1 Set Cover, Vertex Cover, and Hypergraph Matching

1.1 Set Cover

Problem (Set Cover) Given a universe U of elements $\{1, \ldots, n\}$, and collection of subsets $\mathcal{S} = \{S_1, \ldots, S_m\}$ of subsets of U, together with cost a cost function $c(S_i) > 0$, find a minimum cost subset of $\mathcal{S}$, that covers all elements in U.

The problem is known to be NP-Complete, therefore we are interested in algorithms that give us a good approximation for the optimum. Today we will look at greedy algorithms.

Greedy Set Cover Algorithm [Chv79] Let OPT be the optimal cost for the set cover problem. We will develop a simple greedy algorithm and show that its output will differ by a certain factor from OPT. The idea is simply to distribute the cost of picking a set S_i over all elements that are newly covered. Then we always pick the set with the lowest price-per-item, until we have covered all elements.

In pseudo-code have have

```
C ← ∅
Result ← ∅
while  C ≠ U
        S ← arg min_{S∈𝒮} c(S)/|S\C|
        ∀e ∈ (S \ C) : price-per-item(e) ← c(S)/|S\C|
        C ← C ∪ S
        Result ← Result ∪ {S}
end
return  Result
```

Theorem 1. *The algorithm gives us a* $\ln n + O(1)$ *approximation for set cover.*

Proof. It is easy to see, that all n elements are covered by the output of the algorithm. Let $e_1, \ldots, e_n$ be the elements in the order they are covered by the algorithm. At iteration k, there must be a set from $\mathcal{S}$ that is contained in the optimal solution and covers some of the remaining elements $U \setminus C$ including e_k at cost at most OPT. Thus we have

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

and using this, we can upper bound the total cost of the greedy solution by

$$\text{Total Cost} = \sum_{k=1}^{n} \text{price-per-item}(e_k) \leq \sum_{k=1}^{n} \frac{OPT}{n-k+1} = \sum_{k=1}^{n} \frac{OPT}{k} = OPT \cdot H_n \approx OPT \cdot (\ln n + 0.6)$$

$\square$

1.2 From Set Cover to Vertex Cover

Set Cover as a Hypergraph We can view an instance of the set cover as a hypergraph where the vertices are the m subsets and the edges are the n elements, that is, a set S_i is contained in a hyperedge j, if S_i as a set contains element j. Then we get the following two quantities:

- *max degree* $\Delta := \max_{v \in V(H)} \deg(v) \equiv$ size of the largest subset S_i.

- *rank* $f := \max_{e \in E(H)} |e| \equiv$ largest occurrence of an element in the subsets.

Special case 1 (small Δ): If Δ is small, we get a better approximation-factor:

Theorem 2. *[Chv79] The greedy algorithm gives a H_Δ-approximation.*

Proof Sketch. Consider one of the sets S_i in OPT. We analyze how much we pay for covering the elements of this set, and argue that we do not pay more than $c(S_i) \cdot H_\Delta$. The approximation factor then follows. Suppose that S_i covers d elements. Note that by definition of Δ, we have $d \leq \Delta$. The first time that the algorithm takes any set that covers at least one element in S_i, the price per item is at most $\frac{c(S_i)}{d}$, because otherwise taking S_i would give a better price-per-item. Similarly, when k items of S_i are already covered and $d - k$ remain, the price-per-item for the algorithm is at most $\frac{c(S_i)}{d-k}$. Hence, similar to the proof of Theorem 1, we can conclude that the overal price paid for covering elements of S_i is at most $\sum_{k=1}^{d} \frac{c(S_i)}{d-k+1} = c(S_i) \cdot H_\Delta$. $\square$

Special case 2: The next special case gets us to the setting of vertex cover. Let us assume that each set has the same cost. Moreover, for simplicity, let us start with the case $f = 2$. First, we repeat some terms from graph theory here

- *Vertex Cover*: A set of vertices of such that each hyperedge contains at least one vertex of this set.

- *Minimal-cardinality vertex cover*: A vertex cover of minimal cardinality

- *Matching*: A set of hyperedges of a graph, such that no two hyperedges share a common vertex.

- *Maximal Matching*: A matching such that no additional hyperedge can be added to it.

- *Maximum Matching*: A matching of maximum cardinality. Note that every maximum matching is also a maximal matching.

Now in the case $f = 2$, our hypergraph is just a graph. If we pick a minimal-cardinality vertex cover, then this is just a minimal-cardinality subset of $\mathcal{S}$ that covers all elements. Unfortunately the vertex cover problem is also known to by NP-complete in general.
There are some well known connections between vertex covers and matching, for example

Theorem (Königs Theorem). *In a bipartite graph the number of vertices in a minimal-cardinality vertex cover equals the number of edges in a maximum matching.*

Observation 3. *Any matching is always smaller than the size of any vertex covering*

This observation gives us an idea for an approximation algorithm in case $f = 2$

Theorem 4. *(An 2-approximation algorithm for Vertex Cover:) For $f = 2$, simply picking a maximal-matching M and outputting all its endpoints gives a 2-approximation of minimum-cardinality vertex cover.*

Proof. The output set is a vertex cover C, as each edge must have one of its endpoints in the chosen vertex-cover (otherwise the matching was not maximal). Since $|M| \leq OPT$, we have $|C| = 2 \cdot |M| \leq 2 \cdot OPT$. $\square$

Extension to f-approximation algorithm for Hypergraph Vertex Cover: Similarly, for any larger f, we can get an f approximation by picking a maximal matching—i.e., a matching to which we cannot add any more hyperedges—and outputting all of its endpoints.

