english

CSC2420: Algorithm Design, Analysis and Theory Spring 2019

Allan Borodin

February 7, 2019

Week 5

Announcements:

- Assignment 1 is now complete and the due date has been set to February 25 which is the last date to drop a graduate course without penalty.

Todays agenda

- Briefly mention reverse or backward greedy algorithms.
- Local search
- Exact-Max-k-Sat; oblivious and non-onlvious local search
- Oblivious and non-oblivious local search for WMIS wrt. $k+1$ claw-free graphs.
- Matroids
- Submodular functions
- Greedy and local search for maximizing linear and monotone submodular functions subject to a matroid and other independence constraints.

Local search: the "other" conceptually simplest paradigm

Along with greedy and greedy-like algorithms, *local search* is (for me) one of the two conceptually simplest search/optimization paradigms. Like greedy algorithms, there are many variations of this paradigm.

The vanilla local search paradigm

```
"Initialize" $S$
While there is a "better" solution $S'$
          in the "local neighbourhood" $Nbhd(S)$
  $S := S'$
End While
```

If and when the algorithm terminates, the algorithm has computed a *local optimum*.

Local search as a well defined algorithm

To make local search a precise algorithmic model, we have to say:

1. How are we allowed to choose an initial solution?
2. What constitutes a reasonable definition of a local neighbourhood?
3. What do we mean by "better"?

Answering these questions (especially as to defining a local neighbourhood) will often be quite problem specific.

Towards a more precise definition for local search

- We clearly want the initial solution to be efficiently computed and to be precise we can (for example) say that the initial solution is a random solution, or a greedy solution or adversarially chosen. Of course, in practice we can use any efficiently computed solution.
- We want the local neighbourhood $Nbhd(S)$ to be such that we can efficiently search for a "better" solution (if one exists).
  1. In many problems, a solution S is a subset of the input items or equivalently a $\{0,1\}$ vector, and in this case we often define the $Nbhd(S) = \{S' | d_H(S, S') \leq k\}$ for some "small" k where $d_H(S, S')$ is the Hamming distance.
  2. More generally whenever a solution is a vector over a small domain D, we can use Hamming distance to define a local neighbourhood. Hamming distance k implies that $Nbhd(S)$ can be searched in at most time $|D|^k$.
  3. We can view Ford Fulkerson flow algorithms as local search algorithms where the (possibly exponential size but efficiently search-able) neighbourhood of a flow solution S are flows obtained by adding an augmenting path flow.

What does “better” solution mean? Oblivious and non-oblivious local search

- For a search problem, we would generally have a non-feasible initial solution and “better” can then mean “closer” to being feasible.
- For an optimization problem it usually means being an improved solution which respect to the given objective. For reasons I cannot understand, this has been termed *oblivious* local search. I think it should be called greedy local search.
- For some applications, it turns out that rather than searching to improve the given objective function, we search for a solution in the local neighbourhood that improves a related potential function and this has been termed non-oblivious local search.
- In searching for an improved solution, we may want an arbitrary improved solution, a random improved solution, or the best improved solution in the local neighbourhood.
- For efficiency we sometimes insist that there is a “sufficiently better” improvement rather than just better.

Exact Max-k-Sat

- **Given:** An exact k-CNF formula
$$F = C_1 \wedge C_2 \wedge \ldots \wedge C_m,$$
where $C_i = (\ell_i^1 \vee \ell_i^2 \ldots \vee \ell_i^k)$ and $\ell_i^j \in \{x_k, \bar{x}_k \,|\, 1 \leq k \leq n\}$.

- In the weighted version, each C_i has a weight w_i.

- **Goal:** Find a truth assignment τ so as to maximize
$$W(\tau) = w(F \,|\, \tau),$$
the weighted sum of satisfied clauses w.r.t the truth assignment τ.

- It is NP hard to achieve an approximation better than $\frac{7}{8}$ for exact Max-3-Sat and hence that hard for the non exact version of Max-k-Sat for $k \geq 3$.

- Max-2-Sat can be approximated to within a factor $\approx .87856$.

