Random Constraint Optimization

The case of Max-Cut

L.M. Kirousis

Department of Computer Engineering and Informatics, University of Patras

&

RA Computer Technology Institute

Joint work with A.C. Kaporis and E.C. Stavropoulos

^ Research partially supported by European Social Fund (ESF), Operational Program for Educational and Vocational Training II (EPEAEK II); by the Research and Academic Computer Technology Institute (RACTI) and by the Future and Emerging Technologies programme of the EU under contract 001907 “Dynamically Evolving, Large-Scale Information Systems (DELIS).”
Circumventing NP-hardness
Circumventing NP-hardness

Reminder: The introduction of randomness in the design of an algorithm cannot avoid the worst cases of an NP-hard problem.
Circumventing NP-hardness

Reminder: The introduction of randomness in the design of an algorithm cannot avoid the worst cases of an NP-hard problem.

Randomized algorithms are good, e.g. when the input is designed by a limited capacity adversary.

In such a case, randomness in the choice of steps of an algorithm further restricts the adversary’s knowledge.
Circumventing NP-hardness

Reminder: The introduction of randomness in the design of an algorithm cannot avoid the worst cases of an NP-hard problem.

Randomized algorithms are good, e.g. when the input is designed by a limited capacity adversary.

In such a case, randomness in the choice of steps of an algorithm further restricts the adversary’s knowledge.

But has no effect if the adversary is omniscient.

Formally: If $\text{NP} \subset \text{BPP}$ then the polynomial hierarchy collapses at the second level and also $\text{RP} = \text{NP}$.
First approach to NP-Hardness

Introduce randomness in the choice of the inputs. That is, assume inputs are drawn according to a distribution.

- Then *perhaps* a realistic distribution might be found, for which worst case inputs are rare.
- However, an unwisely chosen distribution of inputs might trivialize an interesting problem.
An “easy” $\text{SAT}$ distribution

$n$ variables: $\{x_1, \ldots, x_n\}$. $m$ clauses constructed independently as follows: a literal $l$ (positive or negative) is independently placed in each clause with probability $p(n,m)$. Length of a clause is random with expectation $k = 2pn$ (average $k$-SAT).
An “easy” SAT distribution

$n$ variables: $\{x_1, \ldots, x_n\}$. $m$ clauses constructed independently as follows: a literal $l$ (positive or negative) is independently placed in each clause with probability $p(n, m)$. Length of a clause is random with expectation $k = 2pn$ (average $k$-SAT).

- [Franco 1986]: depending on the value of $p$,
  - asymptotically almost all (a.a.a.) instances of average $k$-SAT are satisfied or
  - a.a.a. instances are not satisfied.

- Also: there are very easy algorithms that return a satisfying assignment or a proof of contradiction, respectively.
Moral: Watch out for trivializing distributions of the input.
Consider a CSP with \( n \) variables, fixed arity \( k \) for the constraints and fixed domain size \( D \) for the variables.

Assume that random input instances are generated as follows:

1. First randomly choose the hyperedges of the constraint hypergraph (a hyperedge comprises of the \( k \) variables that a constraint entails),
2. secondly, for each hyperedge \( C \) choose the \( k \)-tuples of values that are to be non-admissible by the constraint corresponding to \( C \).
Consider a CSP with \( n \) variables, fixed arity \( k \) for the constraints and fixed domain size \( D \) for the variables.

Assume that random input instances are generated as follows:

- First randomly choose the hyperedges of the constraint hypergraph (a hyperedge comprises of the \( k \) variables that a constraint entails),

- secondly, for each hyperedge \( C \) choose the \( k \)-tuples of values that are to be non-admissible by the constraint corresponding to \( C \).

[Achlioptas et al. 1997]: such random instances of CSP are a.a.a.a. not satisfiable (in most interesting cases).
In this presentation we focus on Boolean CSP’s about graphs or Boolean formulas. Random model $G_{n,m}$. 
In this presentation we focus on Boolean CSP’s about graphs or Boolean formulas. Random model $G_{n,m}$.

- $n$: number of vertices or variables,
- $m$: the number of edges or fixed-length clauses selected uniformly at random from the sample space.
In this presentation we focus on Boolean CSP’s about graphs or Boolean formulas. Random model $G_{n,m}$.

- $n$: number of vertices or variables,
- $m$: the number of edges or fixed-length clauses selected uniformly at random from the sample space.

Closely related: the random model $G_{n,p}$.

- $p$: the probability for an edge or clause to be included (independently) in the instance generated.

