CSC373: Algorithm Design, Analysis and Complexity Winter/Spring 2020

Allan Borodin and Sara Rahmati

Week of March 23-27, 2020

Announcements: The revised grading scheme

- Combining both the day time and evening sections there were 168 students who voted.
- 108 voted yes (approving the proposed revised grading scheme); This is $\approx$ 64%yes
- 60 voted no; This is $\approx$ 34%no
- The revised grading scheme has been approved and we will be posting a fourth asasignment A4.

Week 11 : Other Announcements

- We will continue to provide information during this period on the web page, on piazza and on these lecture slides. We will also hold office hours via skype. My skype address is abborodin@gmail.com.
  I am also holding office hours via zoom on Wednesdays at 3PM. This is a slot reserved for CSC303 lectures and tutorials so you need the 303 zoom site password. It is 038954
- Assignment 3 is due Thursday, April 2 at 4:59.
- Assignment 4 is due Thursday, April 16 at 4:59
- In many cases, students are answering questions correctly on piazza and we appreciate that being done.
- While we are always concerned about plagarism, we are particularly concerned due to the additional pressure we are all feeling.

This weeks agenda

1. We begin this week (i.e., first hour) discussing local search and approximation algorithms. This is a big topic but we will only quickly mention some basic ideas and a couple of applications. In partcular, we will discuss the Exact Max-2-Sat and Max-Sat problems where will introduce the concept of “non-oblivious” local search.
2. After discussing local search, this week will be devoted to the final main topic of the course, which is **randomized algorithms**. In particular,
   - The why of randomized algorithms
   - Polynomial identity testing; the symbolic determinant problem.
   - 0-sided vs 1-sided vs 2-sided error; amplifying probability of correctness
   - Naive randomized algorithm for Exact Max-2-Sat and ts de-randomization.
   - A more recent max-sat algorithm
   - The complexity of k-SAT and random walk algorithms for k-SAT.
   - Randomized rounding of LPs.

This weeks agenda continued and next week's agenda

Our agenda for this week is overly ambitious and almost surely will spill over to next week.

There are a couple of underlying themes in what we have discussed before and will continue to try to mention. Namely,

- Algorithms and their limitations suggest both extensions to other problems and new algorithms to overcome some limitations.
- There is often a divide between theorectical worst case analysis and observed performance "in practice". In particular, "simple algorithms" often have relatively good performance, say not matching "state of the art" on applications but sometimes quite competitive in performance while being more efficient and easier to implement.

After finishing up material from this week, next week will be an overview of a number of important topics in algorithm design and analysis that we will not discuss in this course. Algorithm design and analysis continues to be a very active field within computer science.

Local Search: another conceptually simple approach

We now begin a discussion of *local search* which for me, along with greedy algorithms, is one of the two conceptually simplest search/optimization paradigms. (We briefly mentioned local search when discussing flows.)

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*. To make this 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 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. As we previously discussed, 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.

The weighted max cut problem

- Our first local search algorithm will be for the (weighted) max cut problem defined as follows:

The (weighted) max-cut problem

- Given a (undirected) graph $G = (V, E)$ and in the weighted case the edges have non negative weights.
- **Goal:** Find a partition (A, B) of V so as to maximize the size (or weight) of the cut $E' = \{(u, v) | u \in A, v \in B, (u, v) \in E\}$.

- We can think of the partition as a characteristic vector χ in $\{0, 1\}^n$ where $n = |V|$. Namely, say $\chi_i = 1$ iff $v_i \in A$.
- Let $N_d(A, B) = \{(A', B') |$ the characteristic vector of (A') is Hamming distance at most d from $(A)\}$
- So what is a natural local search algorithm for (weighted) max cut?

A natural oblivious local search for weighted max cut

Single move local search for weighted max cut

```
Initialize $(A, B)$ arbitrarily
WHILE there is a better partition $(A', B') \in N_1(A, B)$
    $(A, B) := (A', B')$
END WHILE
```

- This single move local search algorithm is a $\frac{1}{2}$ approximation; that is, when the algorithm terminates, the value of the computed local optimum will be at least half of the (global) optimum value.
- In fact, if W is the sum of all edge weights, then $w(A, B) \geq \frac{1}{2} W$.
- This kind of ratio is sometimes called the absolute ratio or totality ratio and the approximation ratio must be at least this good.
- The worst case (over all instances and all local optima) of a local optimum to a global optimum is called the locality gap.
- It may be possible to obtain a better approximation ratio than the locality gap (e.g. by a judicious choice of the initial solution) but the approximation ratio is at least as good as the locality gap.

