GEORGIA TECH S'22 CS 6550/8803: ADVANCED ALGORITHMS & UNCERTAINTY

Lecture 1: Introduction and Discrete Optimization

Lecturer: *Sahil Singla* Last updated: January 3, 2022

In this course we will focus on designing algorithms, especially for problems where the algorithm's input is "uncertain", i.e., the input is not entirely known. Since algorithm design is often for an optimization problem, in the first few lectures we will study classical discrete and continuous optimization problems, and then in the later lectures we will explore these problems for uncertain inputs. This lecture is to recall the commonly studied discrete optimization problems.

1 What is Discrete/Combinatorial Optimization?

Given a *finite* set of ground elements V, a family of constraints $\mathcal{F} \subseteq 2^V$, and an objective function $f : 2^V \to \mathbb{R}$, the *combinatorial/discrete optimization* problem is to select a *feasible* set $S \subseteq V$, i.e. $S \in \mathcal{F}$, while trying to maximize (or minimize) the objective $f(S)$:

$$\max / \min \quad f(S) \qquad \text{subject to} \qquad S \in \mathcal{F}.$$

Since the number of subsets of V is finite, it is possible to run an algorithm that evaluates f at every feasible subset to find the optimal solution. However, such an approach is not computationally *efficient* as the number of subsets is exponential in $n := |V|$.

In general getting a poly(n) running time algorithms for the above optimization problem is impossible because both $\mathcal{F}$ and f might have an exponential description. One natural way around this is to have an "implicit" description of $\mathcal{F}$ and f which has a polynomial size representation. E.g., in an additive/linear function we are given some weights $\mathbf{w} \in \mathbb{R}^{|\mathbf{V}|}$ and the objective $f(S) := \sum_{e \in S} w_e$ for every $S \subseteq V$. Some examples of implicit constraints are cardinality where for a given integer k we have $\mathcal{F} := \{S : |S| \leq k\}$ and matching constraints where for a given graph G a subset of edges is feasible if they form a matching.

In this lecture we also consider several combinatorial functions and constraints that do not always have a polynomial size description (even implicit) but given access to an "oracle" they can be efficiently optimized. The most common oracles are "independence" and "value" oracles: For any set $S \subseteq V$, an *independence oracle* for $\mathcal{F}$ returns if S is feasible (i.e., $S \in \mathcal{F}$) and a *value oracle* for f tells returns the value $f(S)$. Now our goal is to design *efficient* algorithms that only make poly$(|V|)$ calls to these oracles.

It turns out that for an arbitrary $\mathcal{F}$ or f one can still not design efficient algorithms given oracle access, so we will need to make further assumptions on their structure. But before discussing these common combinatorial constrains and functions, we take a brief detour and introduce the other protagonist of this course, *Uncertainty*.

2 What is Optimization under Uncertainty?

Consider the simplest "single-item" discrete optimization problem: given weights $w_e \in \mathbb{R}_{\geq 0}$ for each element e, the goal is to select the element with the highest weight. This is a trivial problem when the weights are known (pick the highest), however, the problem already becomes non-trivial when the weights are "uncertain" as we discuss below.

There are several ways of modeling input uncertainty, and often the application dictates the best model. The uncertainty models studied in this course can be broadly classified into three categories: *online*, *stochastic*, and *strategic*. We next briefly describe each of these categories, giving a concrete scenario where we might encounter the single-item problem.

1. **Online.** The input to the algorithm is revealed one-by-one and it has to make decisions without knowing the future. E.g., consider a scenario where you own a diamond (item) that you are trying to sell. A sequence of n buyers arrive, each with a different value for your item. On arrival, buyer e makes a take-it-or-leave-it offer for your item, (i.e., reveal w_e). Your goal is to immediately decide whether to sell your item, while trying to maximize the value that you get for your item. The benefit of declining the early bids is that you might get a better bid later, but then you also risk selling your item at a lower price later.

