Approximation and online algorithms: CS60023: Autumn 2018

Instructor: Sudebkumar Prasant Pal
TA: Soumya Sarkar

IIT Kharagpur

email: spp@cse.iitkgp.ernet.in

January 5, 2020

- Total weight of all edges cannot yield better approximation guarante
- Tight example for the greedy weighted set cover algorithm

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 $n = |V|$.

Large DAG subgraphs of directed graphs (cont.)

- 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 [2].

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.

Large cuts for undirected graphs (cont.)

- Therefore, we have total vertex degree across the cut at least half the sum of degrees of all vertices, which is at least $\frac{2e}{2} = e$.
- However, this count is just twice the number of edges across the cut as each edge counts once in the degree of its two vertices. So, we conclude that at least $\frac{e}{2}$ edges are across the cut, which is more than $\frac{OPT}{2}$.

Minimum maximal matchings

- See Problem 1.2 on page 8 in [2]
- 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 [2].
- 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.

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 [2] 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 [2], 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 [2].
- 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 [2] (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 $\alpha y^T Ax$ 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.

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

- This step can be repeated until the graph becomes empty (of edges).
- Alternatively, we may use another rule, where we select an arbitrary edge $\{u, v\} \in E$ and include both u and v in C; we drop all edges incident on u and v and repeat the process until the graph becomes empty.
- For a *straight line graph* (that is, a simple path of n vertices and $n - 1$ edges), the first method finds a vertex cover of size $n - 1$, which is within twice the size of the optimal vertex cover of size $\lfloor n/2 \rfloor$. Why?
- Does it work well also for other classes of graphs like trees, planar graphs, and general graphs?

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

- The second one chooses both vertices of all edges in a *maximal matching* S to be included in the computed vertex cover C.
- *A matching is a set S of edges where no two edges in M share any vertex. A matching S is called a maximal matching if we cannot add an additional edge to M to get a larger matching.*
- We analyze the second heuristic, following the exposition in [1].
- 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?

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

- Vertices in C^* have to cover each edge in the *(maximal) matching* S. So, C^* must include at least one vertex from each of the S edges.
- The $2|S|$ vertices comprise the computed approximate vertex cover C. So, $|C| = 2|S| \leq 2|C^*|$, since $|S| \leq |C^*|$.
- This gives a polynomial time algorithm yielding a vertex cover that is certainly at most twice the size of the minimum vertex cover.
- It is interesting to note that any matching in a graph would force at least as many vertices in the vertex cover as twice the number of edges in the matching.

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

- 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 [2]).
- As mentioned earlier, the *maximal matching* is such that none of its supersets enjoys the same property.
- So, observe that a vertex cover generated by our approximation algorithm might as well be smaller than twice the cardinality of the *maximum matching*.

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

- 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 [2]), 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 [2]).

Maximal matchings lower bound cannot yield better approximation guarantees for vertex covering (cont.)

- 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.

Total weight of all edges cannot yield better approximation guarantees for weighted cut (cont.)

- 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 [2] 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 [2].

Tight example for the greedy weighted set cover algorithm (cont.)

- See Sections 29.7 and 29.9 of [2] for seeing why the obvious greedy algorithm is the best one can hope for.

References

Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2009.

Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.