Proof of totality gap for the max cut single move local search

- The proof is based on the following property of any local optimum:

$$\sum_{v \in A} w(u,v) \leq \sum_{v \in B} w(u,v) \text{ for every } u \in A$$

- Summing over all $u \in A$, we have:

$$2 \sum_{u,v \in A} w(u,v) \leq \sum_{u \in A, v \in B} w(u,v) = w(A,B)$$

- Repeating the argument for B we have:

$$2 \sum_{u,v \in B} w(u,v) \leq \sum_{u \in A, v \in B} w(u,v) = w(A,B)$$

- Adding these two inequalities and dividing by 2, we get:

$$\sum_{u,v \in A} w(u,v) + \sum_{u,v \in B} w(u,v) \leq w(A,B)$$

- Adding $w(A,B)$ to both sides we get the desired $W \leq 2w(A,B)$.

The complexity of the single move local search

- **Claim:** The local search algorithm terminates on every input instance.
  - Why?

The complexity of the single move local search

- **Claim:** The local search algorithm terminates on every input instance.
  - Why?

- Although it terminates, the algorithm could run for exponentially many steps. We saw a similar phenomena concerning some Ford Fulkerson implementations.

- It seems to be an open problem if one can find a local optimum in polynomial time.

- However, we can achieve a ratio as close to the stated $\frac{1}{2}$ totality ratio by only continuing when we find a solution (A', B') in the local neighborhood which is "sufficiently better". Namely, we want

$$w(A', B') \geq (1+\epsilon)w(A, B) \text{ for any } \epsilon > 0$$

- This results in a totality ratio $\frac{1}{2(1+\epsilon)}$ with the number of iterations bounded by $\frac{n}{\epsilon} \log W$.

Final comment on this local search algorithm

- It is not hard to find an instance where the single move local search approximation ratio is $\frac{1}{2}$.
- Furthermore, for any constant d, using the local Hamming neighbourhood $N_d(A, B)$
still results in an approximation ratio that is essentially $\frac{1}{2}$.
And this remains the case even for $d = o(n)$.
- It is an open problem as to what is the best "combinatorial algorithm" that one can achieve for max cut.
- There is a vector program relaxation of a quadratic program that leads to a .878 approximation ratio.

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 for the non exact versions of Max-k-Sat for $k \geq 3$.

The natural oblivious local search

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

- Note: 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 a naive greedy algorithm (which we will soon see) derived from a 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 (for example) a 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.

- 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 C 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 C 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),

$$2W(S_0) \leq W(S_1).$$

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 $d < n/2$.

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

Using the proof to improve the algorithm

- 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

- A thought experiment: Given two solutions (ie., truth assignments) that yield the same objective value, why might we prefer one to the other?

The non-oblivious local search

- A thought experiment: Given two solutions (ie., truth assignments) that yield the same objective value, why might we prefer one to the other?
- 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

- 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:

$$-\frac{1}{2}P_{2,j} - \frac{3}{2}P_{1,j} + \frac{1}{2}N_{1,j} + \frac{3}{2}N_{0,j} \leq 0$$

- 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

$$3W(S_0) \leq W(S_1) + W(S_2)$$

The divide between theory and practice

In the first week of this course, I indicated that our focus will be on worst case analysis. The algorithmic paradigms we consider are a starting point for "algorithms in practice".

For certain problems, there are popular benchmarks (and annual or frequent competitions). One such problem is Max-Sat. The winners of these compeitions are usually algorithms that are developed p from some basic paradigms and then become highly tuned using various heuristics.

For Max-Sat, the state of the art are various implementatiions of simulated annealing (SA) and WalkSat. SA (and Tabu Search TS) are local search algorithms that use principled ideas for escaping local optima. WalkSat is based on random walks.

In addition to the oblivious (OLS) and non-oblivious (NOLS) local search algorithms, there is a 2 pass algorithm algorithm that is the "de-randomization" of a randomized one pass "online" algorithm.