2. **Stochastic.** The input is unknown but it is drawn from a known probability distribution, and the goal is to maximize the expected reward. E.g., suppose you want to purchase a house. You have some estimates on the value of every available house in the market, say based on its location and size (i.e., probability distribution on w_e). However, to find the exact value of a site (house) you have to hire a house inspector and pay her a price. Your goal is to simultaneously maximize the value of the best site that you find and to minimize the total inspection price that you pay. One simple strategy is to inspect every potential site to find the best one, but this strategy incurs a large inspection price. Another strategy is to only inspect the most promising site, but this site might have a value much smaller than the maximum value site. What is the optimal inspection strategy?

3. **Strategic.** The input is distributed between different agents who may misreport their input to maximize their personal gain. E.g., suppose the government wants to auction a resource (like spectrum or mining-license) and give it to a company/agent e that has the best technology since that would add the most value w_e to the society. One option for the government is to ask each agent how much they value the resource (i.e., their bid) and then allocate it to the highest bidder, charging the winner their bid. However, this algorithm might lead to unexpected outcomes since the agents would misreport their bids to get the resource at the cheapest possible price.

We will be discussing each of the above uncertainties in detail during the semester (roughly 3 weeks on each topic). But for the next couple of weeks, we will study some optimization algorithms for settings where the input is entirely given.

3 Combinatorial Constraints

The most common families of constraints are packing and covering constraints. Some examples packing constraints are cardinality, acyclic subgraphs, matroids, matchings, knapsack, and orienteering. Some examples of covering constraints are spanning/connected graphs, vertex/set cover, Steiner tree, and facility location.

Definition 1 (Downward-closed or packing constraints)**.** *An independence family $\mathcal{F} \subseteq 2^V$ is called* downward-closed *if $A \in \mathcal{F}$ and $B \subseteq A$ implies $B \in \mathcal{F}$. Some examples are:*

- Matroid $(V, \mathcal{F})$*: The elements of $\mathcal{F}$ satisfy the matroid exchange axiom: for all pairs of sets $I, J \in \mathcal{F}$ such that $|I| < |J|$, there exists an element $x \in J$ such that $I \cup \{x\} \in \mathcal{F}$. Elements of $\mathcal{F}$ are called* independent *sets. E.g., uniform matroid $\mathcal{F} = \{S : |S| \leq k\}$ and partition matroid $\mathcal{F} = \{S : |S \cap A_i| \leq a_i\}$ for a given partition $\{A_1, A_2, \ldots\}$ of V.*
- Intersection of matroids*: If there exist two matroids $\mathcal{M}_1 = (V, \mathcal{F}_1)$ and $\mathcal{M}_2 = (V, \mathcal{F}_2)$ such $\mathcal{F} = \mathcal{F}_1 \cap \mathcal{F}_2$. E.g., bipartite matching is intersection of two partition matroids.*
- Knapsack*: Given $w \in \mathbb{R}^n_{\geq 0}$ and a knapsack size $B > 0$, set $S \in \mathcal{F}$ iff $\sum_{i \in S} w_i \leq B$.*
- Orienteering*: Given a metric (V, E, d), a budget $B > 0$, and a root $r \in V$, set $S \subseteq V$ is in $\mathcal{F}$ iff there is a walk starting at r of length at most B that visits all nodes in S.*
- k-extendible system*: If for every $A \subseteq B \in \mathcal{F}$ and $e \in T$ where $A \cup \{e\} \in \mathcal{F}$, we have that there is a set $Z \subseteq B \setminus A$ such that $|Z| \leq k$ and $B \setminus Z \cup \{e\} \in \mathcal{F}$. These are more general than intersection of matroids; e.g., a 2-extendible system captures matching in general graphs.*

