Today: how to generalize some well-known approximation results

1 Intuition: Behavior of Functions

- Consider a real-valued function \( g(z) \) on integers (or reals).
- **Defn:** \( \Delta g(1 \mid z) = g(z + 1) - g(z) \) is the discrete derivative of \( g \) at \( z \). It measures how a small change in \( z \) impacts \( g(z) \).
- For some functions, \( \Delta g(1 \mid z) \) is identical for all \( z \).
- **Ex:** for linear functions, \( g(z + 1) = g(z) + g(1) \).
- For some functions, \( \Delta g(1 \mid z) \) decreases as \( z \) increases.
- These are (roughly) the concave functions, such as \( \log z \), \( \sqrt{z} \), \( 1 - z^2 \), or \( 1 - e^{-z} \).
- If \( \Delta g(1 \mid z) \geq 0 \) for all \( z \), we say that \( g \) is monotone (specifically, monotonically increasing).

Who cares?

- We can prove useful things about functions given only the knowledge of the above properties.
- For example, linearity is an extremely helpful property (consider, e.g., linearity of expectation in probability).
- Concavity is also useful for optimization (e.g., efficiently finding the maximum of \( g \)).
- It also helps derive efficient algorithms; for example, biological sequence alignment with concave gap penalties takes time \( O(n^2 \log n) \), vs \( O(n^4) \) if nothing is known about the gap penalty function.
- Finally, note that a monotone concave function models the intuitive idea of “diminishing returns”: as we add more to \( z \), the improvement in \( g(z) \) becomes smaller and smaller.

In what follows, we’re going to extend these intuitive ideas from numeric functions to functions on sets, and show that they are helpful in proving approximation bounds!
2 Modular and Submodular Functions

Consider functions $f$ over sets.

- Let $U$ be a universe of elements.
- $f$ is a real-valued function on $2^U$, the set of all subsets of $U$.
- **Defn:** Let $S \subseteq U$ and $x \in U$ be given. The *discrete derivative* of $f$ w/r to $x$ at $S$ is given by
  \[
  \Delta_f(x \mid S) = f(S \cup \{x\}) - f(S).
  \]
- The discrete derivative quantifies now much adding $x$ to $S$ changes $f$.

We’ll now define set function properties similar to those described above for functions on numbers.

- **Defn:** A set function $f$ is *modular* if, for all $A, B \subseteq U$ and $x \notin A, B$,
  \[
  \Delta_f(x \mid A) = \Delta_f(x \mid B).
  \]
- In other words, the contribution of adding $x$ to a set is independent of the starting set.
- Modularity is comparable to linearity. In particular, if $f$ is a modular set function, and $f(\emptyset) = 0$, then for any $A \subseteq U$,
  \[
  f(A) = \sum_{x \in A} f(\{x\}).
  \]
  (Can easily prove by induction on $|A|$.)
- **Defn:** A set function $f$ is *submodular* if, for all $A \subseteq B \subseteq U$ and $x \notin B$,
  \[
  \Delta_f(x \mid B) \leq \Delta_f(x \mid A).
  \]
- In other words, the marginal improvement in $f$ obtained by adding $x$ to a set $S$ decreases (or stays the same) as we add other stuff to $S$.
- Submodularity is comparable to concavity.
  (Note that every modular function is also submodular by definition.)
- **Defn:** a set function $f$ is *monotone* if, for all $A \subseteq B \subseteq U$,
  \[
  f(A) \leq f(B).
  \]
- In other words, $f(S)$ never decreases as we augment a set $S$ with more elements.
- Once again, a monotone submodular function models the idea of diminishing returns: as we add elements to a set, the marginal impact of each new element on $f$ decreases or stays the same.
3 Examples of Modular and Submodular Functions

Some common elementary functions on sets, as well as measures of interest for real optimization problems, fit the definitions above.