How competitive are simple algorithms against the state of the art?

The following experiments show the applicability of the non-oblivious idea and furthermore how relatively simple combinatorial algorithms can be competitive with state of the art (more time consumming) algorithms.

In applications (and life) there are usual tradeoffs that need to be made (e.g., performance vs complexity).

Some comparative experimental results for local search based Max-Sat algorithms

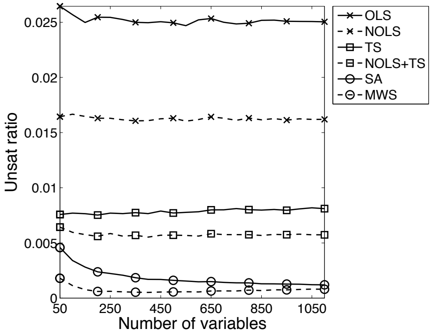


Fig. 1. Average performance when executing on random instances of exact MAX-3-SAT.

[From Pankratov and Borodin 2010]

More experiments 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 2010]

More experiments for benchmark Max-Sat

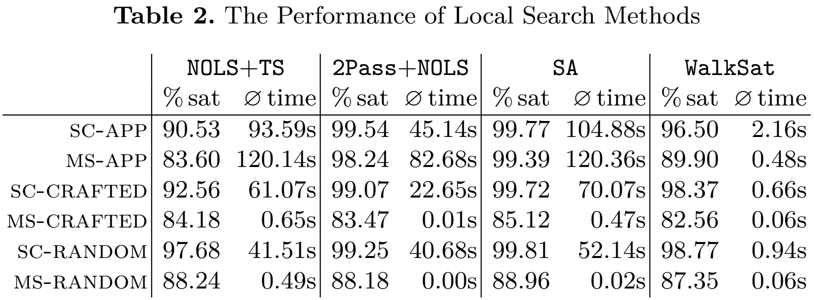


Table 2. The Performance of Local Search Methods

	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

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

- We again consider the maximum weighted independent set problem in a $k+1$ claw free graph.
- 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" oblivious local search by using the neighbourhood defined by bringing in a set S of up to ℓ vertices and removing all vertices adjacent to S.
- For the unweighted MIS, Halldórsson shows that 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" oblivious local search results (essentially) in a $\frac{1}{k}$ 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" oblivious local search results (essentially) in a $\frac{1}{k}$ 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" oblivious local search results (essentially) in a $\frac{1}{k}$ 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" oblivious local search results (essentially) in a $\frac{1}{k}$ 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.

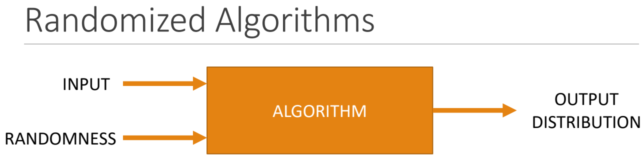
Some concluding comments on local search

- For the metric k-median problem, 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 Svensson (2013) obtained a $(1 + \sqrt{3} + \epsilon) \approx 2.732 + \epsilon$ LP-based 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. The latest improvement is a $2.633 + \epsilon$ approximation by Ahmadian et al (2017).
- An interesting (but probably difficult) open problem is to use non oblivious local search for the metric k-median, facility location, or k-means problems. These well motivated clustering problems play an important role in operations research, CS algorithm design and machine learning.

End of concluding remarks on local search

- Perhaps the main thing to mention now is 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.
- Linear Programming (LP) is often solved by some variant of the simplex method, which can be thought of as a local search algorithm, moving from one vertex of the LP polytope to an adjacent vertex.

Randomized algorithms



In any randomzed algorithm, the output of the algorithm can vary for a given input.

Beyond the usual concerns of (say time) efficiency and correctness (incluing approximation guarantees), we will need to consider randomness as a resource (i.e. how many random bits are needed) and probability guarantees on efficiency and correctness.

We will not worry about the source of randomness (although this is an interesting issue) but assume that we have a supply of random bits (or random numbers).

Randomzed algorithms everywhere

Randomized algorithms

- Unlike our focus on topics such as greedy algorithms, dynamic programming, local search, IP/LP rounding, randomization is not a meta algorithm or algorithmic paradigm.
- Rather, randomization an idea that can be applied in any algorithm.
- We shall mention its use in a variety of applications.