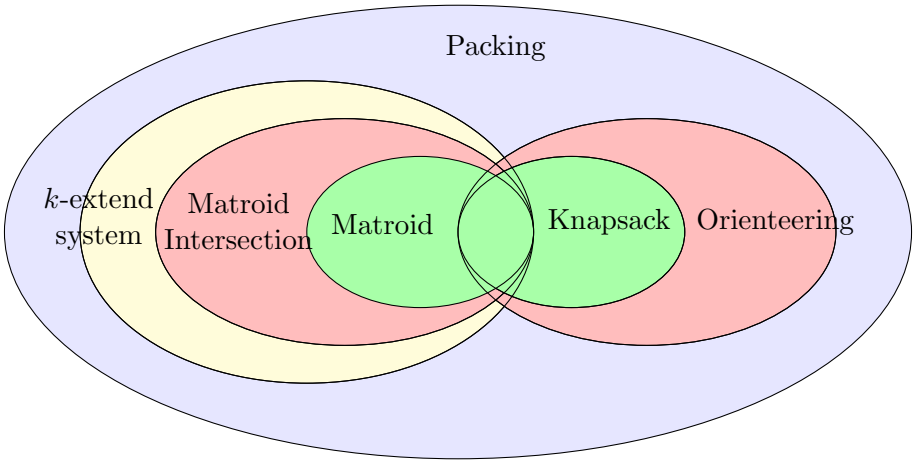

Figure 1: Hierarchy of packing constraints.

We now define and give some examples of covering constraints.

Definition 2 (Upward-closed or covering constraints)**.** *An independence family $\mathcal{F} \subseteq 2^V$ is called* downward-closed *if $A \in \mathcal{F}$ and $B \supseteq A$ implies $B \in \mathcal{F}$. Some examples are:*

- Matroid Basis*: Given a matroid, set S is feasible iff it contains a matroid basis. E.g., size constraint $\mathcal{F} = \{S : |S| \geq k\}$.*
- Set Cover*: Suppose for every $i \in V$ we are guven a set $T_i \subseteq [n]$. Now a set $S \subseteq V$ is feasible iff $\bigcup_{i \in S} T_i \supseteq [n]$.*
- Uncapacitated Facility Location*: Given a graph $G = (V, E)$ with metric (V, d), $Clients \subseteq V$, and facility opening costs $\mathbf{X} : V \to \mathbb{R}_{\geq}$, open facilities at some locations $\mathbb{I} \subseteq V$ to minimize the sum of facility opening costs and the connection costs to Clients, i.e.,*

$$\sum_{i \in \mathbb{I}} X_i + \sum_{j \in Clients} \min_{i \in \mathbb{I}} d(j, i).$$

- Steiner Tree*: Given a graph $G = (V, E)$ with some edge costs $\mathbf{c} : E \to \mathbb{R}$ and a subset of nodes $S \subseteq V$. The goal is to find a tree $T \subseteq E$ that connects all the nodes in S (it may or may not connect nodes in $V \setminus S$) of minimum cost $\sum_{e \in T} c_e$.*

We stress that there are important families of constraints that are not packing/covering constraints, e.g. mixed packing-covering constraints, but we won't discuss them here.

4 Combinatorial Functions

We denote the ground set by V, with $n = |V|$. A function $f : 2^V \to \mathbb{R}$ is *monotone* if $f(S) \leq f(T)$ for all $S \subseteq T \subseteq X$. The most common class of objective functions are additive.

Definition 3 (Additive/Linear function)**.** *Given a vector $\mathbf{c} \in \mathbb{R}^{|\mathbf{V}|}$, we define $f(S) := \mathbf{c}^\intercal \cdot \mathbf{1_S}$ for any $S \subseteq V$, where $\mathbf{1}_S$ is a vector of dimension $|V|$ that contains 1 for $i \in S$ and 0 otherwise.*

Next we consider more general functions that often do not have a polynomial size description, but given value oracle access can be optimized for several constraint families.

Definition 4 (Combinatorial functions)**.** *A function $f : 2^V \to \mathbb{R}$ is*

- Additive/Linear function *Given a vector $\mathbf{c} \in \mathbb{R}^{|\mathbf{V}|}$, we define $f(S) := \mathbf{c}^\intercal \cdot \mathbf{1_S}$ for any $S \subseteq V$, where $\mathbf{1}_S$ is a vector of dimension $|V|$ that contains 1 for $i \in S$ and 0 otherwise.*
- Submodular *if for every $S, T \subseteq V$ it satisfies*

$$f(S \cap T) + f(S \cup T) \leq f(S) + f(T).$$

  E.g., max value function, number of distinct items, matroid rank function, coverage function, cut function, entropy, and log determinant.