1.3 Hypergraph Matching

Above we used maximal matchings in hypergraphs, which are easy to find. How about the maximum matching?

Maximum Matching Approximation Algorithm: If we want to find the size OPT of a maximum matching on a hypergraph of rank f, there is a simple greedy f-approximation algorithm: Build a maximal matching by greedily taking hyperedges from our graph as long as it is possible. The resulting matching is clearly a maximal matching, as no further hyperedges can be added to it.

Lemma 5. *Any maximal matching is an f-approximation of maximum matching.*

Proof. Imagine that we give 1 dollar to every hyperedge e of OPT, and ask these hyperedges to move their money to their neighboring hyperedges in the maximal matching. That is, each hyperedge e of OPT gives its 1 dollar to one of the incident maximal matching hyperedge (there must be one). A hyperedge of the output cannot overlap with more than f hyperedges of OPT (why?). Thus every hyperedge of the output receives at most f dollars. Since we have moved all the OPT dollars to the output matching, while each received at most f dollars, this implies that the cardinality of the output matching must be at least OPT/f. $\qquad\square$

2 Minimum Makespan Scheduling

Problem (Minimum Makespan Scheduling): We have a set of m identical machines and a set of n tasks, where each task takes $p_1, \ldots, p_n$ time to run on a machine. Each machine can only run one task at a time and a task can only be assigned to one machine. We want to assign the tasks in such a way as to minimize the total time, called the *makespan*, until all tasks are finished.

Graham's Algorithm (also called List Scheduling Algorithm): [Gra66] Put each task on the least loaded machine at that point.

Theorem 6. *Graham's gives a 2-approximation of the optimal makespan time OPT.*

Proof. We use the following two obvious lower bounds on OPT:

- LB1: avg load $= \frac{\sum p_i}{m} \le OPT$

- LB2: $p_{max} \le OPT$

where p_{max} is the task with the longest time. Let C_{max} be the makespan return by Graham's algorithm and let j be the task, that finishes last. Then $C_{max} - p_j \le \frac{1}{m} \sum_i p_i \le OPT$ and $p_j \le OPT$ immediately gives us $C_{max} \le 2 \cdot OPT$. $\qquad\square$

This version of Graham's algorithm works in an online fashion, i.e. we can expect the next task to arrive, while the previous tasks are already appointed to their machines. If we know the the times of all the tasks already in the beginning, we get an even better result.

Theorem 7. *Graham's algorithm applied to a sorted list with non-increasing processing times gives a $\frac{4}{3} \cdot$ OPT-approximation.*

Proof. Let $p_1 \ge p2 \ge \cdots \ge p_n$ and let j be the last task to finish, in Graham's algorithm. Then we can w.l.o.g assume that $p_j = p_n$, as otherwise we could remove the last task, which would decrease OPT but not the outcome of the algorithm, so we would get an even better bound. If $p_n \le \frac{\text{OPT}}{3}$, then we are done.
Otherwise suppose $p_n > \frac{\text{OPT}}{3}$. We will show, that in this case, Graham's algorithm actually achieves the optimum. For the sake of contradiction, suppose that p_n finishes in Graham's algorithm after OPT time. Consider the time that Graham's algorithm was adding p_n. Let's order the machines in such a way that the first i machines get a single task and the other $m - i$ get two tasks (right before adding p_n). Notice that no machine can have zero tasks (as then p_n would be added on that machine and it would finish in $p_n < $ OPT time) or more than two tasks (as that would sum up to more than $3p_n > $ OPT. We call the first kind of tasks *heavy tasks*. Notice that we were unable to add p_n to the machines with single heavy tasks and still finish in OPT time. Since p_n is smaller than all tasks before it, this means that even in the optimum schedule, these heavy tasks must be alone. There are $2(m - i) + 1$ other (non-heavy) tasks — including p_n — and somehow in the optimum schedule, these should be processed in the remaining $m - i$ machines. But that is a contradiction as this means some machine should process at least three tasks, each of them of size $p_n > \frac{\text{OPT}}{3}$, which means the optimum schedule finishes in time strictly more than OPT. Having arrived at contradiction starting from the assumption that Graham's algorithm finishes after OPT time, we conclude that the assumption is incorrect, i.e., Graham's algorithm finishes in OPT time (in this case of $p_n > \frac{\text{OPT}}{3}$). $\qquad\square$

References

[Chv79] Vasek Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.

[Gra66] Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966.