CSC2420: Algorithm Design, Analysis and Theory
Fall 2023
An introductory (i.e. foundational) level graduate course.

Allan Borodin

December 5, 2023

Week 12

Announcements:

- Assignment 3 is due Friday, December 8 at 11 AM No extensions.

Todays agenda
We will discuss a collection of miscellaneous results:

- Recalling "basic paradigms".
- An example of a "more clever brute force search" to derive a divide and conquer algorithm for Max-3-Sat.
- The makespan problem for unrelated machines: An example of a more involved "rounding" of an IP/LP.
- Another paradigm: weighted majority
- More examples of vector programs
- The densest subgraph problem. Charikar's reverse greedy algorithm.

Reviewing some basic algorithmic paradigms

We begin with some "conceptually simple" search/optimization algorithms.

The conceptually simplest "combinatorial" algorithms

Given an optimization problem, it seems to me that the conceptually simplest approaches are:

- brute force search
- divide and conquer
- greedy
- local search
- dynamic programming

Comment

- We usually dismiss brute force as it really isn't much of an algorithm approach but might work for small enough problems.
- Moreover, sometimes we can combine some aspect of brute force search with another approach as we will soon see.

More basic paradigms

To these basic paradigms, we should add IP/LP rounding as that also is often conceptually simple. Of course, selecting the most appropriate IP and knowing how best to round can be quite non trivial. But often, we can at least easily come up with a reasonable IP/LP formulation and some naive way to "round".

And to the extent that we have a reasonable IP/LP formulation of a problem, we can also think the primal dual method as a basic paradigm, using the dual to guide setting of primal variables.

Recalling the DLS given by Marijn Heule on the use of SAT solvers, we can think of formulating some search and optimization problems as SAT problems. And again, like IP/LP as a basic paradigm, the choice of the most appropriate SAT formulation can be quite (even very) non-trivial.

We didn't discuss divide and conquer or dynmaic programming but they are definitely basic paradigms. And as mentioned we can combine brute force with other paradigms. In the next few slides we will see an example of combining brute force with divide and conquer.

A relatively simple deterministic algorithm for 3-SAT

Consider a 3-CNF formula with n variables and m clauses. While there are any clauses with 3 literals (for example, x, y, z), branch on each of the 7 possible settings for (x, y, z) that can make the clause true.

In this way, we are creating a 7-ary tree where each node specifies a truth value setting on three distinct variables. We discontinue a branch whenever we falsify a clause.

On any branch the current truth value setting can satisfy some clauses which can then be eliminated. In other clauses, one or two variables will be eliminated.

We continue to do this until there are no clauses remaining that have three literals. If any branch satisfies all clauses or if we are left with only consistent unit clauses, the given formula is saisfiable. If we are left with two contradictory unit clauses, we are also done and the given formula is unsatisfiable.

Finishing the simple deterministic algorithm for 3-SAT

Otherwise at each branch, we have a 2-SAT formula for which we can determine satisfiability.

The depth of the truth value tree is at most $n/3$ since we were eliminating 3 distinct variables at each level.

Hence the time complexity is $O(7^{n/3}poly(m)) = 2^{log_2 7 n/3}poly(m) \approx 2^{\frac{2.81}{3}n}poly(m)) \approx \tilde{O}(1.913)^n$ which improves the exponent obtained by naively trying all 2^n truth values.

We note again that the encoded length of the given formula is $O(m \log n))$ so we can ignore the $poly(m)$ factor with regard to understanding what is the best possible possible exponent (assuming the time complexity is exponential in n).

A modification of the previous "7-way branching" algorithm

Instead of brancing on all 7 possible satisfying truth assignments for a given clause, here is a modification which we can call a "3-way branching" algorithm. Here is am using lecture slides by William Gasarch.

Again consider a clause with 3 literals, say x, y, z. Then either the formula is true setting $x = \mathit{true}$; or $x = \mathit{false}$; $y = \mathit{true}$; or $x = \mathit{false}, y = \mathit{false}, z = \mathit{true}$.

Viewed as a recursive algorithm, we get the recurrence $T(n) = T(n-1) + T(n-2) + T((n-3)$ instead of the recurrence $T(n) = 7T(n-3)$.

Aiming for $T(n)$ roughly equal to α^n for some α, we would have $\alpha^3 = \alpha^2 + \alpha + 1$ yielding $T(n) \approx O((1.84)^n$.

And some tweaking of this 3-way branching algorithm leads to an algorithm with time compleixty $O((1.618)^n)$.

Makespan for the unrelated and restricted machine models: a more sophisticated rounding

In the IP/LP rounding for the weighted vertex problem, the rounding was "(input) independent rounding" and oblivious to the input.

- We now return to the makespan problem (our first problem week 1) ibut now with respect to the unrelated machines model and the special case of the restricted machine model.
- Recall the unrelated machines model where a job j is represented by a tuple $(p_{j,1}, \ldots, p_{j,m})$ where $p_{j,i}$ is the time that job j uses if scheduled on machine i.
- An important scheduling result is the Lenstra, Shmoys, Tardos (LST) [1990] IP/LP 2-approximation algorithm for the makespan problem in the unrelated machine model (when m is part of the input). They also obtain a PTAS for fixed m.