A few of the many applications of randomization

- In complexity theory, efficiently computable might be better defined as “computable in randomized polynomial time” rather than (deterministic) polynomial tme.
- Randomized approximation algorithms
- Number theory algorithms
- cryptography
- Deadlock avoidance in networks
- Hashing
- The probabilistic method for proving existence theorems
- Simulations

The Why of randomization continued

- There are computational settings (simulation, cryptography, sublinear time algorithms) where randomization is provably necessary.
- Our focus is on how to use randomization to either speed up computations and/or to improve an approximation and/or as a step towards a deterministic algorithm
- There are also problems where we do not know how to solve a problem efficiently without randomization.
- BUT as far as we know:

A fundamental complexity theory question

It could be that the class of decision/search/optimization problems solvable in randomized polynomial time is the same as those solvable in polynomial time.

In fact, this seems to be the current wisdom of some experts since if not the case then seemingly stranger things would result.

We will recall probabilistic concepts as needed.

Some problems that so far need randomization

There are a number of problems computable in randomized polynomial time not known to be in polynomial time although it is possible that some or all of these problems could turn out to be in deterministic polynomial time.

1. Multivariate polynomial zero testing and the symbolic determinant problem. As mentioned, some complexity theorists believe this problem can be solved in determinsitic polynomial time. We will begin our study of randomized algorithms with the problem of multivariate polynomial zero testing.
2. Given n, find a prime in $[2^n, 2^{n+1}]$. We know from number theory are sufficiently many primes in any such range that we can randomly guess an integer k in the range and then deterministically test if k is a prine.
3. Estimating the volume of a convex body given by a set of linear inequalitiies.
4. Solving a quadratic equation in $Z_p[x]$ for a large prime p.

Polynomial identity testing

As we may recall from calculus, a non-zero univariate polynomial $P(x)$ (say over the reals $\mathbb{R}$ for definiteness now but in general for any field F) of degree d has at most d zeros; that is, $P(x) = 0$ for at most d values of $x \in \mathbb{R}$. Consider any set S of size $|S|$. It follows that if we choose $r \in S$, uniformly at random, $Prob_{r \in_u S}[P(r) = 0] \leq \frac{d}{|S|}$.
The Schwartz Zipple Lemma extends this fact to multivariate polynomials.

Schwartz Zipple Lemma

Let $P \in \mathbf{F}[x_1, \ldots, x_m]$ be a non zero polynomial over a field $\mathbf{F}$ of total degree at most d. Let S be a finite subset of $\mathbf{F}$. Then
$Prob_{r_i \in_u S}[P(r_1, \ldots .r_m) = 0] \leq \frac{d}{|S|}$

In polynomial identity testing problem, we are implicitly given a multivariate polynomial and wish to determine if they are identical. One way we could be implicitly given polynomials is by an arithmetic circuit. A specific case of interest is the symbolic determinant problem.

The symbolic determinant problem

Consider an $n \times n$ matrix $A = (a_{i,j})$ whose entries are polynomials of total degree (at most) d in m variables, say with integer coeficients. The determinant $det(A) = \sum_{\pi \in S_n} (-1)^{sgn(\pi)} \prod_{i=1}^{n} a_{i,\pi(i)}$, is a polynomial of degree nd. The symbolic determinant problem is to determine whether $det(A) \not\equiv \mathbf{0}$, the zero polynomial. Let $L = \{A | det(A) \not\equiv \mathbf{0}\}$.

By the Schwartz Zipple Lemma, if $det(A)$ is not the zero polynomial, then $Prob_{r_i \in_u S}[P(r_1, \ldots . r_m) = 0] \leq \frac{nd}{|S|}$ Since the determinant with scalar entries (in S) can be computed in time proportional to matrix multiplication, we have a randomized polynomial time algorithm ALG for the symbolic determinant problem.

More precisely, if we say choose $|S| \geq 2dn$, then

- If $det(A) \equiv \mathbf{0}$, then ALG always rejects A (i.e. $A \notin L$))
- If $det(A)$ is not equivalent to $\mathbf{0}$, then ALG accepts (i.e. says $A \in L$) with probability $\geq 1/2$.