- The size of a set $S$, $|S|$, is a modular function.
- Given any weighting function $w$, on elements the total weight of a set, $\sum_{x \in S} w(x)$, is also modular.
- Any concave function of either of these two set functions is submodular, e.g. $\sqrt{|S|}$.
- Now consider a universe $U$ of (possibly overlapping) subsets of some ground set $G$.
- The coverage $C(S)$ of a collection $S = \{s_1 \ldots s_n\}$ of sets is the total number of ground elements contained in $S$. That is,

$$C(S) = \left| \bigcup_{i=1}^{n} s_i \right|.$$

- Observe that coverage is a (monotone) submodular set function!
- (Indeed, suppose we have two collections $S \subseteq S'$, and we add a new set $t$ to each. Every element of $t - S'$ is also an element of $t - S$, but the converse need not be true.)
- The above also holds if the ground set is weighted, and coverage is measured by the total weight of covered elements.

These are nice, simple examples of submodularity, but it also holds for much messier “coverage-like” functions!

- Suppose we are designing a building, and we have $n$ locations $L_1 \ldots L_n$ wired to install smoke detectors.

- If we put detectors at any subset $S$ of these $n$ locations, we have some probability $p(S)$ of detecting a fire in the building.

- Now $p(S)$ may be a complex function of $S$, since the detectors’ fields of effectiveness may differ depending on their location (near walls, in the path of an air vent, etc.), and these fields may overlap in complex ways.
Indeed, we may not even be able to compute $p(S)$ except by simulation.

However, it is reasonable to assume that $p$ is a monotone submodular function!

Adding one more smoke detector to an existing set $S$ never makes $p$ worse.

However, given $S \subseteq S'$, adding a detector $L_i$ to $S'$ is never more effective than adding it to $S$, since $S'$ contains additional detectors whose fields may overlap that of $L_i$.

Coverage-like functions can come in lots of different forms!

Consider the following 	extit{facility location} scenario.

Starbucks has the option to build stores at any of a list of locations $L_1 \ldots L_n$, to serve a group of $m$ customers.

Each customer chooses one store to patronize. If customer $j$ patronizes store $L_i$, he consumes $c_{ij}$ milligrams of caffeine per day.

Consider the total caffeine intake $C(S)$ induced by building a subset $S$ of the $n$ possible stores, assuming that each customer chooses the store that delivers the most caffeine.

$$C(S) = \sum_{j=1}^{m} \max_{L_i \in S} c_{ij}$$

As above, it’s easy to argue that $C(S)$ is a monotone submodular function.

One more non-coverage example.

Consider cuts of a graph $G = (V, E)$ into vertex sets $S$ and $V - S$.

Let $C(S)$, the size of a cut $(S, V - S)$, be the number of edges that cross the cut.

$C$ is not a monotone function – it is 0 for $S = \emptyset$ and $S = V$ and non-zero for sets in between these two.

But $C$ is submodular! Indeed, consider cuts $S \subseteq S'$. If we add a vertex $x$ to $S'$, some subset of its adjacent edges may be to vertices in $S' - S$. These edges contribute to $C(S \cup \{x\})$, but not to $C(S' \cup \{x\})$.

These are just a few examples of interesting submodular functions that arise in real optimization problems. There are lots more examples from, e.g., the field of machine learning.

4 How Hard are Optimization Problems on Submodular Functions?

We can formulate many problems of interest as optimization of a submodular function, possibly subject to constraints.

Many examples above are tied to 	extit{constrained maximization}, e.g. “Find the highest-coverage collection of at most $k$ sets/detectors”, or “Find the best set of facilities of total cost at most $C$”.

4
• Graph cuts have two natural problems: MIN-CUT and MAX-CUT, which are unconstrained min and max problems.

• How hard are these tasks?

Here are some general results:

• **Unconstrained minimization** of submodular functions is “easy” – like LP, it can be done in time polynomial in the problem size. As one approach, we can reduce the problem to a particular convex program that can be solved in polynomial time.