The natural IP and the LP relaxation

The IP/LP for unrelated machines makespan

- Minimize T
- Subject to
  1. $\sum_i x_{j,i} = 1$ for every job j % schedule every job
  2. $\sum_j x_{j,i} p_{j,i} \leq T$ for every machine i % do not exceed makespan
  3. $x_{j,i} \in \{0, 1\}$ % $x_{j,i} = 1$ iff job j scheduled on machine i

- The immdiate LP relaxation is to just have $x_{j,i} \geq 0$
- Even for identical machines (where $p_{j,i} = p_j$ for all i), the integrality gap IG is unbounded since the input could be just one large job with say size T leading to an LP-OPT of T/m and IP-OPT $=$ OPT $= T$ so that the IG $= m$.

Adapting the natural IP

- We use binary search with an LP solver, to find the best T for the LP relaxation of the problem.
  Namely, given a candidate T, we remove all x_{ji} such that $p_{j,i} > T$ and obtain a "search problem" (i.e. constant or no objective function) for finding $x_{j,i}$ satisfying the IP constraints.
- Once we have found the optimal T for the search problem, the LST algorithm then shows how to use a non-independent rounding to obtain an integral solution yielding a 2-approximation.
- Note: We use the term "rounding" in a very general sense to mean any efficient way to convert the LP solution into an integral solution.

Sketch of LST rounding for makespan problem

- Using slack form, LP theory can be used to show that if $\mathcal{L}$ is a feasible LP with $m + n$ constraints (not counting the non-negativity constraints for the variables) then $\mathcal{L}$ has an optimal basic solution such that at most $n + m$ of the variables are non-zero.
- It follows that there are at most m of the n jobs that have fractional solutions (i.e. are not assigned to a single machine).
- Jobs assigned to a single machine do not need to be rounded; i.e. if $x_{j,i} = 1$ then schedule job j on machine i.
- Construct a bipartite graph between the $y \leq m$ fractionally assigned jobs and the m machines.

The rounding continued

- The goal is then to construct a matching of size y; that, is, the matching dictates how to schedule these fractionally assigned jobs. So it "only" remains to show that this bipartite graph has a matching of size y. Note, of course, this is what makes the "rounding" non-independent .
- The existence of this matching requires more LP theory whereby it can be shown (LST credit Dantzig [1963]) that the connected components of the bipartite graph are either trees or trees with one added edge (and therefore causing a unique cycle).
- The resulting schedule then has makespan at most $2T$ since each fractional job has $p_{j,i} \leq T$ and the LP has guaranteed a makespan at most T before assigning the fractional jobs.

New topic: the weighted majority algorithm

I am following a survey type paper by Arora, Hazan and Kale [2008]. To quote from their paper: "We feel that this meta-algorithm and its analysis should be viewed as a basic tool taught to all algorithms students together with divide-and-conquer, dynamic programming, random sampling, and the like".

- The weighted majority algorithm and generalizations
  The "classical" WMA pertains to the following situation:
  Suppose we have say n expert weathermen (or maybe "expert" stock market forecasters) and at every time t, they give a binary prediction (rain or no rain, Raptors win or lose, dow jones up or down, Canadian dollar goes up or down.
- Now some or all of these experts may actually be getting their opinions from the same sources (or each other) and hence these predictions can be highly correlated.
- Without any knowledge of the subject matter (and why should I be any different from the "experts") I want to try to make predictions that will be nearly as good (over time t) as the BEST expert.

The weighted majority algorithm

The WM algorithm

Set $w_i(0) = 1$ for all i

For $t = 0...$

Our $(t+1)^{st}$ predication is

0: if $\sum_{\{i:\text{ expert } i \text{ predicts } 0\}} w_i(t) \geq (1/2)\sum_i w_i(t)$

1: if $\sum_{\{i:\text{ expert } i \text{ predicts } 1\}} w_i(t) \geq (1/2)\sum_i w_i(t)$; arbitrary o.w.

% We vote with weighted majority; arbitrary if tie

For $i = 1..n$

If expert i made a mistake on $(t+1)^{st}$ prediction

then $w_i(t+1) = (1-\epsilon)w_i(t)$;

else $w_i(t+1) = w_i(t)$

End If

End For

End For

How good is our uninformed MW prediction?

Theorem : Perfomance of WM

Theorem: Let $m_i(t)$ be the number of mistakes of expert i after the first t forecasts, and let $M(t)$ be the number of our mistakes. Then for any expert i (including the best expert) $M(t) \leq \frac{2 \ln n}{\epsilon} + 2(1+\epsilon)m_i(t)$.

