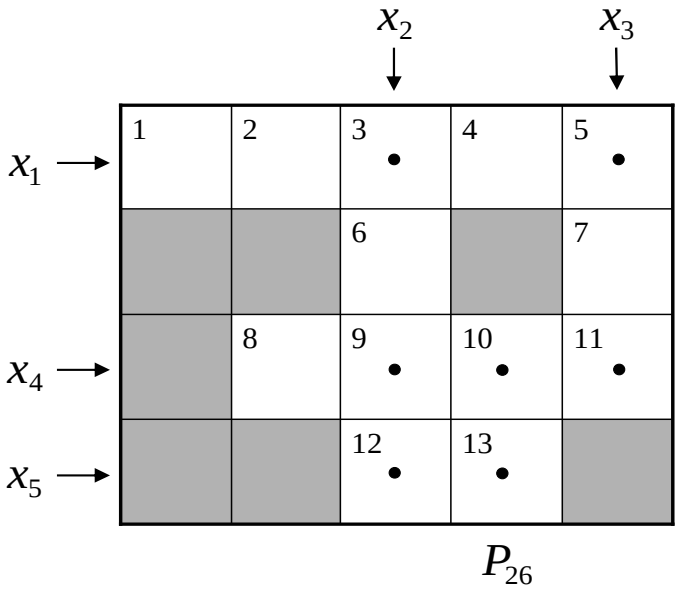
Rina Dechter, Winter-2016

ICS 275, Assignment 4

1. (5 pts.) Listen to the "SAT solving revolution " http://slideslive.com/38894537/the-sat-revolution-solving-sampling-and-counting (link also on the class page, week 3) and provide 1 paragraph summarizing the talk.

2. (20 pts. question 4 chapter 5) Consider The crossword puzzle formulated using the dual-graph representation. Using the ordering $x_1, x_2, x_5, x_6, x_4, x_3$ show a trace, (and data structure) whose length is limited by a 1 page description, for each algorithm.



   (a) Forward-checking

   (b) Using graph parameters, bound the complexity of partial look-ahead on constraint problems whose graph is identical to the crossword puzzle's graph.

3. (30 pts. question 7, chapter 5) Apply the following algorithms to the 6-queens problem. Show the search space and data structures generated for finding all solutions (no more than one page per algorithm).

   (a) Backtracking

   (b) Dynamic variable ordering, DVFC.

   (c) Arc-consistency look-ahead.

4. (15 pts. question 6, chapter 5) Analyze the complexity overhead of each of the following algorithms along a path of the search tree: backtracking, Forward-checking, partial look-ahead, arc-consistency look-ahead (assume that you keep a table of current domains at each level of the search tree.) Bound the complexity of node generation and analyze the complexity of generating a path in the search tree.

5. (10 pts. question 9, chapter 5) We say that a problem is backtrack-free at level m of the search if a backtracking algorithm, once it reaches this level is guaranteed to be backtrack-free. Assume we apply DPC as a look-ahead propagation at each level of the search.

   (a) Can you identify a level for which DPC is backtrack-free?

   (b) Can you bound the complexity of DPC as a function of a width-2 cutset (a cutset is a width-2 if its removal from the graph yields a graph having induced-width of 2.)

6. (extra credit 10 pts. question 10, chapter 5) Describe algorithm DPLL augmented with a look-ahead scheme that does bounded resolution such that any two clauses creating a clause whose size is at most 2, are carried out at each step.

7. (30 pts.) Langford Problem: L(k,n): Arrange k copies of the n digits $1, \ldots, n$ such that there is one digit between the 1s, two digits between the 2s, three digits between the 3s and so on. For example, the solution of $L(2,3)$ is 231213 and the solution of $L(2,4)$ is 23421314. n is also called as the order of the Langford problem.

   (a) (3 pts) Provide one way of encoding the Langford problem as a CSP.

   (b) (5 pts) Suggest a scheme to transform your CSP encoding into a SAT formula.

   (c) (5 pts) What are the pros and cons of modeling the Langford problem as a SAT/CSP.

   (d) (15 pts) SAT solvers take as input a SAT formula in a special format known as the "CNF format". A description of this format is provided in the appendix. Consider a special case of the Langford problem where k=2 i.e. the L(2,n) problem. Write a program that takes n as input and outputs a SAT encoding of L(2,n) in the CNF format. You are free to use any one of the two SAT encodings that you derived in response to part (b). Your program should have the following input/output format:

   langford n cnf-file
   where

   i. langford is the name of the executable
   ii. n is the order of the Langford problem
   iii. cnf-file is the path of the (output) cnf-file

   For part (d), submit the code for your program written in Python (or another language of your choice) to the EEE dropbox. Write a readme file on how to run your code.

A CNF format (Adapted from the DIMACS description of CNF format)

A satisfiability problem in conjunctive normal form consists of a the conjunction of a number of *clauses*, where is clause is a disjunction of a number of variables or their negations. If we let x_i represent variables that can assume only the values *true* or *false*, then a sample formula in conjunctive normal form would be

$$(x_1 \vee x_3 \vee \bar{x_4}) \wedge (x_4) \wedge (x_2 \vee \bar{x_3})$$

where $\vee$ represents the *or* boolean connective, $\wedge$ represents *and* and $\bar{x_i}$ is the negation of x_i.

Given a set of clauses $C_1, C_2, \ldots, C_m$ on the variables $x_1, x_2, \ldots, x_n$, the satisfiability problem is to determine if the formula

$$C_1 \wedge C_2 \wedge \ldots \wedge C_m$$

is satisfiable. That is, is there an assignment of values to the variables so that the above formula evaluates to *true*. Clearly, this requires that each C_j evaluate to *true*.

To represent an instance of such problems, we will create an input file that contains all of the information needed to define a satisfiability problem. This file will be an ASCII file consisting of a two major sections: the preamble and the clauses.

The Preamble. The preamble contains information about the instance. This information is contained in lines. Each line begins with a single character (followed by a space) that determines the type of line. These types are as follows:

- **Comments.** Comment lines give human-readable information about the file and are ignored by programs. Comment lines appear at the beginning of the preamble. Each comment line begins with a lower-case character **c**.

  ```
  c This is an example of a comment line.
  ```

- **Problem line.** There is one problem line per input file. The problem line must appear before any node or arc descriptor lines. For cnf instances, the problem line has the following format.

  ```
  p FORMAT VARIABLES CLAUSES
  ```

  The lower-case character `p` signifies that this is the problem line. The `FORMAT` field allows programs to determine the format that will be expected, and should contain the word "cnf". The `VARIABLES` field contains an integer value specifying n, the number of variables in the instance. The `CLAUSES` field contains an integer value specifying m, the number of clauses in the instance. This line must occur as the last line of the preamble.

The Clauses. The clauses appear immediately after the problem line. The variables are assumed to be numbered from 1 up to n. It is not necessary that every variable appear in an instance. Each clause will be represented by a sequence of numbers, each separated by a newline character. The non–negated version of a variable i is represented by i; the negated version is represented by $-i$.

Each clauses is terminated by the value 0.

Example. Using the example

$$(x_1 \vee x_3 \vee \bar{x_4}) \wedge (x_4) \wedge (x_2 \vee \bar{x_3})$$

a possible input file would be

```
c Example CNF format file
c
p cnf 4 3
1 3 -4 0
4 0
2 -3 0
```