Byron Chiang 200072414
Problem Set 3 CMPT417

Summary
For this latin square completion assignment I wrote three slightly different versions of C++ programs (ps3.cpp, ps3b.cpp, and ps3c.cpp) that transform QCP instances into DIMACS CNF format.

The first variation (ps3.cpp) spurts out a set of CNF that corresponds to Latin Square Properties a), c), and d) in the assignment sheet.

The second variation (ps3b.cpp) spurts out a set of CNF that corresponds to properties a), b), c), and d).

The third variation (ps3c.cpp) spurts out a set of CNF that corresponds to properties a), b), c), d) and e).

Some work-up steps (i.e. De Morgan/Distributive/Associative Law) have been applied to each original subset of CNF formulas, making the code easier to write and comprehend. You will see that the resulting CNFs do not necessarily "translate" directly into properties a) to e) in English, but I can assure you that they are 100 percent equivalent – or they should be to the best of my knowledge.

In order to sequester the effects of multiprocessing I chose just to use the simple version of glucose SAT solver, which solves instances with only one CPU.

Before sending the parser output to the SAT solver, I have predicted:

1. in general the files generated with the first parser (ps3.cpp) - minimal subset - give the best results because they have the least number of clauses when compared to files generated with the other two parsers (about 25% fewer than subset 2, about 50% fewer than subset 3).

2. Files generated with the second parser will come in second in terms of performance, since they contain 20 to 25% fewer clauses than files generated with the third parser.

3. However, due to the effects of (unit) propagation and multiprogramming (OS process scheduling) I suspect I will not observe a 25% increase in CPU time every time I increase the number of clauses by 25%, but keep the Latin square order constant.

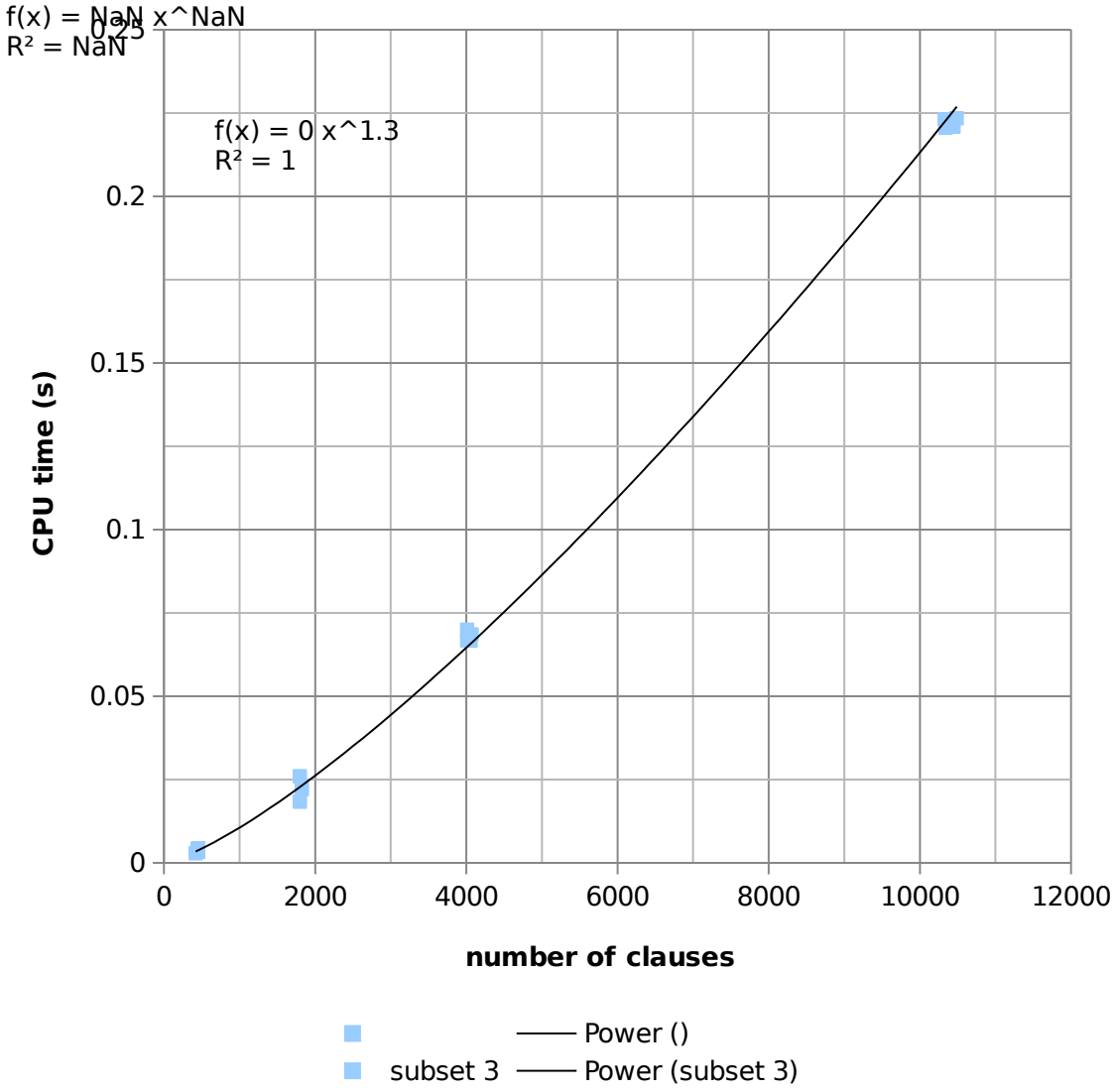
4. Since QCP is NP-complete I predict the worst case is polynomial time

