Assignment

Before you start with this assignment you must understand how search works. Read chapter 5.1.1 and 5.3 carefully. You must understand when and why a variable is removed from the list passed to DFS() in Figure 5.1.

The file `SimpleDFS.java` contains a simple but fully functional depth first search (DFS). It implements the basic functionality of DFS and Branch-and-Bound for minimization. The search is basic, it selects variables in the same order as they are stored in the vector, and it always adds a constrain that binds the variable to a singel value from its domain, starting with the smallest. Read the code in `SimpleDFS.java`. Where is the variable selected, where is a value selected, where is the constraint imposed in the store?

You can try the search on the provided example, `Golomb.java`. It is minimization example and it is described, for example hereLinks to an external site..

In this assignment you have to change the behavior of this search. You are to implement two variants of split search. You are to selects a variable, `x`, based on input order, as it is done in SimpleDFS, but then narrows the domain of the variable instead of assigning a single value to it. You split the domain around its middle point, `c`, that is `c = (x.min() + x.max())/2`.

The two search strategies do the following selections:

1. The first search strategy makes a choice point which first tries the lower half of the selected variable `x`:
    - first choice $x \leq c$, and if this fails
    - second choice: $x > c$, the negation of $x \leq c$
2. The second search strategy first tries the upper half of the selected variable `x`:
    - first choice $x \geq c$, and if this fails
    - second choice: $x < c$, the negation of $x \geq c$.

In this assignment you will replace parts of the search implementation in JaCoP. You will write java code. Minizink IDE or the minizink language is not be used in this assignment.

Task 1: Set up the development environment. You can use eclipse, or any other editor/IDE for writing the java code. The initial setup consists of three files: SimpleDFS.java Download SimpleDFS.java, Golomb.java Download Golomb.java, and jacob.jar. The main method is in Golomb.java. It uses classes in jacob.jar to build the model and SimpleDFS to execute a search, so alla three files must be in the class path. Set up the environment and makes sure you can run Golomb.main() before you make any changes to the files.

Task 2: Study SimpleDFS.java in detail. Compare it to to Figure 5.1 in the book. Witch part of the code is added to support minimisation? What is the responsibility of the class ChoisePoint? Does levelDown() mean you go up or down the search tree?

Task 3: implement search strategy 1. Copy the SimpleDFS.java file and name the copy SplitSearch.java. Change the class name to SplitSearch and do all other changes needed for strategy 1 in the file. Edit Golomb.java line 172/173 to use the SplitSearch class. Compare the solution generated by SimpleDFS with the one generated by your implementation to verify a correct implementation.
If you are concerned about your results: yes, indomain min, which SimpleDFS implements is faster than split search for the golomb problem. A fast solver is not among the learning outcome of this assignment.

Task 4: implement search strategy 2.

Task 5: Experiment with different variable selection methods and select the best method for the golomb example. Report the following statistics for the search:

- total number of search nodes
- number of wrong decisions

You should try at least the original SimpleDFS.java<input_order, indomain_min>, strategy 1 from task 2, strategy 2 from task 3, and one more provided by JaCoP.