The natural oblivious local search

- A natural oblivious local search algorithm uses a Hamming distance d neighbourhood:
  $N_d(\tau) = \{\tau' : \tau \text{ and } \tau' \text{ differ on at most } d \text{ variables } \}$

Oblivious local search for Exact Max-k-Sat

```
Choose any initial truth assignment $\tau$
WHILE there exists $\hat{\tau} \in N_d(\tau)$ such that $W(\hat{\tau}) > W(\tau)$
    $\tau := \hat{\tau}$
END WHILE
```

How good is this oblivious local search algorithm?

- Note: Following the standard convention for Max-Sat, I am using approximation ratios < 1.
- It can be shown that for $d = 1$, the approximation ratio for Exact-Max-2-Sat is $\frac{2}{3}$.
- In fact, for every exact 2-Sat formula, the algorithm finds an assignment τ such that $W(\tau) \geq \frac{2}{3}\sum_{i=1}^{m} w_i$, the weight of all clauses, and we say that the "totality ratio" is at least $\frac{2}{3}$.
- More generally for Exact Max-k-Sat the ratio is $\frac{k}{k+1}$. This ratio is essentially a tight ratio for any $d = o(n)$.
- This is in contrast to an online greedy algorithm derived from a naive randomized algorithm that achieves totality ratio $(2^k - 1)/2^k$.
- "In practice", the local search algorithm often performs better than the naive greedy and one could always start with the greedy algorithm and then apply local search.

Analysis of the oblivious local search for Exact Max-2-Sat

- Let τ be a local optimum and let
  - S_0 be those clauses that are not satisfied by τ
  - S_1 be those clauses that are satisfied by exactly one literal by τ
  - S_2 be those clauses that are satisfied by two literals by τ

- Let $W(S_i)$ be the corresponding weight.

Analysis of obvlivious Exact-Max-2-Sat local search continued

- We will say that a clause involves a variable x_j if either x_j or $\bar{x}_j$ occurs in the clause. Then for each j, let

- A_j be those clauses in S_0 involving the variable x_j.

- B_j be those clauses C in S_1 involving the variable x_j such that it is the literal x_j or $\bar{x}_j$ that is satisfied in C by τ.

- C_j be those clauses in S_2 involving the variable x_j.

- Let $W(A_j), W(B_j), W(C_j)$ be the corresponding weights.

Analysis of the oblivious local search (continued)

- Summing over all variables x_j, we get

- $2W(S_0) = \sum_j W(A_j)$ noting that each clause in S_0 gets counted twice.

- $W(S_1) = \sum_j W(B_j)$

- Given that τ is a local optimum, for every j, we have

  $$W(A_j) \leq W(B_j)$$

  or else flipping the truth value of x_j would improve the weight of the clauses being satisfied.

- Hence (by summing over all j),

  $$2W_0 \leq W_1.$$

Finishing the analysis

- It follows then that the ratio of clause weights not satisfied to the sum of all clause weights is

  $\frac{W(S_0)}{W(S_0)+W(S_1)+W(S_2)} \leq \frac{W(S_0)}{3W(S_0)+W(S_2)} \leq \frac{W(S_0)}{3W(S_0)}$

- It is not easy to verify but there are examples showing that this $\frac{2}{3}$ bound is essentially tight for any N_d neighbourhood for $d = o(n)$.

- It is also claimed that the bound is at best $\frac{4}{5}$ whenever $d < n/2$. For $d = n/2$, the algorithm would be optimal.

- In the weighted case, we have to worry about the number of iterations. And here we can speed up the termination by insisting that any improvement has to be sufficiently better.

Using the proof to improve the algorithm

Aside: Using adversarial examples and viewing algorithms as a *game* against an advesray is an idea that is now vefy active in "adversarial learning".

- We can learn something from this proof to improve the performance.
- Note that we are not using anything about $W(S_2)$.
- If we could guarantee that $W(S_0)$ was at most $W(S_2)$ then the ratio of clause weights not satisfied to all clause weights would be $\frac{1}{4}$.
- Claim: We can do this by enlarging the neighbourhood to include $\tau' =$ the complement of τ.

The non-oblivious local search

- We consider the idea that satisfied clauses in S_2 are more valuable than satisfied clauses in S_1 (because they are able to withstand any single variable change).

- The idea then is to weight S_2 clauses more heavily.

