 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, solvable in polynomial time using network flows. We present a $2 - \frac{2}{k}$ approximation algorithm for this problem as follows for $k \geq 3$.
- For each $i = 1, ..., k$ do
  1. identify the terminals in $S \setminus \{s_i\}$ into a single vertex
  2. compute a minimum weight cut C_i for $(s_i, S - \{s_i\})$ using a network flow algorithm, and

Multiway cut (cont.)

3. discard the heaviest of these cuts. The output 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, $\sum_{i=1}^{k} w(A_i) = 2w(A)$.

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 [6].
- 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 $w(C) \leq (1-\frac{1}{k})\sum_{i=1}^{k} w(C_i) \leq (1-\frac{1}{k})\sum_{i=1}^{k} w(A_i) = 2(1-\frac{1}{k})w(A)$.

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 [6]
- This k-cut problem is a more general problem. The nice approximation bound in this problem requires a complicated analysis using Gomory-Hu trees.
- A k-cut 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.

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.

The Gomory-Hu tree and minimum weight cuts (cont.)

- 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.

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.
- 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.
- 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, $V_1, V_2, ..., V_k$.

Properties of any optimal k-cut A and the approximation algorithm for computing a k-cut (cont.)

- 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, $\sum_{i=1}^{k} w(A_i) = 2w(A)$.

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. 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 $V_1, V_2, ..., V_k$. 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 resective 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 k super-vertices.

Establishing the novel lower bound (cont.)

- Observe that this graph must be connected since T is itself connected.
- Throw edges away from this graph until a tree survives. Let $B' \subseteq B$ be the leftover edges in T'. Clearly, $|B'| = k - 1$ as the tree leftover has k vertices.
- The edges of B' define the required $k - 1$ cuts which are dominated by the $k - 1$ cuts from A.
- Assuming that A_k is the heaviest cut amongst the cuts of A. Imagine rooting tree of edges from B' at V_k.
- We now define a correspondence between the edges in B' and the sets $V_1, V_2, \ldots, V_{k-1}$: each edge corresponding to the set it comes out of in the rooted tree, going towards the parent.

Establishing the novel lower bound (cont.)

- 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 $w'(u, v)$.
- Since A_i is also a $u - v$ cut in G (but may not be the minimum such cut!), we therefore 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 $w(C) \leq \sum_{e \in B'} w'(e) \leq \sum_{i=1}^{k-1} w(A_i) \leq \sum_{i=1}^{k} (1 - \frac{1}{k}) A_i = 2(1 - \frac{1}{k}) w(A)$.

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

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.
- 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

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

- Adding for m steps we have

$$\sum_{t=1}^{m} cost(t) - K * \sum_{t=1}^{m} opt(t) \leq \Phi(C_0, C'_0) - \Phi(C_m, C'_m) \tag{2}$$

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

$$\sum_{t=1}^{m} cost(t) - K * \sum_{t=1}^{m} opt(t) \leq \Phi(C_0, C'_0) \tag{3}$$

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

  We define the potential function as

$$\Phi(C_t, C'_t) = K * d(C_t, C'_t) + \sum_{a_i, a_j \in C_t} d(a_i, a_j) \tag{4}$$

- $d(C_t, C'_t) \implies$ 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.
- **Offline algorithm movement** of servers for the request r_t.
- We have the following equations for potential functions for transition [1].

$$\Phi(C_{t-1}, C'_t) = K * d(C_{t-1}, C'_t) + \sum_{a_i, a_j \in C_{t-1}} d(a_i, a_j) \tag{5}$$

$$\Phi(C_{t-1}, C'_{t-1}) = K * d(C_{t-1}, C'_{t-1}) + \sum_{a_i, a_j \in C_{t-1}} d(a_i, a_j) \tag{6}$$

- By the definition of Φ in Equation 4 and from Equations 5 and 6 we deduce

$$\Phi(C_{t-1}, C'_t) - \Phi(C_{t-1}, C'_{t-1}) = K * [d(C_{t-1}, C'_t) - d(C_{t-1}, C'_{t-1})] \quad (7)$$

- 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 Equality 7 we have

$$\Phi(C_{t-1}, C'_t) \leq \Phi(C_{t-1}, C'_{t-1}) + K * d(C'_{t-1}, C'_t) \quad (8)$$

- We will remember inequality 8 for future use to prove inequality (1).

- Suppose we show for the online movement that

$$\Phi(C_t, C'_t) \leq \Phi(C_{t-1}, C'_t) - d(C_{t-1}, C_t) \tag{9}$$

- Rewriting inequality 9 we get $\Phi(C_t, C'_t) + d(C_{t-1}, C_t) \leq \Phi(C_{t-1}, C'_t)$
- Substituting the RHS above from inequality 8, and moving the terms in the inequality, we get $d(C_{t-1}, C_t) - K * d(C'_{t-1}, C'_t) \leq \Phi(C_{t-1}, C'_{t-1}) - \Phi(C_t, C'_t)$
- But $d(C_{t-1}, C_t) = cost(t)$ and $d(C'_{t-1}, C'_t) = opt(t)$.
- So we have established inequality (1)
- Therefore, inequality (2) follows and we are done.

- Finally, to show inequality 9 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 9 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 9
- This completes the proof.

Facilities location

- The *uncapacitated facilities location problem* and *clustering problems* have been studied extensively, like the *k-median problem* and the *k-center problem*.
- Typically, in the given n-vertex graph, non-negative edge weights obey the triangle inequality.
- In the k-center problem we minimize the maximum distance from a *facility*, whereas in the k-median problem we minimize the total sum of distances from facilities.
- In both these problems we do not consider and costs for the facilities, unlike in the uncapacitated facilities location problem.
- The *uncapacitated facility location* problem is a combinatorial optimization problem. It has applications in setting up facility distribution centres.