- That is, we are "essentially" within a multiplicative factor of 2 plus an additive term of the best expert (without knowing anything).
- Using randomization, the factor of 2 can be removed. That is, instead of taking the weighted majority opinion, in each iteration t, choose the prediction of the i^{th} expert with probability $w_i(t)/\sum_i w_i(t)$. This is the "natural randomization" that we previously saw in maximizaing non monotone submodular function.

Theorem: Performance of Randomized WM

For any expert i, $\mathbf{E}[M(t)] \leq \frac{\ln n}{\epsilon} + (1+\epsilon)m_i(t)$

Proof of deterministic WM

Let's assume that $\epsilon \leq 1/2$. It follows that

$$-\epsilon - \epsilon^2 \leq \ln(1-\epsilon) < -\epsilon$$

Let $w_{i,t}$ be the weight of the i^{th} expert at time t and let $m_i(t)$ be the number of mistakes made by expert i. . Consider the potential function $\Phi(t) = \sum_i w_{i,t}$. Clearly

$$\Phi(t) \geq w_{i,t} = (1-\epsilon)^{m_i(t)}$$

We now need an upper bound on $\Phi(t)$. Since each time the WM algoriithm makes a mistake, at least half of the algorithms make a mistake so that $\Phi(t) \leq (1-\epsilon/2)\Phi(t-1)$. Starting with $\Phi(0) = n$, by induction $\Phi(t) \leq n \cdot (1-\epsilon/2)^{M(t)}$
Putting the two inequlaities together and taking logarithms

$$\ln(1-\epsilon)m_i(t) \leq \ln n + M(t)\ln(1-\epsilon/2)$$

The argument is completed by rearranging, using the above facts concerning $\ln(1-\epsilon)$ and then dividing by $\epsilon/2$.

What is the meaning of the randomized impovement?

- In many applications of randomization we can argue that randomization is (provably) necessary and in other applications, it may not be provable so far but current experience argues that the best algorithm in theory and practice is randomized.
- For some algorithms (and especially online algorithms) analyzed in terms of worst case performance, there is some debate on what randomization is actually accomplishing.
- In a [1996] article Blum states that "Intuitively, the advantage of the randomized approach is that it dilutes the worst case". He continues to explain that in the determinstic algorithm, slightly more than half of the total weight could have predicted incorrectly, causing the algorithm to make a mistake and yet only reducing the total weight by $1/4$ (when $\epsilon = 1/2$). But in the randomized version, there is still a .5 probability that the algorithm will predict correctly. Convincing?

An opposing viewpoint

- In the blog LessWrong this view is strongly rejected. Here the writer comments: "We should be especially suspicious that the randomized algorithm guesses with probability proportional to the expert weight assigned. This seems strongly reminiscent of betting with 70% probability on blue, when the environment is a random mix of 70% blue and 30% red cards. We know the best bet and yet we only sometimes make this best bet, at other times betting on a condition we believe to be less probable. Yet we thereby prove a smaller upper bound on the expected error. Is there an algebraic error in the second proof? Are we extracting useful work from a noise source? Is our knowledge harming us so much that we can do better through ignorance?" The writer asks: "So what's the *gotcha* ... the improved upper bound proven for the randomized algorithm did not come from the randomized algorithm making systematically better predictions - doing superior cognitive work, being more intelligent - but because we arbitrarily declared that an intelligent adversary could read our mind in one case but not in the other."

Further defense of the randomized approach

- Blum's article expresses a second benefit of the randomized approach: "Therefore the algorithm can be naturally applied when predictions are 'strategies' or other sorts of things that cannot easily be combined together. Moreover, if the 'experts' are programs to be run or functions to be evaluated, then this view speeds up prediction since only one expert needs to be examined in order to produce the algorithm's prediction ...."
- I agree with Blum but I can see understand the dissenting argument. To the extent that our algorithms are randomized, an important consideration (that we mainly ignored) is whether the result is just a result is expectation or is it a result where we obtain the expected value with high probability. I think the dissenting argument becomes irrelvant when we can guarantee high probability.
- A basic randomized paradigm for online and greedy algorithm is *classify and randomly select* which only yields results in expectation.
- Of course, for all algorithmic approaches we have to ask the question: does it work well "in practce"?

Generalizing: The Multiplicative Weights algorithm

The Weighted Majority algorithm can be generalized to the multiplicative weights algorithm. If the i^{th} expert or decision is chosen on day t, it incurs a real valued cost/profit $m_i(t) \in [-1, 1]$. The algorithm then updates $w_i(t+1) = (1 - \epsilon m_i(t))w_i(t)$. Let $\epsilon \leq 1/2$ and $\mathbf{\Phi}(t) = \sum_i w_i(t)$. On day t, we randomly select expert i with probability $w_i(t)/\Phi(t)$.