- Specifically, in each iteration we attempt to find a $\tau' \in N_1(\tau)$ that improves the potential function
$$\frac{3}{2}W(S_1) + 2W(S_2)$$
instead of the oblivious $W(S_1) + W(S_2)$.

- More generally, for all k, there is a setting of scaling coefficients $c_1, \ldots, c_k$, such that the non-oblivious local search using the potential function $c_1 W(S_1) + c_2 W(S_2 + \ldots + c_k W(S_k)$ results in approximation ratio $\frac{2^k-1}{2^k}$ for exact Max-k-Sat.

Sketch of $\frac{3}{4}$ totality bound for the non oblivious local search for Exact Max-2-Sat

- Renaming variables, we can assume that τ is the all true assignment.
- Let $P_{i,j}$ be the weight of all clauses in S_i containing x_j.
- Let $N_{i,j}$ be the weight of all clauses in S_i containing $\bar{x}_j$.
- Here is the key observation for a local optimum τ wrt the stated potential:
  .1in
  $-\frac{1}{2}P_{2,j} - \frac{3}{2}P_{1,j} + \frac{1}{2}N_{1,j} + \frac{3}{2}N_{0,j} \leq 0$
- Summing over variables $P_1 = N_1 = W(S_1)$, $P_2 = 2W(S_2)$ and $N_0 = 2W(S_0)$ and using the above inequality we obtain $3W(S_0) \leq W(S_1) + W(S_2)$

Some experimental results concerning Max-Sat

- Of course, one wonders whether or not a worse case approximation will actually have a benefit in "practice".
- "In practice", local search becomes more of a "heuristic" where one uses various approaches to escape (in a principled way) local optima and then continuing the local search procedure. Perhaps the two most commonly used versions are Tabu Search and Simulated Annealing.
- Later, we will also discuss methods based on online algorithm and "random walks" and other randomized methods (and their derandomizations). .
- We view these algorithmic ideas as starting points.
- But for what it is worth, here are some 2010 experimental results both for artifically constructed instances and well as for one of the many benchmark test sets for Max-Sat.
- Recent experimental results by Poloczek and Willamson show that various ways to use greedy and local search algorithms can compete (wrt. various test sets) with "state of the art" simulated annealing algorithms and walk-sat algorithms while using much less time.

Experiment for unweighted Max-3-Sat

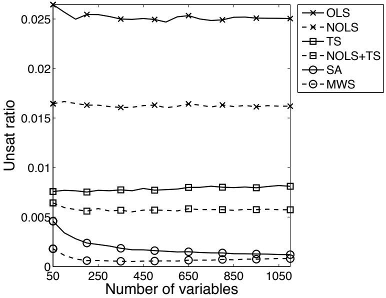
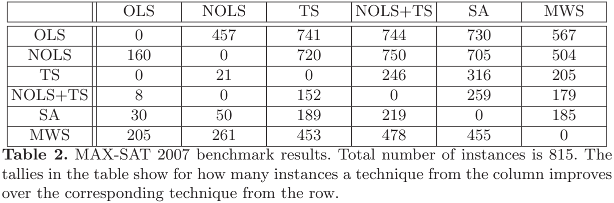


Fig. 1. Average performance when executing on random instances of exact MAX-3-SAT.

[From Pankratov and Borodin]

Experiment for Benchmark Max-Sat



	OLS	NOLS	TS	NOLS+TS	SA	MWS
OLS	0	457	741	744	730	567
NOLS	160	0	720	750	705	504
TS	0	21	0	246	316	205
NOLS+TS	8	0	152	0	259	179
SA	30	50	189	219	0	185
MWS	205	261	453	478	455	0

Table 2. MAX-SAT 2007 benchmark results. Total number of instances is 815. The tallies in the table show for how many instances a technique from the column improves over the corresponding technique from the row.

