by Patrick
August 8th, 2019

weeSeepy: an introduction

Introduction

weeSeepy is a small (wee) constraint programming (CP) toolkit, written in java. It is very much choco in style [2] and is inspired by the recent development of miniCP [1]. It is being developed as an aid to teaching CP in the classroom, to support final year project students (where a project might extend the library of constraints, variable ordering heuristics, search algorithms, i.e. extend the modelling capability) and as always to allow a marriage between teaching and research.

A small example (GCol3)

Below, Listing 1, is a small weeSeepy model for the three colouring of the graph with 5 vertices and 6 edges shown in Figure 1. The model is *Problem*-based. That is, a Problem might be thought of as a place holder for the variables and constraints of the model. In line 6 a problem, named pb, is created and it's given a name.

```
1  import java.util.*;
2
3  public class Col3 {
4
5      public static void main(String[] args){
6          Problem pb = new Problem("Col3");
7
8          int n = 5;
9          int k = 3;
10
11         IntVar[] v = new IntVar[n];
12
13         for (int i=0;i<n;i++) v[i] = pb.intVar("v_" + i,1,k);
14
15         pb.post(new NotEquals(pb,v[0],v[1]));
16         pb.post(new NotEquals(pb,v[0],v[2]));
17         pb.post(new NotEquals(pb,v[1],v[2]));
18         pb.post(new NotEquals(pb,v[1],v[4]));
19         pb.post(new NotEquals(pb,v[2],v[4]));
20         pb.post(new NotEquals(pb,v[3],v[4]));
21
22         pb.trace = true;
23         pb.show();
24         System.out.println(pb.bt());
25         pb.show();
26     }
27
28 }
```

Listing 1: weeSeepy code to three colour the simple graph in Figure 1

IntVar is a constrained integer variable. An array of IntVar is declared in line 11 and in line 13 the actual constrained integer varaibles are created. Five (n=5 in line 8) constrained integer variables are created, each with a domain of {1..3} (k=3 in line 9). Lines 15 to 20 post constraints for the six edges in the graph. In line 22 we enable the trace facility and in line 23 show the problem, i.e. list the variables and constraints that make up our model. In line 24 the problem is solved via the call pb.bt(), that is

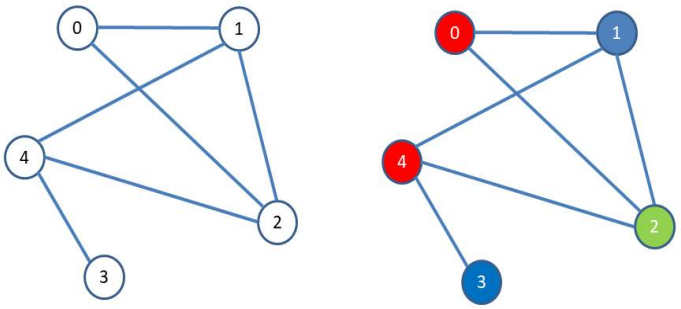

Figure 1: A simple graph and its 3 colouring.

a solution is found via a combination of constraint propagation and chronological backtracking search. The call to pb.bt() delivers true if a solution is found, false otherwise. In line 25 we again show the problem after solving.

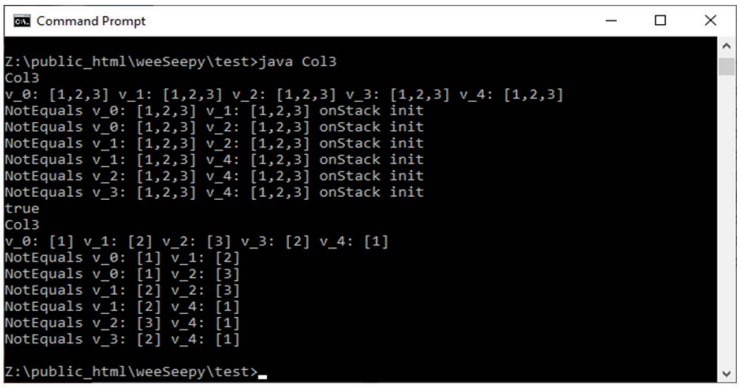

Figure 2: Command line output of Col3.java.

Figure 2 shows the (Window$) command line output of our program. After solving the variables v_0 to v_4 are instantiated to the values 1 (red), 2 (blue), 3 (green), 2 (blue) and 1 (red) respectively.

Another small example (TestEnum)

Listing 2 is an example that shows a global constraint, the max constraint, and how we enumerate solutions. In line 4 we create the problem pb, and in lines 8 to 11 create the constrained integer variable (IntVar) x with domain $\{4, 5\}$ and the array of constrained integer variables v[0], v[1] and v[2] each with domain {0..9}.

In line 13 we create and post the Max constraint into the problem, such that the variable x is equal to the maximum value amongst the variables v[0], v[1] and v[2]. Lines 15 to 19 enumerate and output all solutions to this problem. In more detail, in line 15 the call pb.btProbe() uses a depth first backtracking

```
1  public class TestEnum {
2
3      public static void main(String[] args){
4          Problem pb = new Problem("TestEnum");
5
6          int n = 3;
7
8          IntVar x = pb.intVar("x",4,5);
9          IntVar[] v = new IntVar[n];
10
11         for (int i=0;i<n;i++) v[i] = pb.intVar("v_"+ i,0,9);
12
13         pb.post(new Max(pb,x,v));
14
15         while (pb.btProbe()){
16             System.out.print(x.getValue());
17             for (IntVar y : v) System.out.print(" "+ y.getValue());
18             System.out.println();
19         }
20     }
21 }
```

Listing 2: weeSeepy code to show the Max constraint and enumeration of solutions.

search to find a solution and delivers true if one is found, false otherwise. This solution (if found) is then memoized and acts as a nogood on the next call to btProbe, forcing the search to find the next solution. Line 17 shows how we get hold of, and output, the value of a constrained integer variable. Similarly, line 17 shows output for each of constrained integer variables in the array. More will be said about the Max constraint later, but for now we will report only that there are 152 solutions to this problem, the first output is 4004, then 4014 all the way to 5555.

Outroduction

Choco4, miniCP and weeSeepy share the same look and feel. Maybe this is no surprise. There is a lot of development and testing needing done for weeSeepy (and I believe, also for miniCP). Nevertheless, there will be a number of notes following this one, describing the engineering decisions made and how these have been implemented.

References

[1] Laurent Michel, Pierre Schaus, Pascal Van Hentenryck. MiniCP: A lightweight solver for constraint programming, 2018. Available from `https://minicp.bitbucket.io`.

[2] C. Prud'homme, J.-G. Fages, and X. Lorca. Choco documentation, 2017.