Performance of The MW algorithm

The expected cost of the MW algorithm after T rounds is $\sum_{t=1}^{T} \mathbf{m}(t) \cdot \mathbf{p}(t) \leq \frac{\ln n}{\epsilon} + \sum_{t=1}^{T} m_i(t) + \epsilon \sum_{t=1}^{T} |m_i(t)|$

Reinterpreting in terms of gains instead of losses

We can have a vector $\mathbf{m}(t)$ of gains instead of losses and then use the "cost vector" $-\mathbf{m}(t)$ in the MW algorithm resulting in:

Performance of The MW algorithm for gains

$\sum_{t=1}^{T} \mathbf{m}(t) \cdot \mathbf{p}(t) \geq -\frac{\ln n}{\epsilon} + \sum_{t=1}^{T} m_i(t) - \epsilon \sum_{t=1}^{T} |m_i(t)|$

By taking convex combinations, an immediate corollary is

Performance wrt. a fixed distribution p

$\sum_{t=1}^{T} \mathbf{m}(t) \cdot \mathbf{p}(t) \geq -\frac{\ln n}{\epsilon} + \sum_{t=1}^{T} \mathbf{m}(t) - \epsilon |\mathbf{m}(t)|)\mathbf{p}$

An application to learning a linear binary classifier

Instead of the online application of following expert advice, let us now think of "time" as rounds in an iterative procedure. In particular, we would like to compute a linear binary classifier (when it exists).

- We are trying to classsify objects characterized by n features; that is by points $\mathbf{a}$ in $\Re^n$. We are given m labelled examples $(\mathbf{a}_1, \ell_1), \ldots, (\mathbf{a}_m, \ell_m)$ where $\ell_j \in \{-1, +1\}$
- We are going to assume that these examples can be "well classified" by a linear classifier in the sense that there exists a non negative vector $\mathbf{x}^* \in \Re^n$ (with $x_i \geq 0$) such that $sign(\mathbf{a}_j \cdot \mathbf{x}^*) = \ell_j$ for all j.
- This is equivalent to saying $\ell_j \mathbf{a}_j \cdot \mathbf{x}^* \geq 0$ and furthermore (to explain the "well") we will say that $\ell_j \mathbf{a}_j \cdot \mathbf{x}^* \geq \delta$ for some $\delta > 0$.
- The goal now is to learn some linear classifer; ie a non negative $\mathbf{x} \in \Re^n$ such that $\ell_j \mathbf{a}_j \cdot \mathbf{x}^* \geq 0$. Without loss of generality, we can assume that $\sum_i x_i = 1$.
- Letting $\mathbf{b}_j = \ell_j \mathbf{a}_j$, this can now be veiwed as a reasonably general LP (search) problem.

Littlestone's Winnow algorithm for learning a linear classifier

- Litlestone [1987] used the multiplicative weights approach to solve this linear classification problem.
- Let $\rho = \max_j ||\mathbf{b}_j||_\infty$ and let $\epsilon = \delta/(2\rho)$
- The idea is to run the MW algorthm with the decisions given by the n features and gains specified by the m examples. The gain for feature i with respect to the j^{th} example is defined as $(\mathbf{b}_j)_i/\rho$ which is in [-1,1]. The $\mathbf{x}$ we are seeking is the distribution $\mathbf{p}$ in MW.

The Winnow algorithm

Initialize $\mathbf{p}$
While there are points not yet satisfied
  Let $\mathbf{b}_j \cdot \mathbf{p} < 0$ % a constraint not satisfied
  Use MW to upate $\mathbf{p}$
End While

Bound on number of iterations

The Winnow algorithm will terminate in at most $\lceil 4\rho^2 \ln n/\delta^2 \rceil$ iterations.

Some additional remarks on Multiplicative Weights

The survey by Arora, Hazan and Kale [2012] discusses other modifications of the MW paradigm and numerous applications. In terms of applications, they sketch results for

- Aporoximately solving (in the sense of property testing) the decision problem for an LP; there that is given linear constraints expressed by $A\mathbf{x} \geq \mathbf{b}$, the decision problem is to see if such a non-negative $\mathbf{x}$ exists (or more generally, if $\mathbf{x}$ is in some given convex set). The algorithm either returns a $\mathbf{x}$: $\mathbf{A_i x} \geq \mathbf{b_i} - \delta$ for all i and some additive approximation δ or says that the given LP was infeasible.
- Solving zero sum games approximately.
- The AdaBoost algorithm of Shapire and Freund
- Some other specific applications including a class of online algorithms.

I also suggest lecture slides by Uri Zweig on the MW algorithm and its applications. I will post his slides.

More results via vector programming

In week 8, we briefly discussed the quadradic program and its vector program relaxation for Max-2-Sat. Rounding vector programs provide the best known algorithms for a number of other problems. We will first recall how Max-2-at is solved by this approach and then survey some other results following the same approach.

In particular, we have the following problems:

- We review the Max-2-Sat formulation
- Max-cut
- Cardinality constrained monotone set function maximization

The quadratic program for Max-2-Sat

The following discussion is taken from the Vazirani *Approximation Algorithms* textbook.