In $G_{n,m}$ ($G_{n,p}$, respectively) the number (expected number, respectively) of edges or clauses is assumed to be $\Theta(n)$ (sparse graphs/formulas).
Algorithmics on random graphs

[Frieze and McDiarmid 1996]:

“...average case analysis ...banishes the pessimism of worst-case analysis.”

“...one can criticise the models as being unrealistic but they are probably no more so than the pathological examples used in the proofs of NP-Completeness.”
Algorithmics on random graphs

[Frieze and McDiarmid 1996]:

- "...average case analysis ...banishes the pessimism of worst-case analysis."

- "...one can criticise the models as being unrealistic but they are probably no more so than the pathological examples used in the proofs of NP-Completeness.”

For Example: Hamilton cycles can be found efficiently in $G_{n,m}$, if $m$ large enough to guarantee, in probability, that the min-degree is at least 2.
Second approach to NP-Hardness

In case optimization is the goal, opt for approximation algorithms.

Approximation Factor of an Algorithm: Ratio of the size of the solution returned by the algorithm to the size of the optimal solution (for maximization <1).
Second approach to NP-Hardness

In case optimization is the goal, opt for approximation algorithms.

Approximation Factor of an Algorithm: Ratio of the size of the solution returned by the algorithm to the size of the optimal solution (for maximization <1).

Extended literature both on:

- **Positive approach**: Find efficient approximation algorithms with as large approximation factor as possible.
- **Inapproximability approach**: Under a putative hypothesis (e.g. NP ≠ P) prove that no efficient algorithm exists with approximation factor larger than a certain value.
Combining both approaches

Both randomness of the input and approximation: Go beyond an inapproximability result by assuming the input is drawn according to a distribution.

Basic strategy:

- Find an upper bound $u_b$ such that the size of solutions is a.a.a. at most $u_b$.

- Design an algorithm that a.a.a. returns a solution with size at least $l_b$ (so $l_b$ can be considered as a typical lower bound).

- Try to show that the ratio $l_b/u_b$ exceeds an approximation factor known to be unattainable (under a putative hypothesis) for all inputs.
Previous results

[Håstad 2001] $\text{MAX-3-SAT}$ cannot be approximated by a factor $> 7/8$.

- Improvement [Interian 2004] to $19/20$. 

Dagstuhl Seminar “Complexity of Constraints”, October 2006 – p. 11/26
Previous results

[Håstad 2001] $\text{MAX-3-SAT}$ cannot be approximated by a factor $> 7/8$.

- Improvement [Interian 2004] to $19/20$.

Above results do not make use of the degree structure of literals in the random input (degree of a literal = number of its occurrences in the formula).

So cannot be directly extended to problems where “degree” is more important than in $\text{MAX-SAT}$, like $\text{MAX-CUT}$.

**$\text{MAX-CUT}$**: Color the vertices of a graph either red or blue, so that the number of bichromatic edges is maximized.
Upper bounds

To show that $ub$ is an upper bound (**First Moment Method**):

- Start with Markov inequality:

  $$\Pr[\exists s : |s| > ub] \leq \text{Ex}(|\{s : |s| > ub\}|).$$

- Show that the rhs expectation above is asymptotically zero.

**Lottery phenomenon**: The probability of winning might be almost zero but the expectation of earnings “measurably positive.”
FMM has been used to get upper bounds for random versions of \( \text{MAX}-3\text{-SAT} \), \( \text{MAX}-k\text{-SAT} \) and \( \text{MAX-CUT} \).

However, results obtained by the “naive” FMM are mostly far from optimal.

**Exception:** \( \text{MAX}-k\text{-SAT} \), where asymptotically with \( k \) the “naive” FMM gives good upper bounds [Achlioptas, Naor & Peres 2003].
Fine FMM

- To make the rhs of Markov inequality

\[ \Pr[\exists s : |s| > \text{ub}] \leq \text{Ex}(|\{s : |s| > \text{ub}\}|) \]

closer to its lhs, exclude large solutions that may only occur in small-probability instances.

- NB: Care must be taken so that an instance with a non-empty set of solutions, still retains at least one solution after the exclusion of its large solutions.

**Application:** \textsc{Max-Cut} [Kaporis et al. 2006]
Results for deterministic $\text{MAX-CUT}$:

- [Goemans & Williamson] 1994 Approximation factor $\alpha_{GW} \approx 0.87856$
- [Håstad 2001] No approximation with factor $> 16/17 \approx 0.94118$ (unless $P=NP$).
- [Khot et al. 2004] Even $\alpha_{GW}$ is optimal assuming the Unique Games and Majority is Stablest conjectures.

