KTH Computer Science and Communication

Current Research in Proof Complexity: Problem Set 1

Due: December 4, 2011. Submit as a PDF-file by e-mail to `jakobn at kth dot se` with the subject line `Problem set 1:` ⟨`your name`⟩. Solutions should be written in LaTeX or some other math-aware typesetting system. Please try to be precise and to the point in your solutions and refrain from vague statements. In addition to what is stated below, the general rules stated on the course webpage always apply.

Hints: For most or all problems, "hints" can be purchased at a cost of 5–10 points. In this way, you can configure yourself whether you want the problems to be more creative and open-ended, where sometimes a lot can depend on finding the right idea, or whether you want them to be more of guided exercises providing a useful work-out on the concepts of proof complexity. If you do not solve a problem, there is no charge for the hint (i.e., it is not deducted from the score on other problems).

Collaboration: Discussions of ideas in groups of two to three people are allowed—and indeed, encouraged—but you should write down your own solution individually and understand all aspects of it fully. For each problem, state at the beginning of your solution with whom you have been collaborating. Everybody collaborating on a certain problem is considered to have purchased a hint if one of the collaborators has done so.

Reference material: Some of the problems below are "classic" and hence their solutions can probably be found on the Internet or in research papers. It is not allowed to use such solutions in any way unless explicitly stated otherwise. Anything said during the lectures on in the lecture notes should be fair game, though, unless you are specifically asked to show something that we claimed without proof in class. It is hard to pin down 100% formal rules on what all this means—when in doubt, ask the lecturer.

About the problems: Note that as explained in class, this problem set is *considerably more comprehensive* than following problem sets are expected to be. Some of these problems are meant to be quite challenging and you are not necessarily expected to solve all of them. As a general guideline, a total score of around 120 points on this problem set should be enough to get a pass. Any corrections or clarifications will be posted on the course webpage `www.csc.kth.se/~jakobn/teaching/proofcplx11`.

1 (10 p) In the first lecture, we said that any propositional logic formula F can be transformed to CNF formula F' such that F' only linearly larger than F and is unsatisfiable iff F tautology. We did do the full transformation, however, and your task now is to fill in the missing details regarding the connectives $\wedge$ and $\leftrightarrow$.

Given a formula $F \doteq G \wedge H$, show how to write CNF clauses that force x_F to take the value of $G \wedge H$ assuming that x_G and x_H are computing G and H, respectively. Then solve the same problem for the formula $F \doteq G \leftrightarrow H$.

2 (10 p) In the second lecture, we introduced the *weakening rule* in resolution, which allows us to derive $C \vee D$ from the clause C for any clause D, and then claimed that this rule can be eliminated without loss of generality. Prove this formally.

That is, prove that if $\pi : F \vdash \bot$ is a resolution refutation with weakening, then there is another resolution refutation $\pi' : F \vdash \bot$ that does not use the weakening rule and has at most the same length, clause space, and width as π.

3 (10 p) Prove that, again as claimed in class, restrictions preserve resolution derivations. That is, show that if $\pi : F \vdash D$ is a resolution derivation of a clause D from a CNF formula F and ρ is a restriction, then $\pi\restriction_\rho$ is a derivation of $D\restriction_\rho$ from $F\restriction_\rho$ in at most the same length, clause space, and width as π.

Do you need the weakening rule for this statement to hold? Why or why not? If you do need weakening, is there any way of getting a similar statement without using the weakening rule?

4 (10 p) Prove that a (bipartite vertex) (d, s, κ)-expander is a $(d, s, 2\kappa - d)$-unique-neighbour expander.

5 (30 p) To complete the proof of the lower bound for PHP_n^{n+1} by Ben-Sasson and Wigderson presented in class, we need to establish the existence of $(5, n/c, 3)$-expanders (which is sufficient to get unique-neighbour expansion by problem 4). Prove that if we generate a random bipartite graph $G = (U \dot\cup V, E)$ with $|U| = n + 1$ and $|V| = n$ by picking for each $u \in U$ a random set of 5 neighbours in V chosen uniformly and independently at random from all $\binom{n}{5}$ subsets of vertices with replacement, then there is a universal constant $c > 0$ such that G is a $(5, n/c, 3)$-expander with high probability for large enough n (which in particular means that such graphs must exist).

Hint: Focus on what it means that G would *fail* to be an expander. Consider all the ways in which this can happen, and calculate the probability using the tips given during the lecture. Make sure to explain properly what is going on in your calculations.

6 (30 p) Let F be an unsatisfiable CNF formula and let α denote any truth value assignment to the variables in F. The *search problem* for F given α is to find a clause $C \in F$ falsified by α.