By repeating this random trial t times, the probability of making an error (i.e. saying $A \in L$ when $det(A) \equiv 0$) is at most $\frac{1}{2^t}$.

Randomization and complexity theory

- A 0-sided error or *ZPP* algorithm (standing for zero-sided probabilistic polynomial) is always correct and has polynomial *expected running time* (expectation over the randomness in the algorithm). This is what is often called a Las Vegas algorithm but I do not like this terminology.
- One type of Monte Carlo algorithm is a 1-sided error or *RP* algorithm that always always runs in polynomial time, is always correct for a NO instance but has a "small" probability of error for a Yes instance.

  The symbolic determinant problem (i.e. $\{A : det(A)$ is not the zero polynomial$\}$) is perhaps the best known problem in *RP*, not known to be in *P*. Note that $RP \subseteq NP$. Why?

Randomization and complexity theory

- A 0-sided error or *ZPP* algorithm (standing for zero-sided probabilistic polynomial) is always correct and has polynomial *expected running time* (expectation over the randomness in the algorithm). This is what is often called a Las Vegas algorithm but I do not like this terminology.
- One type of Monte Carlo algorithm is a 1-sided error or *RP* algorithm that always always runs in polynomial time, is always correct for a NO instance but has a "small" probability of error for a Yes instance.

  The symbolic determinant problem (i.e. $\{A : det(A)$ is not the zero polynomial$\}$) is perhaps the best known problem in *RP*, not known to be in *P*. Note that $RP \subseteq NP$. Why?
  We can think of *RP* as *NP* problems which have many random certificates.

Randomization and complexity theory continued

- In fact, it is sufficient to have the probability of error as large as $(1 - \frac{1}{n^k})$ for any k. When the error is $(1 - \frac{1}{t})$ by repeating the algorithm t times, the probability of error becomes $(1 - \frac{1}{t})^t \leq (\frac{1}{e})$.
- It is not difficult to show that $ZPP = RP \cap co - RP$.
- The other type of Monte Carlo algorithm is a 2-sided or BPP algorithm that always runs in polynomial time but has a probability at most $(\frac{1}{2} - \frac{1}{n^k})$ of being incorrect (for either YES or NO instances).
- The error probability can be reduced by running many trials and taking the majority outcome.
- It follows that $ZPP \subseteq RP \subseteq BPP$. However it is not known if $BPP \subseteq NP$.

NOTE In spite of the fact that we do not know how to deterministically solve some problems (such as the symbolic determinant problem) in polynomial time, some prominent complexity theorists believe (with some justification) that $BPP = P$.

The naive randomized algorithm for exact Max-k-Sat

We continue our discussion of randomized algorthms by considering the use of randomization for improving approximation algorithms. In this context, randomization can be (and is) combined with any type of algorithm.

Warning: For the following discussion of Max-Sat, we will follow the prevailing convention by stating approximation ratios as fractions $c < 1$.

- Consider the exact Max-k-Sat problem where we are given a CNF propositional formula in which every clause has exactly k literals. We consider the weighted case in which clauses have weights. The goal is to find a satisfying assignment that maximizes the size (or weight) of clauses that are satisfied.
- Since exact Max-k-Sat generalizes the exact k- SAT decision problem, it is clearly an NP hard problem for $k \geq 3$. It is interesting to note that while 2-SAT is polynomial time computable, Max-2-Sat is still NP hard.
- The naive randomized (online) algorithm for Max-k-Sat is to randomly set each variable to be *true* or *false* with equal probability.

Analysis of naive Max-k-Sat algorithm continued

- Since the expectation of a sum is the sum of the expectations, we just have to consider the probability that a clause is satisfied to determine the expected weight of a clause.
- Since each clause C_i has k variables, the probability that a random assignment of the literals in C_i will set the clause to be satisfied is exactly $\frac{2^k-1}{2^k}$. Hence $\mathbf{E}$ [weight of satisfied clauses] $= \frac{2^k-1}{2^k}\sum_i w_i$
- Of course, this probability only improves if some clauses have more than k literals. It is the small clauses that are the limiting factor in this analysis.
- This is not only an approxination ratio but moreover a "totality ratio" in that the algorithms expected value is a factor $\frac{2^k-1}{2^k}$ of the sum of all clause weights whether satisfied or not.
- We can hope that when measuring against an optimal solution (and not the sum of all clause weights), small clauses might not be as problematic as they are in the above analysis of the naive algorithm.

