Finite Domain Satisfiablity Solver

Note: This solver corrects and improves the solver created by Hemal A. Lal. See: http://www.d.umn.edu/~lalx0004/research/. Currently it implements Non-Chronological backtracking with resolution-based learning. You can find the description of the algorithms in the documentation folder.

Building the Solver

You can easily create executable on Linux using g++ compiler in the following way:

```
cd ~/your-path-to/mvl-solver/src
```

```
make
```

Alternatively download the executable `Solver` for Linux from the current folder. Below follows the description of the input the solver accepts as well as description of how to use the program to generate a benchmark problem, convert Boolean problem to Finite-domain problem or solve a finite-domain problem in DIMACS CNF format.

Extended DIMACS format

The solver accepts the problems in extended DIMACS CNF format, which is an extension of the standard DIMACS CNF format commonly used for Boolean SAT problems. There are 4 possible lines in a DIMACS file:

1. Comment line: This line contains comments and can be ignored.

```
c This is a comment line
```

2. Problem line: This line contains information about the problem. It begins with a p. There is exactly one such line for each problem.

```
p cnf <NumVar> <NumClause>
```

where NumVar is the total number of variables in the problem, and NumClause is the number of clauses in the problem.

3. Domain line: This line contains information about the domain size of a variable. It begins with a d and is followed by the variable and then by the domain size.

`d <VarName> <DomainSize>`

where VarName is the variable name, and DomainSize is the size of the domain of the variable. There should be at most one domain life for each variable.

4. Clause line: each literal is of the form `<VarName>=<DomainValue>` or `<VarName>!=<DomainValue>`. Each clause ends with a 0, which is used as an end-marker, and the variables are represented by numbers from 1 to N where N is the total number of variables in the theory.

EXAMPLE

```
c This is a pigeonhole problem with 3 pigeons and 2 holes
p cnf 3 5
d 1 2
d 2 2
d 3 2
2!=0 1!=0 0
3!=0 1!=0 0
3!=0 2!=0 0
2!=1 1!=1 0
3!=1 1!=1 0
3!=1 2!=1 0
```

Generating Benchmark Problem

If you want to generate a random benchmark problem, use the following format to run the program:

```
exe -genben -var <int> -clause <int> -clausesize <int> -sat <1/0>
-domain <int> -drand <1/0> -bool <1/0> -file <string>
```

where :

```
exe               | * name of executable

  -genben         | * option stating create benchmark problem
  -var            | * number of variables in benchmark problem
  -clause         | * number of clauses in benchmark problem
  -clausesize     |   number of atoms in each clause, [DEFAULT : 3]
  -sat            |   states whether this benchmark problem is satisfiable or not
                      [possible value : 1/0] [default : 1/true]
  -domain         |   states the domain size of each variables, [DEFAULT : 10]
  -drand          |   states whether to assign domain size of each variable randomly
                      [possible value : 1/0] [DEFAULT : 0/false]
```

```
 -bool           |   states whether the file is in boolean cnf (0) or modified cnf (1)
 -file           | * name of the output file

* - required fields
```

EXAMPLE: `./Solver -genben -var 4 -clause 18 -clausesize 2 -sat 0 -domain 2 -bool 1 -file "finite.txt"`

Finite Domain Solver with Non-Chronological Backtracking

Use the following format to run the program. The solver accepts problems in extended DIMACS format.

`exe -solvenc -var <int> -clause <int> -file <string> -time <int>`

where :

```
 exe             : * name of executable
 -solvenc       : * option stating to solve the finite domain problem
 -var            : * number of variables in benchmark problem
 -clause         : * number of clauses in benchmark problem
 -file           : * name of the input file
 -time           : amount of time allowed for solver to run (in seconds)

* - required fields
```

EXAMPLE: `./Solver -solvenc -var 4 -clause 18 -file "example.txt"`

Convert Boolean to Finite Domain

Use the following format to run the program. The solver accepts problems in DIMACS format.

`./Solver -b2f -file <string> -model <string>` where :

```
exe             : * name of executable
-b2f            : * option stating to convert file
-file           : * name of the boolean file
-model          : * name of the finite file

* - required fields
```

Convert Finite Domain to Boolean : Linear Encoding

Use the following format to run the program:

```
exe -linenc -file <string> -model <string>
```

where :

```
exe             : * name of executable
-linenc         : * option stating to convert file
-file           : * name of the boolean file
-model          : * name of the finite file

* - required fields
```

Convert Finite Domain to Boolean : Quadratic Encoding

Use the following format to run the program:

```
exe -quadenc -file <string> -model <string>
```

where :

```
exe             : * name of executable
-quadenc        : * option stating to convert file
-file           : * name of the boolean file
-model          : * name of the finite file

* - required fields
```