Facilities location (cont.)

- 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 $T' \subseteq F$ so as to minimize the total cost of the facilities in T' and the cost of assigning each client $j \in D$ to some facility in T'.
- In other words, we wish to find $T' \subseteq F$ and a function f mapping clients to facilties, such that the following cost is minimised,

$$\sum_{i \in T'} f_i + \sum_{j \in D, f(j) \in T'} c_{f(j)j}$$

where the first part is called *facility cost* and the second part is called *assignment cost* or *service cost*.

Facilities location (cont.)

- This is an NP-hard problem and therefore we need to design approximation algorithms.

Integer programming formulation and its linear programing relaxation

- 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

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

Integer programming formulation and its linear programing relaxation (cont.)

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

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

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

$$x_{ij} \leq y_i$$

Integer programming formulation and its linear programing relaxation (cont.)

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

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

Integer programming formulation and its linear programing relaxation (cont.)

- The linear programming relaxation (LPR) from the ILP 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 program (LPR) can be summarized as follows:

$$\textit{minimize} \qquad \textstyle\sum_{i \in F} f_i y_i + \sum_{i \in F, j \in D} c_{ij} x_{ij} \tag{10}$$

-

$$\textit{subject to} \qquad \textstyle\sum_{i \in F} x_{ij} = 1, \qquad \forall j \in D, \tag{11}$$

$$x_{ij} \leq y_i, \qquad \forall i \in F, j \in D, \tag{12}$$

Integer programming formulation and its linear programing relaxation (cont.)

$$\begin{aligned} x_{ij} &\geq 0, && \forall i \in F, j \in D, \\ y_i &\geq 0, && i \in F. \end{aligned}$$

Lower bounding using dual linear programs

- The dual maximizing LP, which we will call DLP (corresponding to the minimizing primal LPR), is used to achieve as high lower bounds as possible for the primal ILP objective function.
- Typically, we may start any algorithm for computing a feasible solution for the ILP by initializing all primal ILP and DLP variables to zeros.
- In the course of the algorithm, primal ILP variables can be assigned only integral values whereas DLP variables can be assigned rational values.
- The respective values of the objectives functions for the ILP and the DLP is the approximation ratio achieved in the developing solution.
- We now discuss the formulation of a *dual linear program* (DLP) corresponding the relaxed linear program (LPR) as in [7].

Lower bounding using dual linear programs (cont.)

- If we ignore costs of facilities by setting $f_i = 0$ for all $i \in F$, the best strategy would be to open all the facilities and assign each client to its nearest facility. We introduce a variable v_j and set it as $v_j = \min_{i \in F} c_{ij}$ to denote the cost of connecting client j to its nearest facility.
- Observe that a lower bound for the primal integer program's objective function cost in an integral solution (of the ILP), is $\sum_{j \in D} v_j$ therefore; we certainly cannot have a better assignment of facilities.
- We can improve this lower bound estimate by considering non-zero facility costs as well, as follows.
- Each facility may be viewed as distributing its cost f_i, sharing it apportioned amongst the clients it provides service to, that is, $f_i = \sum_{j \in D} w_{ij}$, where each $w_{ij} \geq 0$.

Lower bounding using dual linear programs (cont.)

- A client j needs to pay this share only if it uses facility i. So, we can now set $v_j = \min_{i \in F}(c_{ij} + w_{ij})$.
- This can be enforced in a linear programming formulation with constraints $v_j \leq c_{ij} + w_{ij}$ (see inequality 15), for each client j (where i ranges over all facilities), with the objective function maximizing $\sum_{j \in D} v_j$, subject to further inequality 14.
- Observe that any feasible solution to this dual linear program therefore has objective function value lower bounding the cost of optimal primal objective function value for the (integral) facility location problem ILP.

Lower bounding using dual linear programs (cont.)

- We summarize the dual linear program (DLP) for the primal linear program relaxation (LPR) as:

$$maximize \sum_{j\in D} v_j \tag{13}$$

subject to

$$\sum_{j\in D} w_{ij} \leq f_i, \qquad \forall i, \in F \tag{14}$$

$$v_j - w_{ij} \leq c_{ij}, \qquad \forall i \in F, j \in D \tag{15}$$

$$w_{ij} \geq 0, \qquad \forall i \in F, j \in D \tag{16}$$

$$v_j \geq 0, \qquad \forall j \in D \tag{17}$$

The design of the algorithm: Phase I

- In Phase I of the algorithm, we first compute (i) a *maximal dual solution*, (ii) a tentative set T of facilities to be opened, and (iii) a temporary facilities mapping for clients, assigning a *connecting witness* facility for each client.
- In the second Phase II, we restrict the facilities allocated to a subset T' of T, reworking some assignments of facilites to clients, albeit some additional cost of connectivity, but well within the 3-factor limit (by virtue of triangle inequality).
- A *maximal dual solution* (v^*, w^*) is such that we cannot further enhance the value of any v_j^* and still work out a feasible assignment to variables w_{ij}^*.

The design of the algorithm: Phase I (cont.)