Derandomizing the naive algorithm

We can derandomize the naive algorithm by what is called the method of conditional expectations. Let $F[x_1, \ldots, x_n]$ be an exact k CNF formula over n propositional variables $\{x_i\}$. For notational simplicity let $\mathit{true} = 1$ and $\mathit{false} = 0$ and let $w(F)|\tau$ denote the weighted sum of satisfied clauses given truth assignment τ.

- Let x_j be any variable. We express $\mathbf{E}[w(F)|_{x_i \in_u \{0,1\}}]$ as $\mathbf{E}[w(F)|_{x_i \in_u \{0,1\}}|x_j = 1] \cdot (1/2) + \mathbf{E}[w(F)|_{x_i \in_u \{0,1\}}|x_j = 0] \cdot (1/2)$
- This implies that one of the choices for x_j will yield an expectation at least as large as the overall expectation.
- It is easy to determine how to set x_j since we can calculate the expectation clause by clause.
- We can continue to do this for each variable and thus obtain a deterministic solution whose weight is at least the overall expected value of the naive randomized algorithm.
- NOTE: The derandomization can be done so as to achieve an online algorithm. Here the (online) input items are the propostional variables. What input representation is needed/sufficient?

(Exact) Max-k-Sat

- For exact Max-2-Sat (resp. exact Max-3-Sat), the approximation (and totality) ratio is $\frac{3}{4}$ (resp. $\frac{7}{8}$).
- For $k \geq 3$, using PCPs (probabilistically checkable proofs), Hastad proves that it is NP-hard to improve upon the $\frac{2^k-1}{2^k}$ approximation ratio for Max-k-Sat.
- For Max-2-Sat, the $\frac{3}{4}$ ratio can be improved (as we will see) by the use of semi-definite programming (SDP).
- The analysis for exact Max-k-Sat clearly needed the fact that all clauses have at least k literals. What bound does the naive online randomized algorithm or its derandomztion obtain for (not exact) Max-2-Sat or arbitrary Max-Sat (when there can be unit clauses)?

Johnson's Max-Sat Algorithm

Johnson's [1974] algorithm

```
For all clauses $C_i$, $w'_i := w_i/(2^{|C_i|})$
Let $L$ be the set of clauses in formula $F$ and $X$ the set of variables
**For** $x \in X$ (or until $L$ empty)
   Let $P = \{C_i \in L$ such that $x$ occurs positively$\}$
   Let $N = \{C_j \in L$ such that $x$ occurs negatively$\}$
   **If** $\sum_{C_i \in P} w'_i \geq \sum_{C_j \in N} w'_j$
      $x := true; L := L \setminus P$
      **For all** $C_r \in N$,   $w'_r := 2w'_r$   **End For**
   **Else**
      $x := false; L := L \setminus N$
      **For all** $C_r \in P$,   $w'_r := 2w'_r$   **End For**
   **End If**
   Delete $x$ from $X$
**End For**
```

Aside: This reminds me of boosting (Freund and Shapire [1997])

Johnson's algorithm is the derandomized algorithm

- Twenty years after Johnson's algorithm, Yannakakis [1994] presented the naive algorithm and showed that Johnson's algorithm is the derandomized naive algorithm.
- Yannakakis also observed that for arbitrary Max-Sat, the approximation of Johnson's algorithm is at best $\frac{2}{3}$. For example, consider the 2-CNF $F = (x \vee \bar{y}) \wedge (\bar{x} \vee y) \wedge \bar{y}$ when variable x is first set to true.
- Chen, Friesen, Zheng [1999] showed that Johnson's algorithm achieves approximation ratio $\frac{2}{3}$ for arbitrary weighted Max-Sat.
- For arbitrary Max-Sat (resp. Max-2-Sat), the current best approximation ratio is .7968 (resp. .9401) using semi-definite programming and randomized rounding.
  Note: While existing combinatorial algorithms do not come close to these best known ratios, it is still interesting to understand simple and even online algorithms for Max-Sat.

