KTH Computer Science
and Communication

Current Research in Proof Complexity: Problem Set 5

Due: June 19, 2012. Submit as a PDF-file by e-mail to `jakobn at kth dot se` with the subject line `Problem set 5:` ⟨`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 for that problem if one of the collaborators has done so.
Reference material: For some of the problems, it might be easy to find solutions on the Internet or in research papers. It is not allowed to use such material in any way unless explicitly stated otherwise. You can refer without proof to anything said during the lectures on in the lecture notes, except in the obvious case when 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: 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 140 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) Prove that if a CDCL solver uses the DECISION clause learning scheme as described in [Atserias et al. '11] — i.e., learning the clause A_1 in the notation used in the paper and in our lectures — then the clause learned will be an asserting clause which only contains decision variables and no implied variables. (We used this fact without proof in the lectures.)

2 (20 p) We saw in lecture 17 that using random restrictions of the right form, we can prove size-space trade-offs for pebbling formulas with XOR-substitution where the upper bounds hold for both resolution and polynomial calculus and the lower bounds hold for PCR. Can the same approach be used to obtain trade-offs also for cutting planes? Show how to modify the proof or explain why this does not work.

3 (30 p) Recall that a derivation in *unit resolution* is a derivation where for each application of the resolution rule, one of the clauses is unit (i.e., has size 1). Prove that $Sp_{\mathcal{R}}(F \vdash \bot) \leq 3$ if and only if there is a unit resolution refutation of F.

4 (40 p) Prove the properties of 1-empowerment and absorption crucially used in [Pipatsrisawat & Darwiche '11] but which we just stated as Observation 9 without proof in our lectures. That is, using terminology as in the lectures, prove the following:

4a If $C \subseteq C'$, then C' is absorbed by C.

4b Adding more clauses to the knowledge base can absorb a 1-empowering clause but never make an absorbed clause 1-empowering.

4c Every asserting clause is 1-empowering with respect to the knowledge base at the time of its derivation with its asserted literal as empowering.

5 (50 p) The purpose of this problem is to compare the power of resolution and cutting planes, and also to get some exercise in encoding combinatorial principles as CNF formulas.

5a Show that cutting planes can refute the pigeonhole principle formulas PHP^m_n for $m > n$ with proofs in small size (which yields an exponential separation between cutting planes and resolution).

Hint: Count the number of holes used by the pigeons and conclude that it is larger than the number of pigeonholes available.

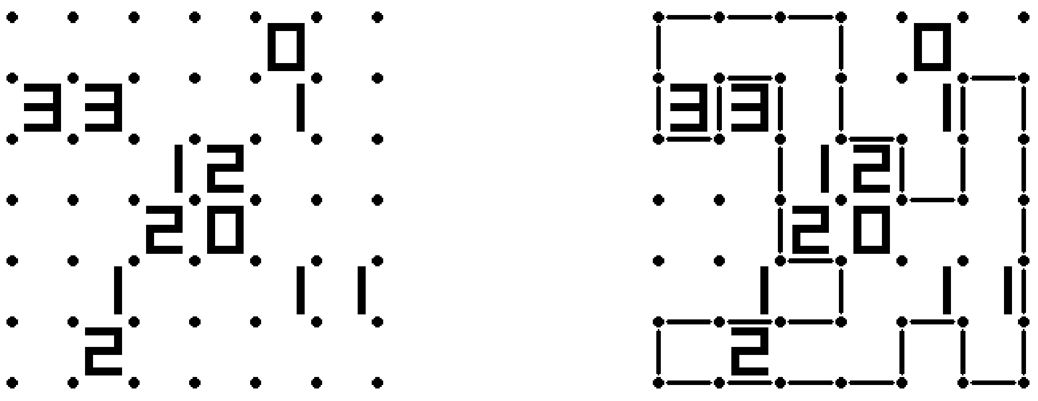
5b Let us now consider an alternative encoding of the pigeonhole principle where the pigeons get access to their hole using keys, every key opens some (non-empty) subset of pigeonholes, and the keys should be distributed among the pigeons in such a way that every pigeon can open only its own private hole. We will denote this formula by $KPHP^{m,k}_n$ for "key pigeonhole principle" (or maybe more appropriately, albeit with a slight misspelling, "konvoluted pigeonhole principle").

As before we have m pigeons and n pigeonholes for $m > n$, but we also have k keys to the holes, where k is (potentially much) larger than m. The formula $KPHP^{m,k}_n$ says that no two pigeons should get the same key. If some pair of pigeons get some pair of keys, then both of these keys are being used, but for privacy reason no two keys currently in use should open the same pigeonhole.

Write down the CNF formula $KPHP^{m,k}_n$ as described above, using variables $p_{i,\ell}$ to mean that pigeon i gets key ℓ, $h_{\ell,j}$ to mean that key ℓ opens pigeonhole j, and $u_{\ell,\ell'}$ to mean that keys ℓ and ℓ' are both in use. (Note that there are some conditions on the variables which are only implicit above, but which are necessary to capture the combinatorial principle as described. Such conditions should be encoded in the same spirit as is done for PHP^m_n.)

5c Can you use the CP-refutation in your solution of problem 5a to refute $KPHP^{m,k}_n$ efficiently? If so, describe how to modify the CP-refutation of PHP^m_n to achieve this, or else explain why this more convoluted encoding of the pigeonhole principle seems harder for CP.