[From Pankratov and Borodin]

More experiments for benchmark Max-Sat

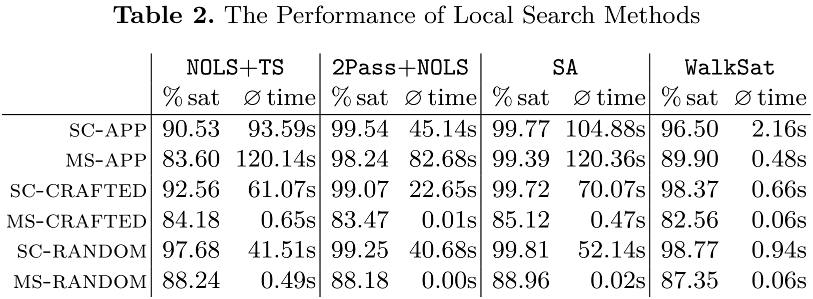


Table 2. The Performance of Local Search Methods

	NOLS+TS		2Pass+NOLS		SA		WalkSat	
	% sat	∅ time	% sat	∅ time	% sat	∅ time	% sat	∅ time
SC-APP	90.53	93.59s	99.54	45.14s	99.77	104.88s	96.50	2.16s
MS-APP	83.60	120.14s	98.24	82.68s	99.39	120.36s	89.90	0.48s
SC-CRAFTED	92.56	61.07s	99.07	22.65s	99.72	70.07s	98.37	0.66s
MS-CRAFTED	84.18	0.65s	83.47	0.01s	85.12	0.47s	82.56	0.06s
SC-RANDOM	97.68	41.51s	99.25	40.68s	99.81	52.14s	98.77	0.94s
MS-RANDOM	88.24	0.49s	88.18	0.00s	88.96	0.02s	87.35	0.06s

Figure: Table from Poloczek and Williamson 2017

Note: *2Pass* is a deterministic "2-pass online algorithm" that is derived from a randomized online algorithm that we will discuss soon.

Oblivious and non-oblivious local search for $k+1$ claw free graphs

- We again consider the weighted max (independent) vertex set in a $k+1$ claw free graph. (Recall the argument generalizing the approximation ratio for the k set packing problem.)
- The standard greedy algorithm and the 1-swap oblivious local search both achieve a $\frac{1}{k}$ approximation for the WMIS in $k+1$ claw free graphs. Here we define an "ℓ-swap" oblivous local search by using neighbrourhoods defined by bringing in a set S of up to ℓ vertices and removing all vertices adjacent to S.

  NOTE: We will continue to use fractional approximation ratios for the maximization problems being considered this week.
- For the unweighted MIS, Halldórsson shows that a a 2-swap oblivious local search will yield a $\frac{2}{k+1}$ approximation.

Berman's [2000] non-oblivious local search

- For the weighted MIS, the "ℓ-swap" oblivous local search results (essentially) in an $\frac{1}{k}$ locality gap for any constant ℓ.
- Chandra and Halldóssron [1999] show that by first using a standard greedy algorithm to initialize a solution and then using a "greedy" k-swap oblivious local search, the approximation ratio improves to $\frac{3}{2k}$.
- Can we use non-oblivious local search to improve the locality gap? Once again given two solutions V_1 and V_2 having the same weight, when is one better than the other?

Berman's [2000] non-oblivious local search

- For the weighted MIS, the "ℓ-swap" oblivous local search results (essentially) in an $\frac{1}{k}$ locality gap for any constant ℓ.
- Chandra and Halldóssron [1999] show that by first using a standard greedy algorithm to initialize a solution and then using a "greedy" k-swap oblivious local search, the approximation ratio improves to $\frac{3}{2k}$.
- Can we use non-oblivious local search to improve the locality gap? Once again given two solutions V_1 and V_2 having the same weight, when is one better than the other?
- Intuitively, if one vertex set V_1 is small but vertices in V_1 have large weights that is better than a solution with many small weight vertices.

Berman's [2000] non-oblivious local search

- For the weighted MIS, the "ℓ-swap" oblivous local search results (essentially) in an $\frac{1}{k}$ locality gap for any constant ℓ.
- Chandra and Halldóssron [1999] show that by first using a standard greedy algorithm to initialize a solution and then using a "greedy" k-swap oblivious local search, the approximation ratio improves to $\frac{3}{2k}$.
- Can we use non-oblivious local search to improve the locality gap? Once again given two solutions V_1 and V_2 having the same weight, when is one better than the other?
- Intuitively, if one vertex set V_1 is small but vertices in V_1 have large weights that is better than a solution with many small weight vertices.
- Berman chooses the potential function $g(S) = \sum_{v \in S} w(v)^2$. Ignoring some small ϵ's, his k-swap non-oblivious local search achieves a locality gap of $\frac{2}{k+1}$ for WMIS on $k+1$ claw-free graphs.

