Advanced Algorithms - Mandatory Assignment 3

Frank Andersen [fand@itu.dk]

September 2020

Sheet 3

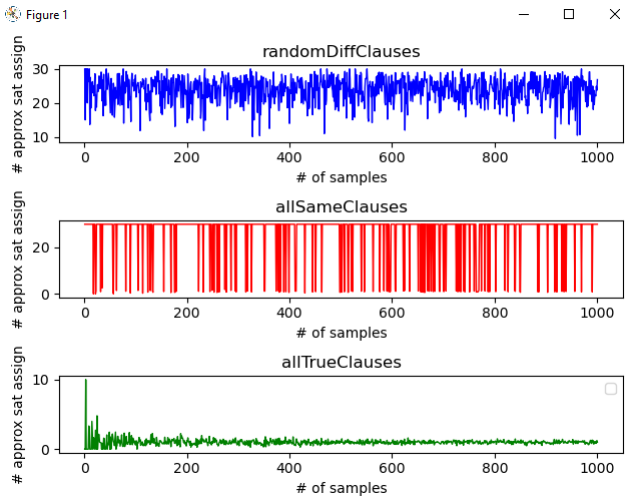
1. Implement the approximate DNF-SAT counting algorithm (Karp-Luby sampling) from the lecture in a language of your choice. Your code should contain functions for

   (a) uniformly sampling from the sample space of pairs (i, a) where i is a clause index and a is an assignment satisfying clause i

   (b) checking whether a sample (i, a) is "good", i.e., checking whether i is the minimal index such that a satisfies clause i.

   Run the algorithm on three interesting DNF formulas (with an appropriate number of variables and clauses) and plot the output of the algorithm as the number of samples grows.

Answer:



Above is the result plot of running my version of the algorithm, tested wiht $t = 30$ clauses, $n = 10$ variables and 1000 samples.

My code can be found here:
https://github.com/dkfrankandersen/ITU_AdvancedAlgorithms/blob/master/third/ApproxKarpLubyDNFCount.py
I have spent way to many hours trying to understand how this should be done, and I'm not confident about the result. A solutions/walk through would be appreciated.

---

2. Solve Exercise 1.1 in Vazirani's book.

1.1 Give a factor 1/2 algorithm for the following.
Problem 1.9 (Acyclic subgraph) Given a directed graph $G = (V, E)$, pick a maximum cardinality set of edges from E so that the resulting subgraph is acyclic.
Hint: Arbitrarily number the vertices and pick the bigger of the two sets, the forward-going edges and the backward-going edges. What scheme are you using for upper bounding OPT?

Answer:

Given a directed graph G we find an maximum acyclic subgraph, e.g. visiting all vertices with no cycles.

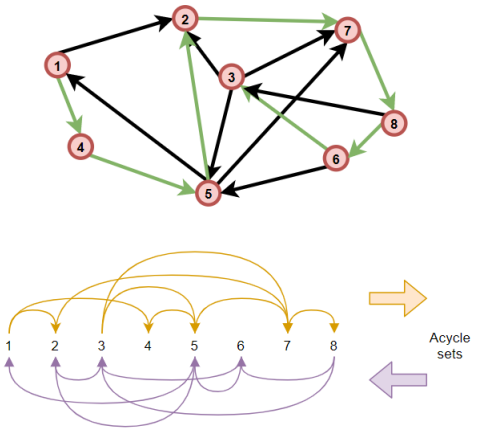


Figure 1: acyclic graph

We create algorithm ALG whichno can approximate the optimal solution,

by a factor 1/2 by arbitrary ordering of the vertices a shown below in figure 1.

This gives us a forwards and backwards going set of edges, ensuring no cycles. We pick the maximum of the two sets, hens forwards in this case.
Prof:

$$\textcircled{1}\ |\overrightarrow{S}| + |\overleftarrow{S}| = |E(G)| \implies max(|\overrightarrow{S}|, |\overleftarrow{S}|) \geq \frac{|E(G)|}{2}$$

The optimal solution is upper bounded by $|E(G)|$

$$\textcircled{2}\ OPT \leq |E(G)|$$

for ① and ② we get that
$$\implies \frac{ALG}{OPT} \geq \frac{\frac{|E(G)|}{2}}{|E(G)|} = 1/2$$

---

3. In this exercise, *we rule out* approximation algorithms for the Traveling Salesperson Problem.
Recall that TSP asks, given a graph $G = (V, E)$ with edge-weights $w : E \longrightarrow \mathbb{N}$, to find a Hamiltonian cycle in G of minimum total weight.