5. Instances with more unit clauses are more likely to be solved faster (due to resolution, Horn clauses etc.)

I pushed four sets of instances (order 10, order 20, order 30, order 49) to the SAT solver and here are the results:

		Instance 1 # clauses	CPU time in seconds	**Instance 2** # clauses	CPU time in seconds	**Instance 3** # clauses	CPU time in seconds	**Instance 4** # clauses	CPU time in seconds
Order	10								
# variables	1000								
	minimal subset (Property A, C, D)	240	0	251	0.006212	250	0	249	0.002387
	subset 2 (Property A,C,D,B)	340	0.004005	351	0	350	0.005324	349	0.005263
	subset 3 (Property A,C,D,B,E)	420	0.002881	453	0.003459	450	0.004384	447	0.004319
Order	20								
# variables	8000								
	minimal subset (Property A, C, D)	1000	0.014146	1000	0.018554	1008	0.018867	1007	0.017899
	subset 2 (Property A,C,D,B)	1400	0.022616	1400	0.017666	1408	0.021543	1407	0.020762
	subset 3 (Property A,C,D,B,E)	1800	0.026042	1800	0.018392	1824	0.022565	1821	0.022105
Order	30								
# variables	27000								
	minimal subset (Property A, C, D)	2236	0.04719	2252	0.045634	2257	0.049105	2237	0.052848
	subset 2 (Property A,C,D,B)	3136	0.055772	3152	0.049305	3157	0.046881	3137	0.054873
	subset 3 (Property A,C,D,B,E)	4008	0.070017	4056	0.0667	4071	0.0686	4011	0.066769
Order	49								
# variables	117649								
	minimal subset (Property A, C, D)	5897	0.183559	5883	0.18139	5844	0.178934	5848	0.176031
	subet 2 (Property A,C,D,B)	8298	0.19012	8284	0.188392	8245	0.200864	8249	0.172396
	subet 3 (Property A,C,D,B,E)	10488	0.223406	10446	0.220919	10329	0.223086	10341	0.220557

The average case for all three approaches, as indicated by the plot below, is between order n and order n^2, which conform with my findings.



Running the program(s)

There are three ps3.cpp, ps3b.cpp and ps3c.cpp, stored in
/asn3/minimal-subset
/asn3/minimal-plus-1
and
/asn3/minimal-plus-2
respectively.

Each folder contains the same set of selected instances, and a version of qcp2cnf that corresponds to:

1. a minimal subset of properties of order n latin square (a, c, d) - in /asn3/minimal-subset
2. four properties of order n latin square (a, c, d, b) - /asn3/minimal-plus-1
3. five properties of order n latin square (a, c, d, b, e) - /asn3/minimal-plus-2

I wrote and compiled the C++ code with Codeblocks 13.12. The parsers and the single-core version of glucose have already been compiled, but depending on the your Linux machine you may have to modify all access rights using the function “chmod”. For example:

```
$       chmod 777 qcp2cnf
```

To parse a qcp instance file, do the following:

1. open Linux terminal

2. go to any of the three folders /minimal-plus-2, /minimal-plus-1, or /minimal-subset

3. the program qcp2cnf takes two arguments. The first argument corresponds to the qcp input file and the second to the desired cnf output in DIMACS format. For instance,

```
$       ./qcp2cnf q_10_01.qcp q_10_01.cnf
```

takes the qcp instance file “q_10_01.qcp” and converts it to a DIMACS format file called “q_10_01.cnf ”

4. Feed the DIMACS file to the SAT solver. For example

```
$       ./glucose < q_10_01.cnf > q_10_01.out
```

where “q_10_01.out ” contains all the standard output text.

Note: If the files still can't run, try to compile parsers/sat solver.

Compilation instructions:

To compile parser files with g++, open Linux terminal and type

```
$       g++ ps3.cpp -o qcp2cnf -std=c++11
$       g++ ps3b.cpp -o qcp2cnf -std=c++11
$       g++ ps3c.cpp -o qcp2cnf -std=c++11
```

To compile single core version of glucose. In /asn3/glucose-syrup/simp, type

```
$       make -rm
```