Modifying Johnson's algorithm for Max-Sat

- It is interesting to note that the naive algorithm and Johnson's algorithm are *online* algorithms in the sense that input items (i.e. the propositional variables) come in some adversarial order $x_1, x_2, \ldots x_n$ and the algorithm makes an irreversible decision (i.e. set to true or false) for each propositional variable x_i without seeing $x_{i+1}, \ldots, x_n$.
- In proving the (2/3) approximation ratio for Johnson's Max-Sat algorithm,, Chen et al asked whether or not the ratio could be improved by using a random ordering of the propositional variables (i.e. the input items). This is an example of the *random order model* (ROM), a randomized variant of online algorithms.
- As an aside, we note that the ROM model was first introduced in what is known as the *secretary problem*. If a sequence of candidates are interviewed (and their value determined) and an irrevocable decision must be made to hire or not hire without seeing further candidates. How would you choose a candidate?

Online and ROM Input models for max-sat

To precisely model the Max-Sat problem as an online or ROM algorithm, we need to specify how each input item is specified. In increasing order of providing more information (and possibly better approximation ratios), the following input models can be considered:

- Model 0: Each propositional variable x is represented by the names of the positive and negative clauses in which it appears. This is sufficient for the naive randomized algorithm.
- Model 1: Additiionally, the lengthis of each clause C_i in which x appears positively, and for each C_j in which it appears negatively. This is sufficient for Johnsons's deterministic algorithm.
- Model 2: Additionally, for each C_i and C_j, a list of the other variables in the clause.
- Model 3: The variable x is represented by a complete specification of each clause it which it appears. This is the model that one would like to fully understand.

Improving on Johnson's algorithm

- The question asked by Chen et al was answered by Costello, Shapira and Tetali [2011] who showed that in the ROM model, Johnson's algorithm achieves approximation $(2/3 + \epsilon)$ for $\epsilon \approx .003653$
- Poloczek and Schnitger [same SODA 2011 conference] show that the approximation ratio for Johnson's algorithm in the ROM model is at most $2\sqrt{15}–7 \approx .746 < 3/4$, the ratio first obtained by Yannakakis' IP/LP approximation that we will soon present.
- Poloczek and Schnitger first consider a "canonical randomization" of Johnson's algorithm"; namely, the canonical randomization sets a variable $x_i = \mathit{true}$ with probability $\frac{w'_i(P}{w'_i(P)+w'_i(N)}$ where $w'_i(P)$ (resp. $w'_i(N)$) is the current combined weight of clauses in which x_i occurs positively (resp. negatively). Their substantial additional idea is to adjust the random setting so as to better account for the weight of unit clauses in which a variable occurs.

Another randomized online (weighted) max-sat $\frac{3}{4}$ approximation algorithm

The following algorithm is due independently to Buchbinder et al [2012] and van Zuelen [2011] as described in Poloczek et al [2016]. The idea of the algorithm is that in setting the variables, we want to balance the weight of clauses satisfied with that of the weight of clauses not yet unsatisfied.

Let S_i be the assignment to the first i variables and let SAT_i (resp. $UNSAT_i$) be the weight of satisfied clauses (resp., unsatsifed clauses) with respect to S_i. Let $B_i = \frac{1}{2}(SAT_i + W - UNSAT_i)$ where W is the total weight of all clauses.

Another randomized online (weighted) max-sat $\frac{3}{4}$ approximation algorithm

The following algorithm is due independently to Buchbinder et al [2012] and van Zuelen [2011] as described in Poloczek et al [2016]. The idea of the algorithm is that in setting the variables, we want to balance the weight of clauses satisfied with that of the weight of clauses not yet unsatisfied.

Let S_i be the assignment to the first i variables and let SAT_i (resp. $UNSAT_i$) be the weight of satisfied clauses (resp., unsatsifed clauses) with respect to S_i. Let $B_i = \frac{1}{2}(SAT_i + W - UNSAT_i)$ where W is the total weight of all clauses.

The algorithm's plan is to randomly set variable x_i so as to increase $\mathbb{E}[B_i - B_{i-1}]$.