NB: some slightly stronger conjectures recently falsified [Charikar et al. 2006]

Goal: Go beyond Håstad’s threshold for Random $\text{MAX-CUT}$. 
Apply FMM to **majority** cuts:

- At least half of the edges incident on any vertex are bichromatic.
- Any vertex of even degree whose exactly half edges are bichromatic is red.

NB: After recoloring a vertex violating any of the above conditions either

- the cut size increases,
- or the cut size remains the same but the number of red vertices increases.
Degree considerations

To compute the expected number of majority cuts exceeding a threshold $\mu_b$ (as a function of the density $d = \frac{m}{n}$, the edges-to-variables ratio), assume that:

$G$ is random conditional that the number of vertices of degree $k$ is fixed and equal to the expectation as if the degree of each vertex were Poisson distributed with mean $2d$.

In other words, we assume that the degree sequence of the random graph is typical.
More about the degrees

Observation: To compute the max cut of a graph, ignore the edges that are incident to a vertex of degree 1 (those must be included in a maximum cut). Therefore:

- Start with a random graph $G$ with typical degree sequence.
- Then recursively delete edges incident on vertices of degree 1, to get a new graph $G'$ known as the 2-core of $G$.

Reminder: the $k$-core of a graph is the maximal subgraph of $G$ obtained
- by edge deletion only,
- and such that all vertices are either isolated or have degree $\geq k$. 
Compute (analytically) the degree sequence of $G'$ by solving the system of differential equations that describes the mean path of the edge deletion process (Wormald’s technique).

Then compute the expected number of *majority* cuts on $G'$ (a non-trivial analytic computation).

NB: $G'$ is random conditional its degree sequence.

Apply FMM.

Thus: for each given value $d > 0$, numerically obtain an upper bound of the max cut $\text{ub}(d)$ that holds for a.a.a. graphs with density $d$. 
The bound $\mathfrak{u}_b(d)$ computed as previously juxtaposed with the bound obtained by direct application of FMM by e.g. [Bertoni et al. 1997] (dashed line).
General strategy:

- Design an algorithm that finds a cut.

- Then for each given density $d$ compute a $\text{lb}(d)$ such that the algorithm a.a.a. returns a cut of size at least $\text{lb}(d)$.

- Method of differential equations (Wormald) for the probabilistic analysis of the algorithm.

- Previously analyzed algorithms did not take advantage of the typical degree sequence, nor of the fact that we can start with the 2-core.
Bird’s eye view of the algorithm

[Kaporis et al. 2006]

- Start with a random graph \( G \), with a typical degree sequence corresponding to density \( d \).
- Find the 2-core \( G' \) of \( G \). The edges deleted to get \( G' \) are part of the cut.
- Successively color one selected vertex of \( G' \) either red or blue.
- Discrepancy of a vertex \( v \) after a coloring step \( t \):

\[
D(v) = |d_b(v) - d_r(v)|, \text{ where}
\]

\[
d_b(v) = \text{number of currently blue neighbors of } v
\]

\[
d_r(v) = \text{number of currently red neighbors of } v
\]
Algorithm—cont/ed

- Always select a vertex with max discrepancy.
- Color it greedily (to guarantee that the number of bichromatic edges to be generated at the current step is maximized).
- Break ties (same max discrepancy) by choosing a vertex where the number of yet uncolored neighbors is minimized (to minimize the impact of the selection on future selections).
Analysis of the algorithm

Method of differential equations:

- Compute the degree sequence of the 2-core $G'$ of $G$.

- Model with differential equations the evolution—at each step of the algorithm—of the following parameters of $G'$:
  - The number of vertices $v$ with a given pair of values for $d_b(v)$ and $d_r(v)$,
  - the number $c$ of the currently bichromatic edges (current cut size).

Previously analyzed algorithms that compute typical-case lower bounds did not make use of the degree sequence.
The bound $lb(d)$ computed as previously juxtaposed with the values of algorithms by [Coja-Oghlan et al. 2003] (dashed) and [Coppersmith et al. 2003] (dotted).
Putting everything together

The approximation ratio $\frac{\text{lb}(d)}{\text{ub}(d)}$. The lower dashed line corresponds to Håstad inapproximability threshold $\frac{16}{17}$, while the upper dashed line to the approximation ratio $0.952 > \frac{16}{17}$ [Kaporis et al. 2006].