The (metric) facility location and k-median problems

- Two extensively studied problems in operations research and CS algorithm design are the related uncapacitated facility location problem (UFL) and the k-median problem. In what follows we restrict attention to the (usual) metric case of these problems defined as follows:

Definition of UFL

Input: (F, C, d, f) where F is a set of faciltites, C is a set of clients or cities, d is a metric distance function over $F \cup C$, and f is an opening cost function for facilities.
Output: A subset of facilities F' minimizing $\sum_{i \in F'} f_i + \sum_{j \in C} d(j, F')$ where f_i is the opening cost of facility i and $d(j, F') = min_{i \in F'} d(j, i)$.

- In the capacitated version, facilities have capacities and cities can have demands (rather than unit demand). The constraint is that a facility can not have more assigned demand than its capacity so it is not possible to always assign a city to its closest facility.

UFL and k-median problems continued

Deifnition of k-median problem

Input: (F, C, d, k) where F, C, d are as in UFL and k is the number of facilities that can be opened.
Output: A subset of facilities F' with $|F'| = k$ minimizing $\sum_{j \in C} d(j, F')$

- These problems are clearly well motivated. Moreover, they have been the impetus for the development of many new algorithmic ideas which we will hopefully at least touch upon throughout the course.
- There are many variants of these problems and in many papers the problems are defined so that $F = C$; that is, any city can be a facility. If a solution can be found when F and C are disjoint then there is a solution for the case of $F = C$.

UFL and k-median problems continued

- It is known (Guha and Khuller) that UFL is hard to approximate to within a factor better than 1.463 assuming NP is not a subset of $DTIME(n^{\log\log n})$ and the k-median problem is hard to approximate to within a factor better than $1 + 1/e \approx 1.736$ (Jain, Mahdian, Saberi).
- The UFL problem is better understood than k-median. After a long sequence of improved approximation results the current best polynomial time approximation is 1.488 (Li, 2011).
- For k-median, until recently, the best approximation was by a local search algorithm. Using a p-flip (of facilities) neighbourhood, Arya et al (2001) obtain a $3 + 2/p$ approximation which yields a $3 + \epsilon$ approximation running in time $O(n^{2/\epsilon})$.
- Li and Svennsson (2013) have obtained a $(1 + \sqrt{3} + \epsilon)$ approximation running in time $O(n^{1/\epsilon^2})$. Surprisingly, they show that an α approximate "pseudo solution" using $k + c$ facilities can be converted to an $\alpha + \epsilon$ approximate solution running in $n^{O(c/\epsilon)}$ times the complexity of the pseudo solution.

Some additional comments on local search

- An interesting (but probably difficult) open problem is to use a non oblivious local search for either the UFL or k-median problems.

- But suffice it to say now that local search is the basis for many practical algorithms, especially when the idea is extended by allowing some well motivated ways to escape local optima (e.g. simulated annealing, tabu search) and combined with other paradigms.

- Although local search with all its variants is viewed as a great "practical" approach for many problems, local search is not often theoretically analyzed. It is not surprising then that there hasn't been much interest in formalizing the method and establishing limits.

- LP is itself often solved by some variant of the simplex method, which can also be thought of as a local search algorithm, moving fron one vertex of the LP polytope to an adjacent vertex.
  - No such method is known to run in polynomial time in the worst case.

Matroids and independence systems

- Independence systems and matroids
  Let $M = (U, \mathcal{F})$, where U is a set of elements, $\mathcal{F} \subseteq 2^{|U|}$; $I \in \mathcal{F}$ is called an independent set.
  An (hereditary) independence system satisfies the following properties:
  1) $\varnothing \in \mathcal{F}$; often stated although not necessary if $\mathcal{F} \neq \varnothing$
  2) $S \subseteq T, T \in \mathcal{F} \Rightarrow S \in \mathcal{F}$
- A matroid is an independence system that also satisfies:
  3) $S, T \in \mathcal{F}, |S| < |T|$, then $\exists x \in T \setminus S$ such that $S \cup \{x\} \in \mathcal{F}$
  Equivalently, matroids are hereditary independence systems that saistify the following rank property:
  3') All maximal independent sets have the same cardinality r and r is called the rank of the matroid. A maximal independent set is called a basis.
- Another equivalent definition, is the following basis exchange property: If S and T are different bases, then
  $\forall x \in S, \exists y \in T : S \setminus \{x\} \cup \{y\}$ is independent.