- For such maximal dual LP solutions, consider the definitions
  (a) of a client j *neighbouring* a facility i when $v_j^* \geq c_{ij}$ (edges (i, j) are called *tight* edges, and i and j are mutually *neighbours* of each other),
  (b) a *saturated* dual constraint Inequality 14 obeying equality when a facility i becomes *tight* or *paid up*, and
  (c) when it is said that a client j *contributes* to a facility i, or $w_{ij} > 0$; such edges (i, j) are called *special* edges.
- Furthermore, 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.
- We sketch the algorithm below as in [7].
- The algorithmic issues are as follows, providing intuition about its design, correctness and performance bound.

The design of the algorithm: Phase I (cont.)

- Suppose the largest w_{ij}^* satisfying the dual inequality 15 with equality, for some $i \in F$ and some $j \in D$, is non-zero.
- If such a w_{ij}^* is non-zero, we have $v_j^* > c_{ij}$. [So, (i, j) is both *special* as well as *tight*, as per the above definitions.]
- Due to the maximality of w_{ij}^*, we can set v_j^* to $c_{ij} + w_{ij}^*$, for the smallest such value over all $i \in F$, keeping the solution feasible for the dual LP.
- Such an $i \in F$ is called *saturated*, and is included in the set T of tentatively opened facilities if inequality 14 is satisfied.
- For such a set T we now argue, as in [7], that every client neighbours a facility in T.

Every client neighbours a facility in T

- First we note that the dual solution being maximal, it must be that $v_j^* = min_{i \in F}(c_{ij} + w_{ij}^*)$, for some $i \in F$.
- Otherwise, we must have $v_j^* < min_{i \in F}(c_{ij} + w_{ij}^*)$ for all $i \in F$, in which case we can enhance v_j^*, contradicting that we have a maximal dual solution.
- So, now suppose client j has no neighbour in T, that is, $v_j < c_{ij}$ for all $i \in T$.
- However, the dual solution being maximal, we must have v_j^* as the smallest of $c_{ij} + w_{ij}^*$ for some i, and if all such clients i are outside T then it must have been the case that for all such $i \in T \cap F$, $\sum_{k \in D} w_{ij}^* < f_i$.
- So, i was not selected to be in T.

Every client neighbours a facility in T (cont.)

- In this case however, we can enhance v_j^* and w_{ij}^*, without violating dual constraint inequalities 14 and 15. This contradicts that we had a maximal dual solution.
- We therefore conclude that all clients will have a neighbour in T once we have computed a dual maximal feasible solution at the termination of Phase I.

Summarizing Phase I

- In this phase we maintain feasibility in the dual solution and devleop a maximal dual solution (v^*, w^*) as already defined.
- We set $S = D$, the set of clients, and $T = \phi$, the set of temporarily selected facilities.
- We raise v_j's and w_{ij}'s uniformly until either (Case 1) some client $j \in D$ *neighbours* some facility $i \in T$, or (Case 2) some facility $i \in F$ becomes *tight*, or *paid for*, or *saturated*.
- Such clients $j \in S$ as in Case 1, that neighbour some facility $i \in T$ $(v_j \geq c_{ij})$, are removed from S.
- Such saturated facilities i as in Case 2 $(\sum_{j \in D} w_{ij} = f_i)$ are moved into the set T.
- Whenever a facility i is added to T, we remove all clients in the neighbouring set $N(i)$ of facility i from the set S.

Summarizing Phase I (cont.)

- When the set S becomes empty and each client neighbours some facility, and Phase I is terminated.
- 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 so that the complemetary slackness condition (i) 18, that is, $v_j - w_{ij} = c_{ij}$, resulting from dual constraint 15 will continue to hold.
- However, this raising of w_{ij} will not be necessary when the facility i is already *paid up* or *saturated* or *tight*, as per dual inequality 14 and complementary slackness condition (ii) 19.
- In this case, all client neighbours $j \in N(i)$ are also removed from S. Consequently, we also stop raising $w_{i'j}$ for any $i' \in F$, where $i' \neq i$ for clients j removed from S.

Summarizing Phase I (cont.)

- We observe that the maximality of the dual solution ensures that all clients have finally got *tight* edges to some facility, thereby acquiring a *connecting witness* as that facility.
- Some client j may have a connecting witness i with $w_{ij} = 0$. Other edges (i, j) will have w_{ij} non-zero, which we have already named as *special edges*.

Phase II

- Once the whole set S is exhausted and we have computed a maximal feasible DLP solution (v^*, w^*), we assign facilities from a set $T' \subseteq T$ to clients.
- Note one important point that each client has a neighbouring facility in T.
- This is due to the maximality of (v^*, w^*) in the Phase I process. We refer to the proof of this fact to section 7.6 of [7], as also elaborated in the above discussion.
- Once T is computed in Phase I, 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 also removed from T if both h and i are *contributed* to by some common client j, that is, if both w_{ij} and w_{hj} are positive.

Phase II (cont.)

- Therefore, finally *opening up* only the facilities surviving in T' will ensure contribution from each client to its respective assigned facility, the only facility to which that client contributes (see the genesis of the dual inequality 14).
- Finally, opened facilities from T' are assigned to all clients as follows.
- If a client $j \in D$ neighbours a facility $i \in T'$ then j is assigned to i and has connection cost c_{ij}, lower bounding v_j^*, that is, $v_j^* \geq c_{ij}$.
- Otherwise, we see due to Lemma 7.13 in [7] (and also as we discuss below) that although j does not have a neighbour in T', there is a facility $i \in T'$ such that $v_j^* \geq \frac{c_{ij}}{3}$.