Another randomized online (weighted) max-sat $\frac{3}{4}$ approximation algorithm

The following algorithm is due independently to Buchbinder et al [2012] and van Zuelen [2011] as described in Poloczek et al [2016]. The idea of the algorithm is that in setting the variables, we want to balance the weight of clauses satisfied with that of the weight of clauses not yet unsatisfied.

Let S_i be the assignment to the first i variables and let SAT_i (resp. $UNSAT_i$) be the weight of satisfied clauses (resp., unsatsifed clauses) with respect to S_i. Let $B_i = \frac{1}{2}(SAT_i + W - UNSAT_i)$ where W is the total weight of all clauses.

The algorithm's plan is to randomly set variable x_i so as to increase $\mathbb{E}[B_i - B_{i-1}]$.

To that end, let t_i (resp. f_i) be the value of $w(B_i) - w(B_{i-1})$ when x_i is set to true (resp. false). Fact: $t_i + f_i \geq 0$.

The randomized max-sat approximation algorithm continued

```
For $i = 1 \dots n$
    If $f_i \leq 0$, then set $x_i =$ true
    Else if $t_i \leq 0$,
      then set $x_i =$ false
    Else set $x_i$ true with probability $\frac{t_i}{t_i+f_i}$.
End For
```

Assuming input model 2, Poloczek [2011] shows that no deterministic online algorithm (even if the algorithm can order the input items as in a greedy algorithm) can achieve a 3/4 approximation. This provides a sense in which to claim that these randomized algorithms "cannot be derandomized" (in contrast to the naive algorithm).
The best deterministic priority algorithm in the third (most powerful) model remains an open problem as does the best randomized priority algorithm and the best ROM algorithm.

Revisiting the “cannot be derandomized comment”

- However, Buchbinder and Feldman [2016] provide a method that shows how to derandomize the Buchbinder et al and van Zeulen algorithm into a polynomial time “parallel” online deterministic algortihm (i.e., spawning $2n$ parallel online threads and taking the best solution amongst these threads).
- Poloczek et al [2017] de-randomize the Max-Sat algorithm using a 2-pass online algorithm. Roughly speaking, the first pass assigns probabilities and the second pass is able to derandomize the algorithm
- We will next consider another approach for obtaining the same $\frac{3}{4}$ approximation. This is not as efficient as the randomized online algorithm and it de-randomization as a 2-pass algorithm but it was the first algorithm to provide a $\frac{3}{4}$ approximation and it illustates the use of randomized rounding.

Yannakakis' IP/LP *randomized rounding* algorithm for Max-Sat

- We will formulate the weighted Max-Sat problem as a $\{0,1\}$ IP.
- Relaxing the variables to be in $[0,1]$, we will treat some of these variables as probabilities and then round these variables to 1 with that probability.
- Let F be a CNF formula with n variables $\{x_i\}$ and m clauses $\{C_j\}$. The Max-Sat formulation is :
  maximize $\sum_j w_j z_j$
  subject to $\sum_{\{x_i \text{ is in } C_j\}} y_i + \sum_{\{\bar{x}_i \text{ is in } C_j\}} (1 - y_i) \geq z_j$
  $y_i \in \{0,1\}; z_j \in \{0,1\}$
- The y_i variables correspond to the propositional variables and the z_j correspond to clauses.
- The relaxation to an LP is $y_i \geq 0$; $z_j \in [0,1]$. Note that here we cannot simply say $z_j \geq 0$.

Randomized rounding of the y_i variables

- Let $\{y_i^*\}, \{z_j^*\}$ be the optimal LP solution,
- Set $\tilde{y}_i = 1$ with probability y_i^*.

Theorem

Let C_j be a clause with k literals and let $b_k = 1 - (1 - \frac{1}{k})^k$. Then $Prob[C_j$ is satisifed $]$ is at least $b_k z_j^*$.

- The theorem shows that the contribution of the j^{th} clause C_j to the expected value of the rounded solution is at least $b_k w_j$.
- Note that b_k converges to (and is always greater than) $1 - \frac{1}{e}$ as k increases. It follows that the expected value of the rounded solution is at least $(1 - \frac{1}{e})$ LP-OPT $\approx .632$ LP-OPT.
- Taking the max of this IP/LP and the naive randomized algorithm results in a $\frac{3}{4}$ approximation algorithm that can be derandomized. (The derandomized algorithm will still be solving LPs.)