Examples of matroids

- Sets having at most k elements constitute the independent sets in a uniform matroid
- Other common examples, include
  1. Partition matroids extend the uniform matroid (i.e. cardinality constraint) as follows: The universe U is a disjoint union $U_1 \cup U_2 \ldots \cup U_r$ and there are individual cardiality constraints k_i for each block U_i of the partition.
  2. Graphic matroids: U is the set of edges E in a graph $G = (V, E)$ and $E' \subseteq E$ is independent if $G = (V, E')$ is acyclic.
  3. Linear matroids where U is a set of vectors in a vector space and I is independent in the usual sense of linear independence. Indeed, matroids are a combinatorial abstraction of linear independence.

In each of these applications, one should verify the matroid properties and, in particular, the exchange property. For example, for a graphic matroid, if S is a maximal independent set of edges in a connected graph, then S is an MST and any edge $e \notin S$ will create a unique cycle.

Matroids and *the* natural greedy algorithm.

- There is an elegant development starting in the 1950's with the work of Rado [1957], Gale 1968 and Edmonds [1970, 1971], (extended by Korte and Lovász [1981, 1984], and others) as to contexts in which *"the natural"* or *standard* greedy algorithm will produce an optimal solution or "good" appromximation for maximizing functions in many applications.
- Here the well known example is the minimum (resp. maximum) spanning tree problem where the edges of a graph are the elements and the indepedent sets are forests in the graph. Kruskal's greedy algorithm is the natural greedy MST algorithm. which sorts edges by the non-decreasing weight. weights (resp. by non-increasing weight for the maximation problem) and then in this order adds an edge to the solution whenever it does not create a cycle.

Rado-Edmonds characterization of matroids in terms of the standard greedy algorithm

The “standard greedy algorithm” *Greedy* for maximizing a linear function (over some universe U) subject to some independence constraint:

Standard Greedy for a Maximization Problem

```
Let $S := \varnothing$
While $\exists u : S \cup \{u\}$ is independent
    $S := S \cup \{u\}$ where $u$ is an element of largest weight
    such that $S \cup \{u\}$ is independent
End While
```

Rado: For independence in a matroid M, *Greedy* is an optimal algorithm for maximizing the weight of basis (i.e. maximizing a linear function subject to a matroid constraint).

Rado-Edmonds characterization continued

Note: For linear functions, the problem of *minimizing* a linear function subject to being a basis is an equivalent problem.

Standard Greedy for a Minimization Problem

```
Let $S := \varnothing$
While $\exists u : S \cup \{u\}$ is independent
    $S := S \cup \{u\}$ where $u$ is an element of smallest weight
    such that $S \cup \{u\}$ is independent
End While
```

The Rado result can be seen as an abstraction of Kruskal's min spanning tree (forest) greedy algorithm.

Edmonds: For an hereditary independence system M, if *Greedy* is optimal for every linear function (i.e. for every set of weights) then M is a matroid.

More general independence systems

As we noted, there are many equivalent ways to define matroids. In particular, the exchange property immediately implies that in a matroid M every maximal independent set (called a *base*) has the same cardinality, the *rank* of M. We can also define a base for any subset $S \subseteq U$. Matroids are those independence systems where all bases have the same cardinality. Let k be a positive integer. A (Jenkyns) k-independence system satisfies the weaker property that for any set S and two bases B and B' of S, $\frac{|B|}{|B'|} \leq k$. Matroids are precisely the case of $k = 1$.
Examples:

- The intersection of k matroids
- Mestre's k-extendible systems where the matroid exchange property is replaced by : If $S \subseteq T$ and $S \cup \{u\}$ and T are independent, then $\exists Y \subseteq T - S : |Y| \leq k$ and $T - Y \cup \{u\}$ is independent.
- Independent sets in $k + 1$ claw free graphs. In such graphs, the neighbourhood of every node has at most k independent vertices.