The analysis: Relaxed complementary slackness

- The 3-factor approximation result of Theorem 7.14 of [7] follows from Lemma 7.13 of [7]; we explain these results in more detail now.
- We know that even if client $j \in D$ neighbours no facility in T', it does neighbour a facility in T, as argued above, by virtue of the method used to construct the set T.
- It turns out that such a client j neighbours some saturated facility $h \notin T'$ such that some other client contributed to both h and some facility $i \in T'$.
- The client j must have neighboured some $h \in T \setminus T'$ in the algorithm's execution when increasing v_j was stopped; it is known that j does not neighbour any facility in T'.
- How was $h \in T$ excluded from being in T'? Another client k was there that *contributes* to both h and another facility $i \in T'$.

The analysis: Relaxed complementary slackness (cont.)

- It is now our goal to show that the cost c_{ij} of assigning j to i is at most $3v_j$.
- In this context, view the client-facility pairs (j,h),(k,h) and (k,i), and the path along these three edges from client j to facility i, through facility h and client k.
- Clearly the cost of connecting j to i is $c_{ij} \leq c_{hj} + c_{hk} + c_{ik}$, by triangle inequality.
- To show $v_j \geq \frac{c_{ij}}{3}$, it is enough to show the v_j is at least as large as each of c_{hj}, c_{hk}, and c_{ik}.
- First, we note that $v_j \geq c_{hj}$ as j neighbours h.
- Second, we also show below that $v_j \geq v_k$. Therefore, v_k being at least as large as both of c_{ik} and c_{hk} (since k contributes to and thus also neighbours both h and i), we conclude that v_j at least as large as all the three of c_{hj}, c_{hk}, and c_{ik}.

The analysis: Relaxed complementary slackness (cont.)

- Now we show that $v_j \geq v_k$. We know that v_j stopped increasing when it neighboured a facility in T.
- Since j neighbours $h \in T$, we understand that h must have already been in T or must have been included in T when v_j stopped increasing.
- Now since k *contributes* to h, and therefore also neighbours h, v_k too must have stopped increasing before or when v_j stopped increasing.
- Furthermore, since dual variables are increased uniformly in the algorithm, we have $v_j \geq v_k$.
- Now that we have explained how $v_j \geq \frac{c_{ij}}{3}$ (Lemma 7.13 of [7]), we establish the 3-factor bound of Theorem 7.14 of [7] as follows.

The analysis: Relaxed complementary slackness (cont.)

- The cost $\sum_{i \in T'} f_i$ of opening facilities has now been shown to be apportioned to clients j that contribute to the respective finally opened facilities $f(j)$ in T' to which j is connected; note that $i \in T'$ means saturating inequality 14 is satisfied as an equality for facility i.
- So, the total facility opening cost $\sum_{i \in T'} f_i = \sum_{i \in T'} \sum_{j \in A(i)} w_{ij}$, is apportioned to neighbouring clients assigned to facilities $i \in T'$, where $A(i)$ is the set of these clients.
- The connection costs for these clients is $\sum_{i \in T'} \sum_{j \in A(i)} c_{ij}$.
- Summing these two costs for neighbouring clients of facilities in T' gives $\sum_{i \in T'} \sum_{j \in A(i)} (w_{ij} + c_{ij}) = \sum_{i \in T'} \sum_{j \in A(i)} v_j$, because the dual inequalities 15 for all $i \in T'$ attain equality.

The analysis: Relaxed complementary slackness (cont.)

- Clients $j \in D$, not neighbouring facilities in T' have connection costs assigned to respective facilities $f(j) \in T'$ such that $c_{f(j)j} \leq 3v_j$, as established already, resulting in a total cost of at most $\sum_{j \in D \setminus \cup_{i \in T'} A(i)} c_{f(j)j} \leq 3 \sum_{j \in D \setminus \cup_{i \in T'} A(i)} v_j$.
- So, the total cost including costs apportioned to neighbouring clients of T' and connection costs of clients not neighbouring facilities in T' add up to $3 \sum_{j \in D} v_j \leq 3 \times OPT$.
- The 3-factor bound thus follows.
- Note that even if we took three times the cost $\sum_{i \in T'} \sum_{j \in A(i)} (w_{ij} + c_{ij}) = \sum_{i \in T'} \sum_{j \in A(i)} v_j$, of directly connected clients, we still get the same bound of $3 \sum_{j \in D} v_j$ for the total cost (Exercise 7.8 in [7]).

More on relaxed complementary slackness

- We will now view the same algorithm (giving 3-factor approximation) using *relaxed complementary slacknesss conditions* as in [6].
- See inequalities 14 and 15. We continue with the same notations.
- Consider the primal and dual complementary slackness conditions (implications) (i)-(iv) here.

$$(i) \quad x_{ij} > 0 \to v_j - w_{ij} = c_{ij}, \tag{18}$$

$$(ii) \quad y_i > 0 \to \sum_{j \in D} w_{ij} = f_i, \tag{19}$$

$$(iii) \quad \forall j \in D \;\; y_j > 0 \to \sum_{i \in F} x_{ij} = 1 \tag{20}$$

$$(iv) \quad \forall i \in F \;\; \forall j \in D \;\; w_{ij} > 0 \to y_i = x_{ij} \tag{21}$$

More on relaxed complementary slackness (cont.)

- Suppose the optimal LPR solution is integral. Then, each open facility is tight, that is, its cost is fully paid up as per primal slackness condition (ii), Implication 19.
- Now consider the (dual) slackness condition (iv) $w_{ij} > 0 \rightarrow y_i = x_{ij}$ (Implication 21); given that a client $j \in D$ is not connected to open facility $i \in F$, that is $y_i = 1 \neq x_{ij}$, it follows that $w_{ij} = 0$, indicating j does not contribute to any facility apart from the one to which it is connected.
- Also, by primal slackness condition (i), Implication18, for any cient j connected to an open facility i, we have $v_j = c_{ij} + w_{ij}$.
- So, we interpret the total price v_j paid by client j as c_{ij} as going to the connection from j to i, and w_{ij} as the contribution of j to i.