- We introduce {-1,1} variables y_i corresponding to the propositional variables. We also introduce a homogenizing variable y_0 which will correspond to a constant truth value. That is, when $y_i = y_0$, the intended meaning is that x_i is set *true* and *false* otherwise.
- We want to express the $\{-1, 1\}$ truth value $val(C)$ of each clause C in terms of these $\{-1, 1\}$ variables.
  1. $val(x_i) = (1 + y_i y_0)/2$
     $val(\bar{x}_i) = (1 - y_i y_0)/2$
  2. If $C = (x_i \vee x_j)$, then $val(C) = 1 - val(\bar{x}_i \wedge \bar{x}_j) = 1 - (\frac{1-y_i y_0}{2})(\frac{1-y_j y_0}{2}) = (3 + y_i y_0 + y_j y_0 - y_i y_j)/4 = \frac{1+y_0 y_i}{4} + \frac{1+y_0 y_j}{4} + \frac{1-y_i y_j}{4}$
  3. If $C = (\bar{x}_i \vee x_j)$ then $val(C) = (3 - y_i y_0 + y_j y_0 + y_i y_j)/4$
  4. If $C = (\bar{x}_i \vee \bar{x}_j)$ then $val(C) = (3 - y_i y_0 - y_j y_0 - y_i y_j)/4$

The quadratic program for Max-2-Sat continued

- The Max-2-Sat problem is then to maximize $\sum w_k val(C_k)$ subject to $(y_i)^2 = 1$ for all i
- By collecting terms of the form $(1 + y_i y_j)$ and $(1 - y_i y_j)$ the max-2-sat objective can be represented as the strict quadratic objective: $\max \sum_{0 \leq i < j \leq n} a_{ij}(1 + y_i y_j) + \sum b_{ij}(1 - y_i y_j)$ for some appropriate a_{ij}, b_{ij}.
- Like an IP this integer quadratic program cannot be solved efficiently.

The vector program relaxation for Max-2-Sat

- We now relax the quadratic program to a vector program where each y_i is now a unit length vector $\mathbf{v}_i$ in $\Re^{n+1}$ and scalar multiplication is replaced by vector dot product. This vector program can be (approximately) efficiently solved (i.e. in polynomial time).
- The randomized rounding (from $\mathbf{v}_i^*$ to y_i) proceeds by choosing a random hyperplane in $\Re^{n+1}$ and then setting $y_i = 1$ iff $\mathbf{v}_i^*$ is on the same side of the hyperplane as $\mathbf{v}_0^*$. That is, if $\mathbf{r}$ is a uniformly random vector in $\Re^{n+1}$, then set $y_i = 1$ iff $\mathbf{r} \cdot \mathbf{v}_i^* \geq 0$.
- The rounded solution then has expected value $2\sum a_{ij} Prob[y_i = y_j] + \sum b_{ij} Prob[y_i \neq y_j]$; $Prob[y_i \neq y_j] = \frac{\theta_{ij}}{\pi}$ where θ_{ij} is the angle between $\mathbf{v}_i^*$ and $\mathbf{v}_j^*$.

The approximation ratio (in expectation) of the rounded solution

Let $\alpha = \frac{2}{\pi} \min_{\{0 \leq \theta \leq \pi\}} \frac{\theta}{(1-cos(\theta)} \approx .87856$ and let OPT_{VP} be the value obtained by an optimal vector program solution.
Then **E**[rounded solution] $\geq \alpha \cdot (OPT_{VP})$.

The Goemans and Williamson program algorithm for max-cut

This is very similar to Max–2-sat.

- We introduce {-1,1} variables y_i corresponding to the vertex variables x_i. We also need a homogenizing variable y_0; the intended meaning is that vertex $v_i \in S$ and iff $y_i = y_0$.
- The max-cut problem can then be represented by the following (strict) quadratic programming problem:

  Maximize $\frac{1}{2}\sum_{1\le i<j\le n} w_{ij}(1 - y_i y_j)$ subject to
  $y_i^2 = 1$ for $0 \le i \le n$
- This is relaxed to a vector program by introducing vectors on the unit sphere in $\mathbf{v}_i \in \mathbb{R}^{n+1}$ where now the scalar multiplication becomes the vector inner product.

The same rounding of the vector program

- The randomized rounding (from $\mathbf{v}_i^*$ to y_i) proceeds by choosing a random hyperplane in $\Re^{n+1}$ and then setting $y_i = 1$ iff $\mathbf{v}_i^*$ is on the same side of the hyperplane as $\mathbf{v}_0^*$. That is, if $\mathbf{r}$ is a uniformly random vector in $\Re^{n+1}$, then set $y_i = 1$ iff $\mathbf{r} \cdot \mathbf{v}_i^* \geq 0$.
- The rounded solution then has expected value $\sum_{1 \leq i < j \leq n} w_{ij} \mathbf{Pr}[\mathbf{v_i} \text{ and } \mathbf{v_j} \text{ are separated}] = \sum_{1 \leq i < j \leq n} w_{ij} \frac{\theta_{ij}}{\pi}$ where θ_{ij} is the angle between $\mathbf{v}_i^*$ and $\mathbf{v}_j^*$.