The standard greedy algorithm for k-systems and $k+1$ claw free graphs

Jenkyns shows that the standard greedy algorithm is a $\frac{1}{k}$-approximation for maximizing a linear function subject to independence in a k-independence system. It follows (as we already know from our earlier study of greedy algorithms) that the standard greedy algorithm is a $\frac{1}{k}$-approximation for independence in a $k+1$ claw free graph.

The same approximation applies for a 1-exchange local search algorithm.

Submodular functions

Let U be a universe. In what follows, we will only be interested in set functions that satisfy $f(S) \geq 0$ for all $S \subseteq U$. We will also assume functions are *normalized* in that $F(\varnothing) = 0$, These assumptions are not that essental but they are standard and without these assumptions statements and proofs become somewwhat more complex.

- A *sublinear set function* satisfies the property that $f(S \cup T) \leq f(S) + f(T)$ for all subsetes S, T of U.
- When $f(S \cup T) + f(S \cap T) = f(S) + f(T)$, the function is a linear (also called modular) function.
- A *submodular set function* $f : U \rightarrow \mathbb{R}$ satisfies the following property: $f(S \cup T) + f(S \cap T) \leq f(S) + f(T)$
- It follows that modular set functions are submodular and submodular functions are sublinear.
- Submodular functions can be monotone or non-monotone. A monotone submodular function also satisifes the property that $f(S) \leq f(T)$ whenever $S \subseteq T$.

An alternative characterization and examples of submodular functions

Submodular functions satisfy and can also be defined as those satisfying a *decreasing marginal gains* property. Namely,
For $S \subset T, f(T \cup \{x\} - f(T) \leq f(S \cup \{x\}) - f(S)$. That is, adding additional elements has decreasing (more precisely, non increasing) marginal gain for larger sets.

Most applications of submodular functions are for monotone submodular functions. For example, in practice, when we are obtaining results from a search engine, as we obtain more and more results, we tend to obtain less additional value.

Modular functions are monotone.

The rank function of a matroid is a monotone submodular function.

The two most common examples of non-monotone submodular functions are max-cut and max-di-cut (i.e., max directed cut)

Monotone submodular function maximization

- The monotone problem is only interesting when the submodular maximization is subject to some constraint.
- Probably the simplest and most widely used constraint is a cardinality constraint; namely, to maximize $f(S)$ subject to $|S| \leq k$ for some k and since f is monotone this is the same as the constraint $f(S) = k$.
- Following Cornuéjols, Fisher and Nemhauser [1977] (who study a specific submodular function), Nemhauser, Wolsey and Fisher [1978] show that the standard greedy algorithm achieves a $1 - \frac{1}{e}$ approximation for the cardinality constrained monotone problem. More precisely, for all k, the standard greedy is a $1 - (1 - \frac{1}{k})^k$ approximation for a cardinality k constraint.

Standard greedy for submodular functions wrt cardinality constraint

```
$S := \varnothing$
While $|S| < k$
    Let $u$ maximize $f(S \cup \{u\}) - f(S)$
    $S := S \cup \{u\}$
End While
```

Proof: greedy approx for monotone submodular maximization subject to cardinality constraint

We want to prove the $1 - (1 - \frac{1}{k})^k$ approximation bound.
Let S_i be the set after i iterations of the standard greedy algorithm and let $S^* = \{x_1, \ldots, x_k\}$ be an optimal seti so that $OPT = f(S^*)$. For any set S and element x, let $f_S(x) = f(S \cup \{x\}) - f(S)$ be the marginal gain by adding x to S. The proof uses the following sequence of inequalities:

$$
\begin{aligned}
f(S^*) &\leq f(S_i \cup S^*) \text{ by monotonicity} \\
&\leq f(S_i) + (f(S_i) \cup \{x_1\} - f(S_i)) + (f(S_i) \cup \{x_1, x_2\} - f(S_i \cup \{x_1\})) + \ldots \\
&\qquad \text{(by submodularity; question 5(a) on assignment)} \\
&\leq f(S_i) + f_{S_i}(x_1) + f_{S_i}(x_2) + \ldots f_{S_i}(x_k) \\
&\qquad \text{(again by submodularity)} \\
&\leq f(S_i) + k \cdot (f(S_{i+1} - f(S_i)) \text{ by the greedy assumption}
\end{aligned}
$$

Equivalently, $f(S_{i+1} \geq f(S_i) + \frac{1}{k}(f(OPT) - f(S_i)$

The proof is completed by showing $f(S_i) \geq (1 - (1 - \frac{1}{k})^i) \cdot OPT$ by induction on i.