More on relaxed complementary slackness (cont.)

- Now we observe that by relaxing the primal complementary slackness conditions suitably, we may limit the objective function value of the ILP solution to within thrice that of the DLP as follows.
- Assume that $f(j) \in F$ is the facility to which client $j \in D$ is connected. The cost in the ILP is $\sum_{j \in D} c_{f(j)j} + \sum_{i \in T'} f_i$, where $T' \subseteq F$ is the final set of opened facilities.
- So, by altering the primal slackness conditions (i) and (ii) respectively, as
  (I) $\frac{1}{3} c_{f(j)j} \leq v_j - w_{f(j)j} \leq c_{f(j)j}$ for all $j \in D$, and
  (II) $\frac{1}{3} f_i \leq \sum_{j:f(j)=i} w_{ij} \leq f_i$,
  we can ensure factor three approximation because the w_{ij} terms would cancel out on summing the primal objective function as seen in the first inequalities in conditions (I) and (II).

More on relaxed complementary slackness (cont.)

- We will however not use these slackness conditions. We will consider two cases of assigning clients to facilities, and call them *direct* and *indirect* assignments, as presented in [6].
- This is done to improve the approximation factor though not in the worst-case, as we present here in Theorem 5.
- However, this analysis is important as we will use the same technique for proving approximation bounds for another important optimization problem, the k-median problem in Chapter **??**.
- For indirect assignment of a client j to a facility i, we have $w_{ij} = 0$, whence the condition (I) becomes
(I')$\frac{1}{3}c_{f(j)j} \leq v_j \leq c_{f(j)j}$.

More on relaxed complementary slackness (cont.)

- Primary complementary slackness condition (i) is preserved for a directly connected facility j such that $x_{ij} > 0$ implies $v_j - w_{f(j)j} = c_{f(j)j}$, and condition (ii) is maintained, rather than condition (II), so that we have
  (II') $\sum_{j \in D} w_{ij} = f_i$,
  where such clients pay for the facilities costs.
- Our algorithm must achieve the raising of dual variables v_j, paying for costs of opening facilities as well as connecting clients to facilities maintaining conditions (I') and (II').
- So, let us view v_j as $v_j^f + v_j^c$, where v_j^f is the facilities part of the cost and v_j^c is the connection cost.
- For an indirectly connected client j therefore, we wish to enforce $v_j^f = 0$ and $v_j^c = v_j$.

More on relaxed complementary slackness (cont.)

- For a directly connected client j, we know from the complementary slackness condition (i) that $v_j - w_{f(j)j} = c_{f(j)j}$, where $v_j^f = w_{f(j)j}$ and $v_j^c = c_{ij}$, and by the complementary slackness condition (ii) that $\sum_{j \in D} w_{ij} = f_i$.
- In this context we have two observations.

Observation

Observation 2:
$\sum_{(i,j):j \in N(i)} v_j^f = f_i$.

- Clearly, here j *neighbours* i or $j \in N(i)$ and j *contributes* to i.
- Note that $v_j^f = w_{ij}$ for the case where j is directly connected to i and zero, otherwise.

More on relaxed complementary slackness (cont.)

- Furthermore, we can deduce

Observation

Observation 3:
$\sum_{i \in T'} f_i = \sum_{j \in D} v_j^f$.

- Here, T' is the set of finally opened facilities and D is the set of clients.
- Now we claim the following lemma.

Lemma

Lemma 4:
$c_{ij} \leq 3 v_j^c$ *for* $j \in D$ *assigned indirectly to* $i \in T'$.

More on relaxed complementary slackness (cont.)

- We have already discussed the proof of Lemma 4 in Subsection 6 of Section 14. The theorem follows. This theorem is also established in Subsection 6 of Section 14.

Theorem

Theorem 5:
$\sum_{j \in D, i \in F} x_{ij} c_{ij} + 3 \sum_{i \in F} f_i y_i \leq 3 \sum_{j \in D} v_j$, *where the variables are from the primal and dual solutions computed by the algorithm.*

More on relaxed complementary slackness (cont.)

Proof.

For a directly connected client j, $c_{ij} = v_j^c \leq 3v_j^c$, where j is assigned to $i = f(j)$. Lemma 4 further asserts $\sum_{j\in D, i\in F} c_{ij}x_{ij} \leq 3\sum_{j\in D} v_j^c$, even considering indirectly connected clients. Now adding $3\sum_{i\in T'} f_i = 3\sum_{j\in D} v_j^f$ from Observation 3, concludes the proof of this theorem since $\sum_{j\in D, i\in F} c_{ij}x_{ij} \leq 3\sum_{j\in D} v_j^c$ and $3\sum_{i\in T'} f_i = 3\sum_{j\in D} v_j^f$ imply $\sum_{j\in D, i\in F} x_{ij}c_{ij} + 3\sum_{i\in F} f_i y_i \leq 3\sum_{j\in D}(v_j^c + v_j^f) = 3\sum_{j\in D} v_j$. $\square$

- Here, $OPT \geq \sum_{j\in D} v_j$, thereby implying the 3-factor approximation bound, based on Theorem 5.

Amortize analysis for paging

- 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.

Amortize analysis for paging (cont.)

- 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.

Amortize analysis for paging (cont.)