• **Constrained minimization**, for many natural types of constraint, is both an NP-optimization problem and very hard to approximate – often cannot do better than a ratio that grows polynomially with problem size unless P=NP.

• **Maximization** of submodular functions, either unconstrained or subject to various natural types of constraints, is an NP-optimization problem.

• (For unconstrained, reduce from MAX-CUT; for constrained, the coverage and facility-location examples shown above are all known to be hard.)

• But it turns out that submodular maximization is relatively easy to approximate well!

• These approximation results can then generalize to a lot of problems that naturally reduce to submodular maximization.

• We’ll show some examples...

### 5 Maximization with Cardinality Constraint

• Let $f$ be a non-negative, monotone submodular function.

• Let $U$ be the universe of elements, and suppose we want to find a subset $S \subseteq U$ that maximizes $f$, subject to $|S| \leq k$.

• (WLOG, $|S| = k$, since $f$ is monotone.)

• An example of this problem is finding a collection of at most $k$ sets that cover the maximum number of elements from a ground set. This problem is MAX-COVER, the maximization version of SET-COVER.

• (Since SET-COVER is hard, so is MAX-COVER, and hence so is submodular maximization with cardinality constraints.)

Here’s a very simple heuristic that was analyzed by Nemhauser et al. (1978).

• Start with the empty set.

• Repeatedly add the item from $U$ that gives the greatest marginal increase in $f$.

• In pseudocode,
$S_0 \leftarrow \emptyset$

for $i = 1..k$
   $S_i \leftarrow S_{i-1} \cup \{\text{argmax}_x \Delta_f(x \mid S_{i-1})\}$

- Note that this algorithm generalizes the obvious greedy heuristic for maximizing coverage.

- **Thm:** if we run the greedy algorithm for $\ell$ steps, then
  $$f(S_\ell) \geq \left(1 - e^{-\ell/k}\right) f(S^*)$$

  where $S^*$ is the subset of $U$ of size $\geq k$ that maximizes $f$.

On with the proof!

- Again WLOG, $|S^*| = k$, since $f$ is monotone.
- Suppose $S^* = \{x^*_1 \ldots x^*_k\}$.
- For all $i \leq L$, we have
  $$f(S^*) \leq f(S^* \cup S_i)$$

  $$= f(S_i) + \sum_{j=1}^k \Delta_f(x^*_j \mid S_i \cup \{x^*_1 \ldots x^*_{j-1}\})$$

  (First line by monotonicity, second because $f(S)$ is simply the sum of $\Delta$’s for each element of $S$.)

- By the diminishing returns property of submodular functions, we have
  $$f(S^*) \leq f(S_i) + \sum_{x \in S^*} \Delta_f(x \mid S_i).$$

- Now the greedy algorithm adds the element $\hat{x}$ to $S_i$ s.t. the improvement $\Delta(\hat{x} \mid S_i)$ is maximized. Hence,
  $$f(S^*) \leq f(S_i) + \sum_{x \in S^*} [f(S_{i+1}) - f(S_i)]$$

  and so
  $$f(S^*) - f(S_i) \leq k[f(S_{i+1}) - f(S_i)]$$

- Intuitively, we have shown that adding the greedy choice to $S_i$ gets us at least a $1/k$ fraction of the remaining distance toward the value of $S^*$.

The rest is just inequality hacking to get the final result...
Let $\delta_i = f(S^*) - f(S_i)$. Then we can rewrite the last inequality as

$$\delta_i \leq k[\delta_i - \delta_{i+1}]$$

or equivalently,

$$\delta_{i+1} \leq \left(1 - \frac{1}{k}\right) \delta_i.$$

• Conclude that

$$\delta_\ell \leq \left(1 - \frac{1}{k}\right)^\ell \delta_0$$

$$= \left(1 - \frac{1}{k}\right)^\ell (f(S^*) - f(\emptyset))$$

$$\leq \left(1 - \frac{1}{k}\right)^\ell f(S^*)$$

$$\leq e^{-\ell/k} f(S^*).$$

• (The last step follows from the generally useful fact that $1 - x \leq e^{-x}$.)