- Monotone XOS (Fractionally subadditive) *if it can be written as the maximum of additive functions: i.e., there are vectors* $\mathbf{c_i} \in \mathbb{R}^{\mathbf{n}}_{\geq \mathbf{0}}$ *for* $i \in \{1 \dots W\}$ *such that*

$$f(S) := \max_{i=1}^{W} \left(\mathbf{c}_i^{\intercal} \cdot \mathbf{1}_S \right) = \max_{i=1}^{W} \Big(\sum_{j \in S} \mathbf{c}_i(j) \Big).$$

  An alternative characterization due to Feige [1], which motivates the name fractionally subadditive*: a function is XOS if* $f(T) \leq \sum_i \alpha_i f(S_i)$ *for all* $\alpha_i \geq 0$ *and* $\chi_T = \sum_i \alpha_i \chi_{S_i}$*. The* width *of an XOS function is the smallest number* W *such that* f *can be written as the maximum over* W *linear functions.*

  E.g., maximum-weight matching, rank of intersection of k *matroids, knapsack, longest increasing subsequence, and largest independent set in a graph.*

- Subadditive *if for every* $S, T \subseteq V$ *it satisfies*

$$f(S \cup T) \leq f(S) + f(T).$$

  E.g., Set cover, Steiner tree, facility location, multiway cut, TSP, and makespan.

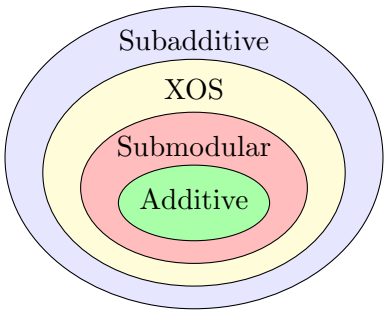

Figure 2: Hierarchy of monotone combinatorial functions.

Although the above combinatorial functions capture many natural problems, we stress that there are many interesting classes of functions beyond these classes. We list a few such functions below, but they will not be discussed in much detail in this lecture.

- k-median or k-center for a set of vertices in a metric.
- Maximum flow from s to t: a subadditive function over paths but not edges.
- Largest connected component.
- Some scheduling objectives like sum of job completion times.

5 Greedy Algorithms

Algorithm design for discrete optimization problems is a vast field which is still an active field of research. We refer the interested readers to books such as [3, 4]. In this section we

study one of the most basic optimization algorithms, the *Greedy Algorithm*, where we want to maximize an objective function f subject to a packing constraint $\mathcal{F}$. In this iterative algorithm we select the element with the highest "marginal" value in each iteration.

Algorithm 1 Greedy Algorithm

1: Start with $A_0 = \emptyset$.
2: **for** steps $i = 1, 2, \ldots, |V|$ **do**
3: Let $e_i = \arg\max_{e \in V \setminus A_{i-1} \ \&\ A_{i-1} \cup e \in \mathcal{F}} \{f(A \cup e) - f(A)\}$, where ties are broken arbitrarily. If no such element exists then return Return A_{i-1}.
4: $A_i = A_{i-1} \cup e_i$.

One of the major reasons why matroid constraints are interesting is because for linear objective functions they precisely capture the constraint family for which the greedy algorithm returns the optimum solution. We will prove the following theorem in Homework 1.

Theorem 1. *For the linear function $f(S) := \sum_{e \in S} w_e$, where w_e is the non-negative weight of element $e \in V$, the greedy algorithm finds the max-weight feasible set* if and only if *$\mathcal{F}$ is a matroid (i.e., the algorithm succeeds whenever $\mathcal{F}$ is a matroid. Also, if $\mathcal{F}$ is not a matroid, then there are instances of weights for which the algorithm fails).*

Beyond linear functions, the greedy algorithm also has good performance guarantees for submodular objectives.

Theorem 2 ([2])**.** *For any monotone*[1] *submodular function $f : 2^V \to \mathbb{R}_{\geq 0}$ and cardinality constraint k (i.e., set size at most k), the greedy algorithm returns a solution with value at least $(1 - 1/e)$ times the optimum.*

Proof. Let $\mathrm{OPT} \subseteq V$ denote the set of k elements selected by an optimal solution. We start by observing that for cardinality constraint the greedy algorithm selects new elements for exactly the first k iterations/steps, and then returns A_k. The heart of the proof is the following claim which gives a lower on the increase in algorithm's value in each step.