- 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.
- Each $C_A(t)$ may not be at most $c.C_{OPT}(t)$, where A is LRU and OPT is the offline algorithm.

Amortize analysis for paging (cont.)

- However, we show that the amortised cost $C_A(t) + \phi(t) - \phi(t-1)$ is at most $c.C_{OPT}(t)$. How does this follow from the above discussion?
- Thus the total amortised cost $\sum_{t=1}^{m} C_A(t) + \phi(m) - \phi(0) \leq c. \sum_{t=1}^{m} C_{OPT}(t)$.
- As the potential $\phi(0) = 0$ and ϕ is non-negative throughout, A is c-competitive, where $c = k$.

Deterministic paging

- Theorem 1 from [1] 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.

Deterministic paging (cont.)

- To show that the other phases too have at least one page fault for OPT, we use Lemma 6.

Lemma

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 6, 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.

Deterministic paging (cont.)

- So, $P(i)$ must have a page fault for OPT. *(Page requests may lead to page faults. Lemma 6 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 6 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 6 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 $P(i)$.

Deterministic paging (cont.)

- 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.)

Deterministic paging (cont.)

(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 6.

Randomized paging

- If R is a randomized online paging method then its competitiveness is at least H_k against any oblivious adversary. See Theorem 6 in [1].
- We take $k+1$ pages in a set S and choose a probability distribution for choosing the request sequence.
- The first request is chosen uniformly at random from S.
- Each subsequent request is made for a page $\sigma(t)$ that is chosen uniformly at random from $S \setminus \{\sigma(t-1)\}$.
- Now we define a phase that has page requests from $\sigma(i)$ to $\sigma(j)$, for the smallest j, such that the sequence has $k+1$ distinct pages, so that OPT has a fault on this sequence.
- For a deterministic online paging algorithm, the expected cost of A in a phase is $\frac{1}{k}$ for each request, with respect to the distribution as above for the phase sequence.

Randomized paging (cont.)

- We can show using random walks that this sequence has an expected length kH_k.
- The random walk on K_{k+1} has expected number of steps kH_k, for visiting all vertices.
- From this deterministic algorithm A and the distribution of inputs for it, to any randomized online paging algorithm, we now apply Yao's minimax principal for establishing the lower bound for the expected cost of that randomized algorithm. See Theorem 6 in [1].
- Theorem 5 in [1] gives a $2H_k$-competitive randomized online paging algorithm against any oblivious adversary.

Move-To-Front algorithm for list update

- MFT will follow the rules for *access* and movements to the front of the list for each access.
- We can show that access costs can subsume/simulate costs for *insertion* at the end of the list or even *deletions*.
- Also, OPT will suffer access costs, if not any other costs we cannot know.
- Let us look at all elements other than $x = \sigma(i)$, where $\sigma(i)$ is the i-th access in the request string σ.
- All other elements before x in the MFT list are either behind x in OPT's list or after it, and let these be k and l elements, respectively.
- So, the cost of an access by MFT is $C_{MFT}(t) = k + l + 1$.

Move-To-Front algorithm for list update (cont.)

- The cost $C_{OPT}(t)$ of OPT is no lesser than that of skipping k elements in its access query for x plus may be a non-zero cost, and thus at least $k+1$, that is, $C_{OPT}(t) \geq k+1$.
- The number of *inversions* beetween the two lists may be considered as a non-negative potential function ϕ for amortizing costs of MFT.
- Clerly, in MFT's move after accessing x, reduces the number of inversions by l.
- How OPT will move its items is not known but surely MFT's moves cannot raise inversions by more than k.
- So, $C_{MFT}(t) + \phi(t) - \phi(t-1) \leq C_{MFT}(t) + k - l =$ $(k + l + 1) + k - l = 2k + 1 \leq 2(C_{OPT}(t) - 1) + 1 = 2C_{OPT}(t) - 1$
- Adding up over all m steps of a request sequence of length m, we get the 2-competitive upper bound thus.

MAXSAT

- We know that MAXSAT is NP-hard even if we allow at most two literals per clause (see Theorem 9.2 in [5]).
- Suppose we have a set Φ of Boolean expressions ϕ_1, ϕ_2,..., ϕ_m on n variables where each expression has "at most" k variables, where k is a constant.
- We wish to satisfy most expressions. So, we look at all possible 2^n truth assignments.
- If we pick any one random truth assignment the 2^n possibilities then how many expressions in Φ would be satisfied?
- We call this problem k-MAXGSAT (see [5]).
- We can find the expected number $p(\Phi)$ of satisfied expressions as $\sum_{i=1}^{m} p(\phi_i)$, where $p(\phi_i) = \frac{t_i}{2^k}$. Here, t_i is the number of truth assignments satisfying ϕ_i and we already assumed that each ϕ_i uses "exactly" k variables.

MAXSAT (cont.)

- Suppose (on the other hand) we have a deterministic polynomial time algorithm that determines a truth assignment satisfying at least $p(\Phi)$ expressions. This can be shown to hold using a derandomization argument leading to Theorem 13.2 in [5].
- Now we look at the optimal solution whose value OPT is no bigger than the number of expressions with $p(\phi_i) > 0$.
- The approximation ratio is $p(\Phi)/OPT$ must be less than 1 and this must be at least the smallest non-zero $p(\phi_i) \geq \frac{1}{2^k}$. Why?
- Now in MAXSAT, any clause with at least k literals has probability of satisfaction $1 - \frac{1}{2^k}$. So, the approximation ratio by the above result is $1 - \frac{1}{2^k}$, though in the general case at least $\frac{1}{2}$ approximation ratio is trivial. Why?