The approximation ratio (in expectation) of the rounded solution

Let $\alpha = \frac{2}{\pi} \min_{\{0 \leq \theta \leq \pi\}} \frac{\theta}{(1-cos(\theta)} \approx .87856$ and let OPT_{VP} be the value obtained by an optimal vector program solution.
Then $\mathbf{E}$[rounded solution] $\geq \alpha \cdot (OPT_{VP})$.

Set function maximization

Densest subgraph, max cut, max-di-cut and max-sat are problems where the goal is to maximize a non-monotone set function and hence (given that they are non-monotone) make sense in their *unconstrained* version.

Of course, similar to monotone set function maximization problems (such as max coverage), these problems also have natural constrained versions, the most studied version being a cardinality constraint.

More generally, there are other specific and arbitrary matroid constraints, other independence constraints, and knapsack constraints.

Cardinality constrained set function maximization

Max-k-densest subgraph, max-k-cut, max-k-di-cut, max-k-uncut and max-k-vertex-coverage are the natural cardinality constrained versions of well studied graph maximization problems. They all are of the following form:

Given an edge weighted graph $G = (V, E, w)$ with non negative edge weightes $w : E \to \mathbb{R}$, find a subset $S \subseteq V$ with $|S| = k$ so as to maximize some set function $f(S)$. (Of course, In the unweighted versions, $w(e) = 1$ for all $e \in E$.)

For example, the objective in max-k-uncut is to find S so as to maximize the edge weights of the subgraphs induced by S and $V \setminus S$. That is, in a social network, divide the graph into two "communities".

NOTE: Max-k-sat is *not* the cardinality constrained version of max-sat in the same sense as the above problems. Although not studied (as far as I know), the analogous problem would be to find a set of propositional variables of cardinality k so as to maximize the weights of the satisfied clauses.

The SDP/vector program algorithms for the cardinality constrained problems

In what follows, I will briefly sketch some of the SDP based analysis in Feige and Langberg [2001]. This paper was proceeded and followed by a substantial number of important papers including the seminal Goemans and Williamson [1995] SDP approximation algorithm for max-cut.

(See , for example, Feige and Goemans [1995] and Frieze and Jerrun [1997] for proceeding work and Halperin and Zwick [2002], Han et al [2002] and Jäger and Srivastav for some improved and unifying results.)

There are also important LP based results such as the work by Ageev and Sviridenko [1999, 2004] that introduced *papage rounding*.
Many papers focus on the *bisection* versions where $k = n/2$ and also $k = \sigma n$ for some $0 < \sigma < 1$ for which much better approximations atre known relative to results for a general cardinality k constraint.

The Goemans and Williamson program algorithm for max-cut

As stated in Lecture 6, vector programs can be solved to arbitrary precision within polynomial time.

- We introduce {-1,1} variables y_i corresponding to the vertex variables x_i. We also need a homogenizing variable y_0; the intended meaning is that vertex $v_i \in S$ and iff $y_i = y_0$.
- The max-cut problem can then be represented by tbhe following (strict) quadratic programming problem:

  Maximize $\frac{1}{2}\sum_{1\leq i<j\leq n} w_{ij}(1 - y_i y_j)$ subject to
  $y_i^2 = 1$ for $0 \leq i \leq n$
- This is relaxed to a vector program by introducing vectors on the unit sphere in $\mathbf{v}_i \in \mathbb{R}^{n+1}$ where now the scalar multiplication becomes the vector inner product.

The rounding of the vector program

- The randomized rounding (from $\mathbf{v}_i^*$ to y_i) proceeds by choosing a random hyperplane in $\Re^{n+1}$ and then setting $y_i = 1$ iff $\mathbf{v}_i^*$ is on the same side of the hyperplane as $\mathbf{v}_0^*$. That is, if $\mathbf{r}$ is a uniformly random vector in $\Re^{n+1}$, then set $y_i = 1$ iff $\mathbf{r} \cdot \mathbf{v}_i^* \geq 0$.
- The rounded solution then has expected value $\sum_{1 \leq i < j \leq n} w_{ij} \mathbf{Pr}[\mathbf{v_i} \text{ and } \mathbf{v_j} \text{ are separated}] = \sum_{1 \leq i < j \leq n} w_{ij} \frac{\theta_{ij}}{\pi}$ where θ_{ij} is the angle between $\mathbf{v}_i^*$ and $\mathbf{v}_j^*$.

The approximation ratio (in expectation) of the rounded solution

Let $\alpha = \frac{2}{\pi} \min_{\{0 \leq \theta \leq \pi\}} \frac{\theta}{(1-cos(\theta)} \approx .87856$ and let OPT_{VP} be the value obtained by an optimal vector program solution.
Then $\mathbf{E}$[rounded solution] $\geq \alpha \cdot (OPT_{VP})$.