6 (60 p) In this problem we want to study one of the key lemmas in [Pipatsrisawat & Darwiche '11] and the way it was presented in our lectures (as Lemma 16 or the "3rd key lemma"). In particular, while proving the lemma the lecturer claimed that during the whole process of turning a literal into non-empowering in the proof, any variable in the formula will be asserted by the CDCL solver only once. In discussions after class, it was further claimed that this in turn implies that we can absorb a clause after $O(n^2)$ conflicts, which would improve the $O(n^4)$ bound in the polynomial simulation result to $O(n^3)$. Your task is to verify these claims.



(a) Slitherlink problem instance. (b) Solution.

Figure 1. Slitherlink puzzle with solution

6a Is it true that every variable gets asserted only once during the process to turn a literal into non-empowering? For full credit, a correct formal proof of or convincing counter-example to this claim is needed.

6b Can you show, using either a positive answer to the claim in Problem 6a (if that was what you established), or possibly arguing by other means, that one can absorb a clause after only $O(n^2)$ conflicts instead of $O(n^3)$ conflicts as proven in class? Again a correct proof or convincing counter-example is needed for full full credit.

7 (80 p) *Slitherlink* is yet another one of those Japanese combinatorial puzzles that are fun to study also in the context of SAT solving. Each Slitherlink puzzle consists of a rectangular lattice of dots with some *clues* in various places in the form of integers between 0 and 3. The object is to connect the dots surrounding each clue so that the number of lines equals the value of the clue and also so that the lines around all clues form one continuous loop with no crossings or branches. Empty squares without clues may be surrounded by any number of lines. An example Slitherlink puzzle with solution is given in Figure 1.

For more information, including a more detailed description of the rules which might be helpful when solving this problem, see e.g. the Wikipedia page `en.wikipedia.org/wiki/Slitherlink` (from which the example above has been taken). To get a feel for how to solve Slitherlink puzzles by hand, see the tutorials at `www.nikoli.com/en/puzzles/slitherlink/rule.html` or `www.conceptispuzzles.com/index.aspx?uri=puzzle/slitherlink/tutorial`.

The purpose of this problem is to use SAT solvers to solve Slitherlink puzzles. In your solution, do not submit any code, but instead describe how the algorithm 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 KTH 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. For the SAT solving you should use MiniSAT. Some helpful practical information about MiniSAT and about the standard *DIMACS* format used by MiniSAT and other SAT solvers can be found on the webpage `www.csc.kth.se/~jakobn/teaching/proofcplx11/minisat.php`.

```
7 x 7

 1 - 2 - - - 3

 - 3 - - 0 - 3

 3 - - - 2 - 3

 - - - - - - -

 3 - 3 - - - 2

 2 - 3 - - 2 -

 2 - - - 3 - 3
```

(a) Input format.

```
   ---   -----
 1|  2| |    3|
   -   -     -
   3|    0  |3
 ---   -     -
|3    | |2   3|
 -----   -   -
          | |
 -   -----   -
|3| |3       2|
| |  -   ---  |
|2|  3| |  2| |
|  ---   -  | |
|2       3| |3|
 ---------   -
```

(b) Output format

7a Describe how Slitherlink puzzles can be solved using a SAT solver as a subroutine. Your account should be brief and concise, but should in principle contain enough information for a competent programmer to translate the description into working code without having any knowledge of Slitherlink (other than what problem instances look like). In particular, do not leave out any crucial details in any translations of problem instances to CNF formulas.

Hint: There are different approaches here, in that you can use either single calls or repeated calls to the SAT solver. Be warned that there are some subtleties involved to get a solution that is guaranteed to be correct, and for full credit you need to deal with these subtleties. 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 solution is needed for full credit.)

7b Write a program that implements the algorithm you describe in your solution to problem 7a. If given no arguments, the program should read a Slitherlink instance from standard input and write the solution on standard output (or, if no solution exists, print a message to this effect).

The format of the Slitherlink input file is as in Figure 2(a). That is, the first line `<int> x <int>` specifies the dimensions of the Slitherlink puzzle grid (rows x colums). Then the puzzle grid follows, row by row, but with a blank line before each row with clues (to simplify comparison of the problem instance with the solution printed as described below). Each row starts with a blank and then contains the clues separated by further blanks. A dash instead of a number signifies an empty square without any clue. Each row is terminated by a newline character.

To print the solution, first erase the first line in the input file and convert all dashes encoding non-clues to blanks. Then use dashes and horizontal bars to indicate the Slitherlink loop as shown in Figure 2(b).

7c Use your implementation in problem 7b to investigate if and how MiniSAT can be used to solve Slitherlink puzzles efficiently.

There are tons of different Slitherlink instances on the web — can you find an instance that is not solved within 5 minutes of processor time on `u-shell.csc.kth.se` as reported by e.g. `top`? (Note that we are interested in actual processor time here, not wall-clock time. Also note that here it is required that the hardness should not be the result of an obviously inefficient CNF encoding, but should in some intuitive sense be intrinsic to the instance.)

What happens if you take a solvable instance and add an inconsistent clue somewhere — does the instance get harder or easier?

In case you were considering different options for your algorithm, 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 `slitherlink1.txt` to `slitherlink4.txt` with (solvable) Slitherlink puzzles in the format described above. Note, however, that for full credit you are required to evaluate your program on a broader set of benchmarks not limited to these example instances.