A *decision tree* T_F for F is a binary tree with leaves labelled by clauses in F, internal vertices labelled by variables x, and two edges from each internal vertex labelled 0 and 1. Any assignment α defines a path through T_F starting from the root and following from each internal vertex x the edge label agreeing with the value assigned to x by α. Such a path ends in some leaf C, which is the answer of T_F on α. The tree T_F *solves* the search problem for F if on any α the answer C is a clause falsified by α.

Let us write $S_D(F)$ to denote the minimal size (i.e., number of vertices) of any decision tree solving the search problem for F. Recall that $L_\mathcal{T}(F \vdash \bot)$ denotes the minimal length of any tree-like resolution refutation of F.

We claimed in class that decision trees and tree-like resolution refutations are essentially the same. Your task is now to formalize this claim as follows.

6a Prove that $S_D(F) \leq L_\mathcal{T}(F \vdash \bot)$ by showing that any tree-like resolution refutation of F can be made into a decision tree solving the search problem for F.

6b Prove that $L_\mathcal{T}(F \vdash \bot) \leq S_D(F)$ by showing that any decision tree solving the search problem for F can be made into a tree-like resolution refutation of F. (For partial credit, just prove $L_\mathcal{T}(F \vdash \bot) = \mathrm{O}(S_D(F))$ using weakening.)

6c Argue that this proves the implicational completeness of resolution, and show that any unsatisfiable CNF formula over n variables has a resolution refutation π in simultaneous length $L(\pi) = \mathrm{O}(\exp(n))$ and clause space $Sp(\pi) = \mathrm{O}(n)$. What are the best concrete bounds you can get, not using big-oh notation but providing explicit constants instead? What bounds can you get expressed in terms of the number of clauses m of the formula?

7 (30 p) In the first lecture, we discussed the proof systems resolution, Cutting Planes, and Polynomial Calculus but did not say much about how they are related.

7a Prove that Cutting Planes can polynomially simulate resolution by showing that given any resolution refutation $\pi : F \vdash \bot$, Cutting Planes can simulate this refutation line by line in almost the same length, size and space (where size is the total number of literals plus in Cutting Planes also the sums of the logarithms of all the coefficients, and where space is the number of clauses in resolution and the number of inequalities in Cutting Planes). Making clear what "almost" means is part of the problem, but any increase should be small.

7b Is it true that Polynomial Calculus can simulate resolution in the same way? Make the necessary modifications to the proof in problem 7a or explain why this cannot be done.

8 (50 p) In the fourth lecture, we defined black-white pebbling and pebbling contradictions, and argued rather informally that the properties of pebbling contradiction CNF formulas with respect to resolution are related to properties of their corresponding graphs with respect to the black-white pebble game. We now want to make this more precise.

In this problem, assume that G is a directed acyclic graph (DAG) with one unique sink (usually denoted z) and with all non-source vertices having a constant number of incoming edges (fan-in 2 can be assumed without loss of any credit).

8a Prove that given a pebbling strategy $\mathcal{P}$ for G that only uses black pebbles, there is a resolution refutation π of Peb_G in length $L(\pi) = \mathrm{O}\big(\mathsf{time}(\mathcal{P})\big)$ and clause space $Sp(\pi) = \mathrm{O}\big(\mathsf{space}(\mathcal{P})\big)$.

8b Prove that given a resolution refutation $\pi : Peb_G \vdash \bot$, there is a black-white pebbling strategy $\mathcal{P}$ such that $\mathsf{time}(\mathcal{P}) = \mathrm{O}(L(\pi))$ and $\mathsf{space}(\mathcal{P}) = \mathrm{O}(TotSp(\pi))$. For simplicity, let us specify explicitly that the resolution refutation does *not* use weakening.

Hint: For this problem there are two helpful technical assumptions that you are allowed to make without a proof (although it is not too hard to show that these assumptions can be made without loss of generality):

- Any clause appearing in any configuration of the proof (other than the final empty clause) is resolved over at least once before being erased.
- When a clause is erased from a configuration after having been used in a resolution inference for the last time, it is erased immediately after this final resolution inference step (or, if both clauses used for the inference are erased, then they are erased immediately after one another in any order you prefer).

8c Can you improve the claim in problem 8a to the statement that given any black-white pebbling strategy $\mathcal{P}$ for G, there is a resolution refutation π of Peb_G in length $L(\pi) = \mathrm{O}\big(\mathsf{time}(\mathcal{P})\big)$ and clause space $Sp(\pi) = \mathrm{O}\big(\mathsf{space}(\mathcal{P})\big)$? If so, what modifications are needed in the proof? If this seems hard, can you pinpoint what is the problem?

9 (40 p) We have seen that pigeonhole principle formulas PHP_n^{n+1} are hard in theory, but what does this mean in practice? In this problem, you are asked to investigate this by running the state-of-the-art SAT solver *MiniSAT* on various flavours of pigeonhole principle formulas and report your results.