Extending the vector program formulation for the cardinality constraint

The basic idea is to add an additional constraint:

$$\sum_{i=1}^{n} \mathbf{v}_i \mathbf{v}_0 = 2k - n$$

It turns out that it is sometimes important to define an improved relaxation by using instead (for all $j \in \{0, \ldots, n\}$) the "caraidnality constraints": $\sum_{i=1}^{n} \mathbf{v}_i \mathbf{v}_j = \mathbf{v_j} \mathbf{v}_0 (2k - n)$
For vectors $\mathbf{v}_j$ in the unit sphere, these constraints are equivalent to the constraints $\sum_{i=1}^{n} \mathbf{v}_i = \mathbf{v}_0 (2k - n)$

It also turns out that sometimes problems also use the following "triangle inequality constraints": $\mathbf{v}_i \mathbf{v}_j + \mathbf{v}_j \mathbf{v}_k + \mathbf{v}_k \mathbf{v}_i \geq -1$
and $\mathbf{v}_i \mathbf{v}_j + \mathbf{v}_j \mathbf{v}_k + \mathbf{v}_k \mathbf{v}_i \geq -1$ for all $i, j, k \in \{0, 1, \ldots, n\}$

What else has to be done?

Skipping some important considerations (not used for the max-k-coverage and max-k-densest subgraph problems) regarding how to merge this SDP/vector program relaxation with the LP max-cut formulation by Ageev and Svridenko, there is one very essential consideration that we have ignored thus far.

What else has to be done?

Skipping some important considerations (not used for the max-k-coverage and max-k-densest subgraph problems) regarding how to merge this SDP/vector program relaxation with the LP max-cut formulation by Ageev and Svridenko, there is one very essential consideration that we have ignored thus far.

The random hyperplane rounding insures the required probability that the round vectors will be separated. **BUT** this rounding does *not* enforce the desired k cardinality constraint.

What else has to be done?

Skipping some important considerations (not used for the max-k-coverage and max-k-densest subgraph problems) regarding how to merge this SDP/vector program relaxation with the LP max-cut formulation by Ageev and Svridenko, there is one very essential consideration that we have ignored thus far.

The random hyperplane rounding insures the required probability that the round vectors will be separated. **BUT** this rounding does *not* enforce the desired k cardinality constraint.

This is rectified by Feige and Langberg by modifying the s SDP results so as to penalize the resulting sets S by a penalty depending on the deviation from the desired cardinality k.

What else has to be done?

Skipping some important considerations (not used for the max-k-coverage and max-k-densest subgraph problems) regarding how to merge this SDP/vector program relaxation with the LP max-cut formulation by Ageev and Svridenko, there is one very essential consideration that we have ignored thus far.

The random hyperplane rounding insures the required probability that the round vectors will be separated. **BUT** this rounding does *not* enforce the desired k cardinality constraint.

This is rectified by Feige and Langberg by modifying the s SDP results so as to penalize the resulting sets S by a penalty depending on the deviation from the desired cardinality k.

Namely, they run the SDP sufficiently many times and output the set that maximizes $Z = \frac{w(S)}{OPT_{SDP}} + \theta_1 \frac{n-|S|}{n-k} + \theta_2 \frac{|S|(2k-|S|)}{n^2}$ where $w(S)$ is the SDP rounded output, OPT_{SDP} is the optimum relaxed value, and the θ are appropriately optimized scalars.

The formulation for other set function maximization problems

The formulation and idea of the relaxation follows the same idea by mainly changing the objective function. (Recall the objective for max-2-sat.)

- For the max-k-densest subgraph problem, the objective (wrt. $y_i \in \{-1, 1\}$) is to maximize $\sum_{e_{ij} \in E} w_{ij} \frac{1+y_i y_0+y_j y_0+y_i y_j}{4}$
- The max-k-vertex-coverage problem a special case of the max coverage where each element (i.e. an edge) occurs in eactly two of the sets (i.e. vertices).

  The objective is to maximize $\sum_{e_{ij} \in E} w_{ij} \frac{3+y_i y_0+y_j y_0-y_i y_j}{4}$
  Here by monotocity we do not have to worry about outputs with $|S| < k$

  Now to compensate for $|S| > k$, we optimize $Z = \frac{w(S)}{OPT_{SDP}} + \theta \frac{n-|S|}{n-k}$.

The results in Feige and Langberg

TABLE 1
Approximation ratios achieved on our four maximization problems. Our results appear in columns in which the SDP technique is mentioned.