Where we ended on February 7

The lecture on Thursday, February 7 basically ended on slide 36 with the statement of the standard greedy algorithm for maximizing a monotone submodular subject to a cardinality constraint. This algorithm provides a $1 - \frac{1}{e}$-approximation for any caridnality constraint. I added some additional slides for the purpose of the assignment. We will continue next week with a sketch of the proof of this approximation bound.

Generalizing to a matroid constraint

- Nemhauser and Wolsey [1978] showed that the $1 - \frac{1}{e}$ approximation is optimal in the sense that an exponential number of value oracle queries would be needed to beat the bound for the cardinalily constraint.
- Furthermore, Feige [1998] shows it is NP hard to beat this bound even for the explicitly represented maximum k-coverage problem.
- Following their first paper, Fisher, Nemhauser and Wolsey [1978] extended the cardinality constraint to a matroid constaint.
- Fisher, Nemhauser and Wolsey show that both the standard greedy algorithm and a 1-exchange local search algorithm (that will follow) achieve a $\frac{1}{2}$ approximation for maximzing a monotone submodular function subject to an arbitrary matroid constraint.
- They also showed that this bound was tight for the greedy and 1-exchange local search algorithms.

Monotone submodular maximization subject to a matroid constraint

We need some additional facts about matroids and submodular functions.

- Brualdi [1969] Let O and S be two independent sets in a matroid of the same size (in particular they could be two bases). Then there is a bijection π between $O \setminus S$ and $S \setminus O$ such that for all $x \in O, (S \setminus \{\pi(x)\}) \cup x$ is independent.
- We have the following facts for a submodular function f on a ground set U:
  1. Let $C = \{c_1, \ldots, c_\ell\} \subseteq U \setminus S$. Then
     $$\sum_{i=1}^{\ell} [f(S + c_i) - f(S)] \geq f(S \cup C) - f(S)$$
  2. Let $\{t_1, \ldots, t_\ell\}$ be elements of S. Then
     $$\sum_{i=1}^{\ell} [f(S) - f(S \setminus \{t_i\}] \leq f(S)$$

The 1-exchange local search algorithm

We can start with any basis S (eg using the natural greedy algorithm). Then we keep trying to find an element of $x \notin S$ such that $(S \setminus \{\pi(x)\}) \cup \{x\} > f(S)$. Here π is the bijection as in Brualdi's result.

The previous local seach algorithm provides a $\frac{1}{2}$-approximation for maximizing a monotone submodular funstion.
Now let S be a local optimum and O an optimal solution. By local optimality, for all $x \in O \setminus S$, we have

$$f(S) \geq f((S \setminus \{\pi(x)\}) \cup \{x\})$$

Subtracting $(S \setminus \{\pi(x)\})$ from both sides, we have

$$f(S) - f(S \setminus \{\pi(x)\}) \geq f((S \setminus \{\pi(x)\}) \cup \{x\}) - f(S \setminus \{\pi(x)\})$$

From submodularity,

$$f((S \setminus \{\pi(x)\}) \cup \{x\}) - (S \setminus \{\pi(x)\}) \geq f(S \cup \{x\}) - f(S)$$

Thus for all $x \in O \setminus S$

$$f((S \setminus \{\pi(x)\} \geq f(S \cup \{x\}) - f(S)$$

Completing the local search approximation

Summing over all such x yields
$\sum_{x\in O\setminus S}[f(S)-f(S\setminus\{\pi(x)\})] \geq \sum_{x\in O\setminus S}[f(S\cup\{x\})-f(S)]$
Applying the first fact on slide 28 to the right hand side of this inequality and the second fact to the left hand side, we get

$$f(S) \geq f(S\cup(O\setminus S)) - f(S) = f(O\cup S) - f(S) \geq f(O) - f(S)$$

which gives the desired $\frac{1}{2}$-approximation.