Some helpful practical information about MiniSAT and about the standard *DIMACS* format used in SAT solving can be found on `www.csc.kth.se/~jakobn/teaching/proofcplx11/minisat.php`.

For this problem, do not submit the code, but instead describe how it works. Place the actual code in a directory in the AFS file system where `jakobn` has reading and listing permission `rl` (as shown by `fs la .`). Note that permission `l` is needed for the whole path leading to the directory. Make sure your code works in the CSC Ubuntu Linux environment. Include a Makefile in the directory, or a shellscript `make` that will compile your code. If there are problems with any of the above, contact the lecturer to agree on some other technical solution.

9a Write a program that takes as input a file specifying bipartite graph G and outputs a file containing the formula $PHP(G)$ in DIMACS format. If given no arguments on the command line, the program should read the graph from standard input and write the formula on standard output. The format of the input graph file should be:

```
m n
1 : neighbour1  neighbour2 ...  neighbourD1
2 : neighbour1  neighbour2 ...  neighbourD2
...
m : neighbour1  neighbour2 ...  neighbourDm
```

where the first line specifies the number of vertices on the left m and on the right n, and where each following line specifies a vertex on the left followed by a colon and then a list of neighbours on the right separated by spaces.

9b Feed complete bipartite graphs $K_{n+1,n}$ to this program to generate formulas PHP_n^{n+1}, and run MiniSAT on these formulas. How large instances can you solve? How does the running time scale with n? With the total size of the formula (measured as the total number of literals counted with repetitions)? Do you get better or worse results when you turn off preprocessing (`-pre=none`) of the formula in MiniSAT? Please specify what kind of hardware you have run your experiments on (processor, clock frequency, amount of internal memory). Providing nice plots of the results will give brownie points.

9c Generate random graphs G according to the distribution described in problem 5 and use the program from problem 9a to obtain the corresponding CNF formulas $PHP(G)$. For how large n can MiniSAT solve these formulas? Do they seem to be easier or harder than PHP_n^{n+1} for the same n? How does the time scale with n? With the total size of the formula? Do you get better or worse results when you turn off preprocessing?

10 (40 p) Sudoku is played over a 9x9 grid, divided to 3x3 subgrids called "regions," where some of the grid cells are already filled with numbers 1 to 9. The goal is to fill the remaining cells with numbers 1 to 9 so that when the whole 9x9 grid has been completed, every number appears exactly once in each row, each column, and each region. For more details see, e.g., `en.wikipedia.org/wiki/Sudoku`. The purpose of this problem is to investigate if and how SAT solvers can be used to solve Sudoku puzzles.

10a Describe a way to encode a specific Sudoku instance as a CNF formula in such a way that the formula is satisfiable if and only if the Soduko puzzle has a solution, and so that a solution to the puzzle can be read off from any satisfying assignment to the formula. The encoding should be explicit and have reasonable size and complexity.

In case you happen to consider several different options, describe what these are and discuss what you think are possible pros and cons. (All such variants should be correct, of course. Also, this is optional in the sense that only one correct encoding is needed for full credit.)

10b For this problem, do not submit the code, but instead follow the technical instructions given in problem 9. If there are any problems, contact the lecturer to agree on some other technical solution. The information on the webpage `www.csc.kth.se/~jakobn/teaching/proofcplx11/minisat.php` may come in handy here as well.

Write a program that generates your CNF encoding from problem 10a for a a given Sodoku instance and outputs it in DIMACS format. The format of the Sudoku input file should be as in the following example:

```
2--8-6--5
---------
76-1-3-92
53-----28
-2-3-1-7-
17-----64
49-7-8-31
---------
6--9-5--7
```

That is, the file contains 9 rows with 9 characters in each row, and "`-`" denotes an empty cell. If given no arguments, the program should read a Sudoku instance from standard input and write the CNF formula on standard output.

Write another program that reads a satisfying assignment as produced by MiniSAT and writes a solved puzzle on the format described above. Again the program should read from standard input and write on standard output if no command line arguments are given.

How good is MiniSAT at solving Sudoku, and how much time does it take when the solver is successful? Can you find a solvable or unsolvable Sudoku instance that MiniSAT cannot handle? What happens for the empty instance (i.e., no figures in any cells)? What happens for an overconstrained instance (e.g., if you look at a solution and fill in some cell in the original puzzle in a way that is in conflict with this solution)? In case you were considering different encoding options above, do they seem to make any difference in practice? (You do not need to answer this final question in order to get full credit.)

To get you started, the directory `www.csc.kth.se/~jakobn/teaching/proofcplx11/files` contains four files `sudoku1.txt` to `sudoku4.txt` in the format described above.