Claim 3. *For $i \leq k$, in the i-th iteration we have*

$$f(A_i) - f(A_{i-1}) \geq \frac{1}{k} \left(f(OPT) - f(A_{i-1}) \right).$$

Proof of Claim 3. Observe that in this iteration i the algorithm could have selected any of the elements in the set $\mathrm{OPT} \setminus A_{i-1}$. Since on adding this entire set to A_{i-1} the algorithm's value increases by $f(\mathrm{OPT} \cup A_{i-1}) - f(A_{i-1}) \geq f(\mathrm{OPT}) - f(A_{i-1})$, we get that the best element in this set gives a marginal value that is at least the average gain (formally, this uses that submodular functions are subadditive):

$$\frac{1}{|\mathrm{OPT} \setminus A_{i-1}|} \Big(f(\mathrm{OPT}) - f(A_{i-1}) \Big) \geq \frac{1}{k} \Big(f(\mathrm{OPT}) - f(A_{i-1}) \Big).$$

Since e_i gives a marginal value that is at least the marginal value of the best element in $\mathrm{OPT} \setminus A_{i-1}$, this proves the claim. □

[1] That is, $f(A) \leq f(B)$ for any $A \subseteq B$.

Given Claim 3, the proof of the theorem follows from a simple calculation. The claim gives:

$$\begin{aligned} f(A_i) &\geq \frac{1}{k} f(\mathrm{OPT}) + \left(1 - \frac{1}{k}\right) \cdot f(A_{i-1}) \\ &\geq \frac{1}{k} f(\mathrm{OPT}) + \left(1 - \frac{1}{k}\right) \cdot \left(\frac{1}{k} f(\mathrm{OPT}) + \left(1 - \frac{1}{k}\right) \cdot f(A_{i-2})\right), \end{aligned}$$

and so on. Simplifying and using $f(A_0) \geq 0$, we get

$$\begin{aligned} f(A_k) &\geq \frac{1}{k} f(\mathrm{OPT}) \cdot \left(1 + \left(1 - \frac{1}{k}\right) + \left(1 - \frac{1}{k}\right)^2 + \ldots + \left(1 - \frac{1}{k}\right)^{k-1}\right) \\ &= f(\mathrm{OPT}) \cdot \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \quad \geq \quad f(\mathrm{OPT}) \cdot \left(1 - \frac{1}{e}\right), \end{aligned}$$

where the last inequality uses $(1 - 1/k) \leq e^{-1/k}$. □

Interestingly, this factor $1-1/e$ is tight for submodular maximization subject to a cardinality constraint: it is known that given value oracle access to f, no (randomized) algorithm that makes polynomial number of queries to this oracle can obtain an approximation ratio $1 - 1/e + \Omega(1)$.

There are many other interesting results that are known about greedy algorithms; you might want to prove the following:

Exercise. For non-negative linear function maximization subject to intersection of k matroids (or even k-extendible system), the greedy algorithm gives a k-approximation.

Exercise. For non-negative monotone submodular maximization subject to a matroid constraint, the greedy algorithm gives a 2-approximation.

For the latter problem of monotone submodular maximization subject to a matroid constraint, a remarkable result shows that it's even possible to obtain an $e/(e-1)$-approximation in polynomial time (which is unimprovable even for cardinality constraint). This algorithm is roughly based on a "continuous" version of the greedy algorithm; see [5] for details.

References

[1] Feige, Uriel. On maximizing welfare when utility functions are subadditive. SIAM Journal on Computing 39.1 (2009): 122-142.

[2] Nemhauser, G. L., L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—I. Mathematical programming (1978).

[3] Schrijver, Alexander. Combinatorial optimization: polyhedra and efficiency. Vol. 24. Springer Science & Business Media, 2003.

[4] Williamson, David P., and David B. Shmoys. The design of approximation algorithms. Cambridge university press, 2011.

[5] Calinescu, G., Chekuri, C., Pal, M., and Vondrák, J. (2011). Maximizing a monotone submodular function subject to a matroid constraint. SIAM Journal on Computing.