CMPT 417/827, Fall 2020, Assignment 2

You are to carry out, and report on, an experiment in SAT-based solving of Quasigroup Completion Problems (QCP), a.k.a. Latin square completion. Follow these steps:

(1) **Consider** the following properties of order $n \times n$ (order n) Latin squares. We will write $[n]$ as an abbreviation of $\{1, \ldots, n\}$.

  (a) Each cell of the square contains a number in $[n]$;

  (b) No cell contains two distinct numbers;

  (c) No row has two cells containing the same number;

  (d) No column has two cells containing the same number;

  (e) Every number in $[n]$ appears in every row;

  (f) Every number in $[n]$ appears in every column.

  Let A be the set of n^3 atoms of the form $C_{i,j,k}$, with $i, j, k \in [n]$. Interpreting $C_{i,j,k}$ as meaning "Cell (i,j) contains k", we can write CNF formulas over A that correspond to each property. Various combinations of the properties amount to a correct definition of the Latin squares of order n, and the corresponding CNF formulas will define the set of truth assignments corresponding to these squares. By adding appropriately chosen unit clauses, we can make such a formula define the completions of a partial Latin square, thus giving us a transformation from QCP to SAT.

(2) **Choose** at least 3 distinct ways of transformating QCP to SAT. At least one should be a minimal subset of the properties (a) through (f). At least one should consist of this set, plus some additional properties among (a) through (f). For the third set, you can choose any other correct subset of clauses corresponding to (a) through (f), or you can add other "redundant" clauses, or carry out some other modification that does not change correctness but might affect solver performance. Make a prediction of which transformation will result in the best performance, and which the worst, and write it down. (You may find it useful to look at "Modelling Choices in Quasigroup Completion: SAT vs. CSP", by Ansotegui, del Val, Dotu, Fernandez and Manya. You can ignore the parts about CSP.)

(3) **Implement** a program that transforms an instance of QCP to a CNF formula representing its solutions (if any). Make one version of the program for each of your three chosen clause sets. You can use any programming language you like as long as it can be compiled and run on the CSIL Linux workstations. The preferred program design is as a Linux command-line utility using standard input and output so the command

```
%>  ./qcp2cnf  < example1.qcp > example1<....>.cnf
```

  inputs the file example1.qcp, and outputs the resulting CNF to file example1-$\langle \ldots \rangle$.cnf, where the $\langle \ldots \rangle$ is something mnemonic about the transformation used to produce the formula. (Alternately, your program can take two command-line arguments: `%> ./qcp2cnf example1.qcp example1.cnf`.) The formats for the input and output files are as follows.

- **QCP instance** files have extension `.qcp`. The remainder of the name should uniquely identify each instance. The first line of the file, is the string "order n". The rest of the file consists of n lines, one for each row of the board. Each line is a sequence of n "words", separated by one or more spaces. Each word is an integer in $[1, n]$ (for a fixed entry) or "." (for a value that must be found). Here is the first 4 lines of an order 12 instance, with 4 un-known entries on each row.

  ```
  order 12
  1 6 3 . 4 . 12 10 . . 5 9
  . . 8 4 . . 6 5 3 2 1 7
  2 3 6 . 5 . 10 7 . . 8 11
  ⋮
  ```

- **CNF formulas** have extension `.cnf`, and will be in DIMACS format, the standard for SAT solvers. The file can start with comment lines, which begin with the character c. After the comments, a line of the form "`p cnf nvars nclauses`", specifies that the instance is in CNF, has exactly nvars distinct atoms, and nclauses clauses. (Most SAT solvers will work fine if there are not exactly nvars atoms or nclaues clauses, as long as nvars and nclauses are upper bounds on the actual numbers, but some might not.) The clauses follow, one per line, each given as a sequence of distinct non-zero integers from the interval [-nvars,nvars], separated by spaces, and terminated with a 0. A line may not contain a number and its negation. Positive numbers denote corresponding atoms; Negative numbers denote their negations. For example:

  ```
  c
  c start with comments
  c
  p cnf 5 3
  1 -5 4 0
  -1 5 3 4 0
  -3 -4 0
  ```

  represents the formula

  $$(1 \vee \neg 5 \vee 4) \wedge (\neg 1 \vee 5 \vee 3 \vee 4) \wedge (\neg 3 \vee \neg 4)$$

(4) **Download** at lease one SAT solver. There are many on-line – you should choose one that has taken part in at least one SAT solving competition, to ensure you get a reasonable one. (The competition pages have information about the solvers.) Source code may or may not be available; some compile on Windows and Linux, some on just one; sometimes there can be issues with out-of-date libraries. (They are mostly research code, not commercial products.) Run the solver on all the the CNF formulas generated by all of your transformations on a collection of QCP instances of at least three different sizes. Make plots illustrating the effect of the choice of transformation on the solver running time. (More details on instances and plots will be provided elsewhere.)

(5) **Write** a brief report, including (brief) explanations of what you did, what you found, how to use your program, your file naming, etc. When your program and report are finished, create a .zip file containing all your files, and submit it on Coursys. Your report should be in a file called Report.pdf. Other files and folders should be helpfully named. Include in the .zip file a README file that explains how to compile your program, if it requires compilation, or any other information needed to run it.