MAXSAT (cont.)

- Now we show how we compute deterministically, the truth assignment that yields a guaranteed lower bounded number of satisfied expressions, by fixing one variable at a time.
- On setting the next variable to one of the two truth values, we get a new set of expressions. That truth value is assigned to the variable whose redefined set of expressions has higher expectation, no lesser than the expectation of the initial/previous set of expectations.
- Carrying on this process finally leads to a set of at least $p(\Phi)$ expressions satisfied.

Hardness of approximating MAX3SAT (MAXkSAT)

- In MAX3SAT, we have CNF formulas with at most three literals per clause and we wish to maximise the number of satisfied clauses by some truth assignment of the Boolean variables.
- Earlier, we saw that we can design a $(1 - \frac{1}{8})$-approximate, deterministic algorithm for MAX3SAT, where the input is a 3CNF formula with each clause having at most 3 literals.
- Also, in general, it is easy to see that such an algorithm that is 50% approximate is possible.
- For kCNF formulas, the MAXkSAT problem has a deterministic $(1 - \frac{1}{2^k})$-approximate solution.

Hardness of approximating MAX3SAT (MAXkSAT) (cont.)

- We say for every $\rho \leq 1$, an algorithm A is a ρ-approximation for MAX3SAT if for every 3CNF formula ϕ with m clauses, A outputs a truth assignment satisfying at least $\rho OPT(\phi)m$ clauses of ϕ. Here, $OPT(\phi)$ is the maximum number of clauses satisfied by some truth assignment for ϕ.
- Let $L \in NP$ be a language in the class NP. Let x be a string of size n.
- Suppose we design a polynomial time reduction that constructs a 3CNF formula $f(x)$ from x such that (i) if $x \in L$ then all SM clauses in $\phi(x)$ are satisfiable, and (ii) if $x \notin L$ then less than $SM - \frac{S}{2}$ clauses of $\phi(x)$ are satisfiable.
- Here, S is a polynomial in n, the number of $O(\log n)$-sized random strings in the $PCP(\log n, 1)$ characterzation of $L \in NP$, and M is the number of clauses in the 3CNF formula $f(x)$.

Hardness of approximating MAX3SAT (MAXkSAT) (cont.)

- Further suppose that there is a $(1-\frac{1}{2M})$-approximate algorithm A running in polynomial time for the MAX3SAT problem.
- Given $x \in L$, $\phi(x)$ will have all SM clauses satisfied and thus algorithm A will find more than $(1-\frac{1}{2M})SM = SM - \frac{S}{2}$ satisfied clauses, thereby concluding $x \in L$.
- Given $x \notin L$, $\phi(x)$ will have at most $SM - \frac{S}{2}$ clauses satisfied and thus algorithm A will conclude $x \notin L$.
- The gap we see in these two exhaustive cases helps us establish this reduction, leading to a polynomial time decision algorithm for $L \in NP$, thereby showing $P = NP$.
- Here S is a polynomial in n, the number of random strings of $O(\log n)$ bits.

Hardness of approximating MAX3SAT (MAXkSAT) (cont.)

- For each such string, the inspection of k bits in the "proof" y, will lead to deciding "yes" if $x \in L$. So, for whatever k bit positions and whatever bits 0/1 we have in these k places, we can view the decision as being the result of a Boolean function of k variables with at most 2^k clauses in CNF.
- This will blow up to $(k-2)2^k$ clauses in its equivalent 3CNF formula.
- So, we have at most $M = k2^k$ clauses for each of the S random choices.
- See Lemma 10.12 and Theorem 10.7 as covered in class from [3], Section 10.8.1. PCP theorem is stated in Theorem 10.6 [3].

Hardness of approximation

- For a minimization problem an α-approximation algorithm is a polynomial time algorithm with a performance guarantee of α.
- This guarantee is that of the 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 $\alpha = 2$.
- 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.

Hardness of approximation (cont.)

- 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.
- 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.

Hardness of approximation (cont.)

- This partition problem is a well known NP-hard problem.
- Is there an α-approximation algorithm for the 2-bins bin-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.
- Now consider the general bin-packing problem.

Epsilon nets and their applications

- We know that it is easier to hit larger sets by a transversal. For a finite system $(X, \mathcal{F})$, a set $N \subseteq X$ is called an ϵ-net if $N \cap S \neq \phi$ for all $S \in \mathcal{F}$ with $|S| \geq \epsilon|X|$.
- Now suppose we have infinite sets but the probability measure μ is concentrated on finitely many points, that is, a finite set $Y \subseteq X$ has a function w mapping Y to (0,1] with $\sum_{y \in Y} w(y) = 1$, and μ is such that $\mu(A) = \sum_{y \in A \cap Y} w(y)$.
- The measure may or may not be uniform over Y.
- Now if $(X, \mathcal{F})$ is a system of μ-measurable subsets of X, then a subset $N \subseteq X$ is called an ϵ-net for $(X, \mathcal{F})$, with respect to μ, if $N \cap S \neq \phi$ for all $S \in \mathcal{F}$, with $\mu(S) \geq \epsilon$.
- How do we generate a random sample of $s = Cdr \ln r$ draws?
- Here each element is drawn from X as per probability distribution μ.

Epsilon nets and their applications (cont.)