Problem	Technique	Approximation ratio	Range
Max-VC$_k$	Random	$1-(1-k/n)^2$	all k
	Greedy	$\max(1-1/e, 1-(1-k/n)^2)$	all k
	LP	$\frac{3}{4}$	all k
	SDP	0.8	$k \geq n/2$
	SDP	0.8	k size of minimum VC
	SDP	$3/4+\varepsilon$	all k, universal $\varepsilon > 0$
Max-DS$_k$	Random	$\frac{k(k-1)}{n(n-1)}$	all k
	Greedy	$O(k/n)$	all k
	LP	$\frac{k}{n}(1-\varepsilon)$	all k, every $\varepsilon > 0$
	SDP	$k/n+\varepsilon_k$	$k \sim n/2$
Max-Cut$_k$	Random	$\frac{2k(n-k)}{n(n-1)}$	all k
	LP	$\frac{1}{2}$	all k
	SDP	$1/2+\varepsilon$	all k, universal $\varepsilon > 0$
Max-UC$_k$	Random/LP	$1-\frac{2k(n-k)}{n(n-1)}$	all k
	SDP	$1/2+\varepsilon_k$	$k \sim n/2$

The results in Jäger and Srivastav

I think the following still represents the latest improvements in cardinality constrained set function maximization for $k = \sigma n$ from Jäger and Srivastav [2005]

I think the following represents the latest improvements in cardinality constrained set function maximization for $k = \sigma n$ from Jäger and Srivastav [2005]

Problem	σ	Prev.	Our Method
MAX-k-CUT	0.3	0.527	0.567
MAX-k-UNCUT	0.4	0.5258	0.5973
MAX-k-DIRECTED-CUT	0.5	0.644	0.6507
MAX-k-DIRECTED-UNCUT	0.5	0.811	0.8164
MAX-k-DENSE-SUBGRAPH	0.2	0.2008	0.2664
MAX-k-VERTEX-COVER	0.6	0.8453	0.8784

Table 1: Examples for the improved approximation factors

The densest subgraph problem

The (unconstrained) densest subgraph problem is defined as follows:

Given a graph $G = (V, E)$, find a subset $V' \subseteq V$ so as to maximize $\frac{|e:u,v \in V'|}{|V'|}$; that is, to maximize the density (or equivalently the average degree) in a subgraph of G.

There is also a directed graph version of this problem. We will consider the undirected case. And also weighted versions of the problem.

The densest subgraph problem

The (unconstrained) densest subgraph problem is defined as follows:

Given a graph $G = (V, E)$, find a subset $V' \subseteq V$ so as to maximize $\frac{|e:u,v \in V'|}{|V'|}$; that is, to maximize the density (or equivalently the average degree) in a subgraph of G.

There is also a directed graph version of this problem. We will consider the undirected case. And also weighted versions of the problem.

The densest subgraph problems can be solved in polynomial time by a flow based algorithm as described in Lawler's 1976 text and improved in Gallo et al [1989]. There is also an LP duality based optimal method given in Charikar [2000] that is the starting point for the MapReduce $(1 + \epsilon)$ approximation algorithm due to Bahmani, Goel and Munagala [2014].

The $(1 + \epsilon)$ approximation follows a MapReduce $2(1 + \epsilon)$ approximation by Bahmani, Kumar and Vassilvitskii [2012] based on the Charikar "reverse greedy" 2-approximation. (Note that these papers use approximation ratios ≥ 1.)

Some comments on the densest subgraph problem

One immediate application is that the densest subgraph can be used to identify a "community" in the web or a social network. There are other applications as well in biological networks.

Some comments on the densest subgraph problem

One immediate application is that the densest subgraph can be used to identify a "community" in the web or a social network. There are other applications as well in biological networks.

The k-densest subgraph problem asks for the densest subgraph V' of size $|V'| = k$. As far as I know Bhaskara et al [2010] have the current best approximation $O(n^{\frac{1}{4}+\delta})$ for the k-densest subgraph problem. It is also known that there cannot be a polynomial time algorithm if the exponential time hypothesis is true.

Some comments on the densest subgraph problem

One immediate application is that the densest subgraph can be used to identify a "community" in the web or a social network. There are other applications as well in biological networks.

The k-densest subgraph problem asks for the densest subgraph V' of size $|V'| = k$. As far as I know Bhaskara et al [2010] have the current best approximation $O(n^{\frac{1}{4}+\delta})$ for the k-densest subgraph problem. It is also known that there cannot be a polynomial time algorithm if the exponential time hypothesis is true.

Obviously if one can (approximately) solve the k-densest subgraph problem then one can (approximately) solve the "densest subgraph with at least (or at most) k vertices. But the converse is not apparentlly true. Andersen and Chellapilla [2009] show the following:

- There is a 3-approximation algorithm for the "at least k vertices" variant.
- If there is a γ approximatin for the "at most k vertices" variant then there is an $\frac{\gamma^2}{8}$ approximation for the k-densest subgraph problem

Charikar's Reverse greedy algorithm for densest subgraph

$S_n := V$
Compute density d_n of S_n
While $n > 0$
  $v^* := argmin_{v \in S_n}[degree(v)]$
  $S_{n-1} := S_n \setminus \{v^*\}$
  $d_{n-1} :=$ density of S_{n-1}
  $n := n - 1$
End While

Charikar's reverse greedy algorithm achieves an $\frac{1}{2}$ approximation.

It would be interesting (at least to me) to formalize (and analyze limitations of) "reverse greedy-like algorithms" similar to our priority formaization of greedy-like algorithms.

The reverse greedy algorithm for maximizing the weight of an independent set in a matroid achieves optimality.