   (a) Assuming $P \neq NP$, show that there is no polynomial-time 2-approximation algorithm for TSP, even when the edge-weights are all non-zero. (Hint: Reduce from the Hamiltonian Cycle problem. Given a graph G for this problem, construct a weighted graph G' for TSP such that a 2-approximation algorithm for TSP allows you to infer whether G has a Hamiltonian cycle.)

Answer:

From G we construct a complete graph G'. For edges in G, we set weight to 1, and for edges not in G $2|V| = 2 \cdot 4 = 8$
Running a 2-approx TSP algorithm on G' it will either use all the original edges or the new one, so a Hamiltonian Cycle exists.
If the output of the 2-approx TSP is $s \leq 2 \cdot |V|$ then edges is from G and a Hamiltonian Cycle, if not, new edges is used and we have $s > 2 \cdot |V|$.
So based on the result of 2-approx TSP algorithm we can conclude whether a path exits in G, if a path exist there is no 2-approx algorithm for TSP.

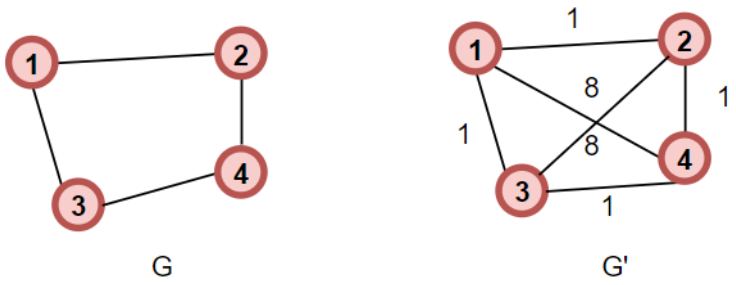


Figure 2: G and G'

(b) Assuming $P \neq NP$, show that there is not even a polynomial-time $p(|V|)$-approximation algorithm (for any polynomial p) for TSP.

Answer:

From above, we know that no 2-approx TSP $\leq 2 \cdot OPT(G)$ exist. For a polynomial-time approx to exist, it would be the same as replacing 2 with any α.

---

4. The Maximum Independent Set problem is NP-hard, and assuming $P \neq NP$, there is not even a polynomial-time algorithm that outputs an independent set of size $\frac{OPT}{n^{1/4}}$.

   (a) In this part, we restrict the problem to graphs G of constant maximum degree. That is, there is some constant Δ such that every vertex in G is incident with at most Δ edges. Maximum Independent Set remains NP-hard on such graphs, even for $\Delta = 3$. Give an algorithm that always outputs an independent set of size at least

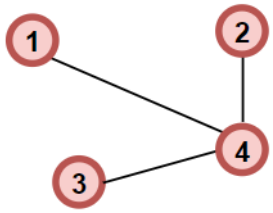
      i. $\frac{OPT}{\Delta+1}$ (greedily put vertices into the independent set, give a lower bound on the number of vertices picked this way, and a trivial upper bound on OPT)

Answer:
In graph G(V,E) to find the maximal independent set, we construct a gready algorithm A that selects a vertex and discards neighbors.

Algorithm 1: Maximal Independent Set

```
let s = empty set
while V is not empty do
    select v ∈ V
    add v to s
    for all neighbor to v do
        remove neighbor from V
    end
end
return s
```

Prof: That A is a $\frac{OPT}{\Delta+}$ approximation algorithm for maximum independent set.
Lets consider graph G, with $\Delta = 3$



G

If the algorithm above pick vertex 4, the solution is $\frac{1}{\Delta+1}$ as Δ is defined for all $v \in V$ this is the worst case, and a lower bound.
We trivally upper bound at $|V|$, giving us $\frac{A(G)}{OPT} \leq \frac{\frac{|V|}{\Delta+1}}{|V|} = \frac{1}{\Delta+1}$

ii. $\frac{OPT}{\Delta}$ (same approach as before, but find a tighter upper bound on OPT—this part may be trickier)

Answer:
Not answered

(b) In this part, we restrict the problem to *planar* graphs. Such graphs can be drawn in the plane without crossings, and Maximum Independent Set remains NP-hard on such graphs. A famous theorem establishes that planar graphs are 4-colorable, i.e, their vertices can be colored using 4 colors such that the two endpoints of any edge receive distinct colors. Assuming you get an algorithm that outputs a 4-coloring of a planar graph, show how to construct an algorithm

that always outputs an independent set of size at least $\frac{OPT}{4}$.

Answer:
For a planar graph, a 4 coloring algorithm will output the vertices of different colors, so this should be no more than $\frac{|V|}{4}$