- Theorem 10.2.4 [4] is about X with probability measure μ, but more so $\mathcal{F}$ here is a system of μ-measurable sets. It claims that there is an $\frac{1}{r}$-net for $(X, \mathcal{F})$ with respect to μ of size at most $Cdr \ln r$, where C is a constant.
- To prove Theorem 10.2.4 from Matousek [4], we will require the Lemma 10.2.6 [4] about binomial distributions, a small digression. For the sum X of n independent $0/1$ variables appearing with probabilities p and $1 - p$, we have $\text{Prob}[X \geq \frac{1}{2}np] \geq \frac{1}{2}$, given that $np \geq 8$, that is, $\text{Prob}[X < \frac{1}{2}np] \leq \frac{1}{2}$.
- In the proof of Theorem 10.2.4, let E_0 be the event that a random sample N of $s = Cdr \ln r$ elements from the set X of the set system (range space) $(X, \mathcal{F})$ is not a $\frac{1}{r}$-net.
- Here C is a constant and d is the VC-dimension of the space $(X, \mathcal{F})$.

Epsilon nets and their applications (cont.)

- We show that Prob$[E_0] < 1$ by showing that (i) Prob$[E_0] \leq 2.$Prob$[E_1]$ and (ii) Prob$[E_1] < \frac{1}{2}$, where E_1 is defined as below.
- So, for the event E_0, N not being a $\frac{1}{r}$-net means it misses out a certain set $S \in \mathcal{F}$.
- Let M be another random sample of s elements from X such that M meets S in at least $k = \frac{s}{2r}$ places/points, whereas we already assumed that N misses S entirely.
- Therefore, we define the event E_1 that there is a set $S \in \mathcal{F}$ that misses N but meets M in at least k points.
- Now E_1 requires E_0 and something more. Therefore, Prob$[E_1] \leq$ Prob$[E_0]$. However, we show below that Prob$[E_0]$ is no more than twice Prob$[E_1]$, which in turn is shown to be less than 0.5.

Epsilon nets and their applications (cont.)

- For (ii), we view the genesis of N and M in another equivalent way; we first take a $2s$-long random set A of independent draws from X. Then we randomly choose s positions in A as N and the remaining in M in a total of $\binom{2s}{s}$ ways.
- Then we show for each fixed A that Prob$[E_1|A]$ is small, and thereby Prob$[E_1]$ is small, and consequently Prob$[E_0]$ is small.
- Now for a fixed A, consider the conditional probability $P_S =$Prob$[N \cap S = \phi,\ |M \cap S| \geq k|A]$.
- This probability P_S is non-zero for $|A \cap S| \geq k$, where we bound it by $P_S =$Prob$[N \cap S = \phi|A]$.
- But this is the probability that a random sample of s positions out of $2s$ in A avoids at least k positions in S. An upper bound for this probability is $\frac{\binom{2s-k}{s}}{\binom{2s}{s}}$.

Epsilon nets and their applications (cont.)

- This is at most $(1-\frac{k}{2s})^s \leq e^{-k/2s}s = e^{k/2} = e^{-Cd\ln r/4} = r^{-Cd/4}$.
- This was for a fixed S. The number of such S's that meet a fixed A is the function $\phi_d(2s)$, where d is the VC-dimension of $\mathcal{F}$.
- Since A was also fixed to $2s$ vertices in X, we use the VC-dimension of the $2s$ element subsystem of $(X,\mathcal{F})$, whose hyperedges cardinality is thus bounded by $\sum_{i=0}^{d}\binom{2s}{i} \leq (\frac{e.2s}{d})^d$.
- We observe therefore that Prob$[E_1|A] \leq (\frac{e.2s}{d})^d r^{-Cd/4} = (2Cer\ln r.r^{-C/4})^d < 0.5$, with a large enough choice of C and assuming d and r are more than 2.
- So, for any fixed A we have Prob$[E_1|A] < 0.5$ and therefore Prob$[E_1] < 0.5$.
- Now all we need to show that Prob$[E_0] < 1$ is to show (i), that is Prob$[E_0] \leq 2.$Prob$[E_1]$.
- Here we will use the binomial distribution properties as follows.

Epsilon nets and their applications (cont.)

- We argue with N and M as in the beginning considering the conditional probability Prob$[E_1|N]$.
- Not being a $\frac{1}{r}$-net, N therefore misses out entirely on some $S \in \mathcal{F}$, which we call S_N now.
- We also know now that Prob$[E_1|N] \geq$ Prob$[|M \cap S_N| \geq k(= \frac{s}{2r})]$. Why?
- Now view $|M \cap S_N|$ as the sum of 0-1 random variables with $n = s$ independent draws with probability $p = \frac{1}{r}$, conditional of course to N not meeting S, where each of the s draws for elements in M is with uniform probability $\frac{1}{r}$, giving a random value 0 if it is not in S_N and 1 if it is in S_N.
- So, Prob$[|M \cap S_N| \geq k] \geq 0.5$, as in Theorem 10.2.6 in [4]. Here, $k = \frac{s}{2r} = 0.5np$.
- Thus, Prob$[E_1|N] \geq 0.5$, for any N.

Epsilon nets and their applications (cont.)

- But trivially, Prob[$E_0|N$]$\leq$ 1, and thus Prob[$E_0|n$]$\leq$ 2 Prob[$E_1|N$], for all N.
- So, Prob[E_0]$\leq$ 2Prob[E_1].

References

- Sussane Albers. *Competitive online algorithms*. BRICS, 1996.
- Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2009.
- Dorit S. Hochbaum. *Approximation algorithms for NP-hard problem*. Thomson Asia Pte Ltd., Singapore, First reprint 2003.
- Jiri Matousek. *Lectures on Discrete Geometry*. Springer, 2002.
- Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.
- Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.
- David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.