• Finally, expanding $\delta_\ell$ gives us

$$f(S_\ell) \geq \left(1 - e^{-\ell/k}\right) f(S^*),$$

as desired.

So what?

• If $\ell = k$, then the greedy algorithm gives a solution that is at least $1 - 1/e \approx 0.63$ times the optimum.

• If we are willing to pick $\ell > k$ items, we can get much closer to the optimum for $k$ items.

• It is known that no algorithm that evaluates $f$ on only a polynomial number of different sets can do better than a $1 - 1/e$ ratio in the worst case.

6 Two Further Fun Results (Without Proofs)

• The result above generalizes MAX-COVER. Is there an analogous result for SET-COVER-like problems?

• What is the analogous problem for general montone submodular functions?

• Roughly, try to get $f$ over some threshold using the smallest set possible.

• For SET-COVER, $f$ is coverage, and we want to reach $f(U) = |G|$, the size of our ground set, using the fewest possible sets.

• Natural analogues exist for more general coverage problems like the examples above.
SET-COVER has a log-factor approximation. Does this result generalize?

- Wolsey (1982) proved the following result for integer-valued, monotone submodular functions $f$.
- Let $q$ be any threshold, $0 \leq q \leq f(U)$. Suppose we run the greedy algorithm above, and let $\ell$ be the least value for which $f(S_\ell) \geq q$.
- Then
  $$\ell \leq (1 + \ln \max_{x \in U} f(\{x\}))|S^*|$$
  where $S^*$ is a set of minimum cardinality s.t. $f(S^*) \geq q$.
- This result also holds if the elements of $U$ have non-uniform cost, and the goal is to minimize the total cost of elements in the solution while exceeding threshold $q$.
- The latter result generalizes the known log-factor approximation for weighted SET-COVER.

One more interesting generalization.

- In the 0-1 KNAPSACK problem, our goal is to maximize the total value of items chosen, but each element has a weight $w_i$, and we cannot exceed total weight $C$.
- If we think of weights as costs and the total weight as a budget, this kind of constraint looks similar to the one from our facility-location problem.
- In general, each $x \in U$ has a cost $c(x)$, and we want to maximize some function $f$ of the chosen set (in this case, total value) while keeping total cost within some budget $C$.
- Not surprisingly, this scenario is called “maximizing $f$ subject to a knapsack constraint.”
- Note that for KNAPSACK itself, $f$ is modeled as a sum of per-item values and hence is modular.

We can get arbitrarily good approximations (an FPTAS) for monotone modular maximization subject to a knapsack constraint. How well can we do for monotone submodular $f$?

- Suppose we define a “value density” analog for the general problem. For $x \in U$, let
  $$d(x \mid S) = \frac{\Delta f(x \mid S)}{c(x)}.$$  
- Now consider the greedy algorithm that starts with $\emptyset$ and, at each step $i$, adds the element $x$ of maximum density with respect to the current set $S_i$ (that does not cause the total cost to exceed $C$), until no more choices are possible.
- The above algorithm does not provide any approximation ratio (it can do arbitrarily badly for KNAPSACK!).
• However, let $S_1$ be the solution obtained by the above algorithm, and let $S_2$ be the solution obtained by ignoring costs and running the previous section’s greedy algorithm until the next choice would exceed the budget $C$.

• Suppose we compute $S_1$ and $S_2$ and return the better of the two.

• (This is roughly analogous to a well-known 2-approximation for KNAPSACK.)

• Lekovec et al. (2007) showed that this heuristic is a $2/(1 - 1/e)$ approximation.

• A somewhat hairier algorithm due to Sviridenko (2004) achieves a $1/(1 - 1/e)$ approximation.