A Note on CSPLIB prob007

H. Simonis
N. Beldiceanu

COSYTEC SA
4, rue Jean Rostand
F-91893 Orsay Cedex
France
{simonis,beldiceanu}@cosytec.com

Abstract

In this note we present a CHIP program to solve the all-interval series problem, presented as problem prob007 in the CSPLIB. We show that one solution can be easily constructed for any problem size, or found without backtracking by a simple search strategy. We then present results on finding all solutions for problem sizes 6,8,10 and 12. Two different constraint models show the advantage of using global constraints for solving this problem. The problem was presented originally as a difficult problem for stochastic local search methods, we show that the CHIP model allows a simple and relatively efficient solution of this problem.

Problem

The problem was first presented in [Hoo98], and then proposed to the CSPLIB problem library (http://dream.dai.ed.ac.uk/group/csplib) as problem prob007. It can be expressed the following way:
We search for a permutation

$[X_1, X_2, .., X_n]$

of the numbers 1..n such that the list

$[|X_1-X_2|, |X_2-X_3|,...,|X_{n-1}-X_n|]$

is a permutation of [1,2,...,n-1]. This all-interval series can be defined for all sizes n. For n=12, the problem corresponds to arranging the half-notes of a scale in such a way that all musical intervals (minor second to major seventh) are covered.
The problem was presented in [Hoo98] as a benchmark for comparing different stochastic local search methods, by translating a CSP formulation into a 3-SAT problem. Results for different methods were disappointing, not all methods were able to find solutions for the problem size 12.

Program

We can express the problem easily in CHIP with the following program. The problem is expressed by N variables with a domain 1..N. These variables must be alldifferent. The absolute distance between two variables is expressed using a minimum, maximum and equality constraint for each pair of variables. We use the identity

|X-Y| = max(X,Y)-min(X,Y)

These constraints generate a set of distance variables. We impose the constraint that all these variables must be different with a second alldifferent constraint. A standard search method labeling/4 is used to find values.

```
run(N,L):-
     length(L,N),
     L :: 1..N,
     alldifferent(L),
     differences(L,K,N),
     alldifferent(K),
     labeling(L,0,input_order,indomain).

differences([_],[],_).
differences([A,B|R],[X|T],N):-
     difference(A,B,X,N),
     differences([B|R],T,N).

difference(A,B,X,N):-
     [Min,Max,X] :: 1..N,
     minimum(Min,[A,B]),
     maximum(Max,[A,B]),
     Max #= Min+X.
```

The CHIP built-in alldifferent only performs syntactic constraint propagation with forward-checking. A more powerful propagation can be achieved using the cycle constraint [BC94] to express that the set of values form a permutation. The program is changed to

```
run(N,L):-
     length(L,N),
     L :: 1..N,
     A :: 1..N, cycle(A,L),
     differences(L,K,N),
     N1 is N-1,
     B :: 1..N1, cycle(B,K),
     labeling(L,0,input_order,indomain).
```

The constraint propagation of the cycle constraint uses a bi-partite matching algorithm between the variables and the possible values. This propagation can detect failure much earlier or enforce necessary conditions in the assignment.
In the evaluation below we will use both variants.

Finding one solution

The program above can be used to find a solution of the problem. But it is easy to see that a solution for any size N can be constructed by the sequences

[N,1,N-1,2,N-2,3,...,N/2+2,N/2-1,N/2+1,N/2] with the differences [N-1,N-2,...2,1] for even N and

[N,1,N-1,2,N-2,3,...,$\left\lfloor \frac{N}{2} \right\rfloor + 2, \left\lfloor \frac{N}{2} \right\rfloor, \left\lfloor \frac{N}{2} \right\rfloor + 1$] with the differences [N-1,N-2,...2,1] for odd N.

This sequence can be generated in the above program by changing the labeling strategy to

```
run(N,L):-
     length(L,N),
     L :: 1..N,
     alldifferent(L),
     differences(L,K,N),
     alldifferent(K),
     labeling(K,0,input_order,large),
     labeling(L,0,input_order,large).

large(X):-
     indomain(X,max).
```

This program will first instantiate the distance variables to the values N-1,N-2,...,1. After setting the first variable in L to the value N, all other variables will be instantiated by constraint propagation. Note that this program does not backtrack to find a first solution. A second solution starting with the value 1 is found as the first alternative.
This construction shows that finding one solution to the problem is not difficult and does not require any search.

Finding all solutions

A more challenging problem is to find all solutions for a given problem size. We have run the program for sizes 6,8,10 and 12 using our two program variants, one with the syntactic alldifferent constraint, the other with the global cycle constraint. The labeling strategy was selecting the variables in the input_order, using the standard indomain value selection, starting with the smallest value in the domain. The following results are obtained (CHIP V5.2 on Pentium MMX 233 MHz, Windows NT):

Size	Backtracking Steps	Execution Time	Solutions
6	162	0.24 s	24
8	1698	1.2 s	40
10	38978	27 s	296
12	1047274	803 s	1328

Table1: Results with alldifferent constraint

Size	Backtracking Steps	Execution Time	Solutions
6	124	0.25 s	24
8	538	1.1 s	40
10	6530	15 s	296
12	109242	311 s	1328

Table 2: Results with cycle constraint

The results show a significant improvement of the second program for larger problem sizes. For the smallest size 6, both programs have comparable results for backtracking steps and execution times. For the problem size 12, the cycle constraints reduces the number of backtracking steps by nearly a factor of ten, and the execution time by a factor of 2.5.
The next table shows the weak impact of the variable selection strategy in the labeling method. We have run the cycle based program for problem size 12 with the variable selection methods input_order, most_constrained and first_fail. Only relatively small variations of the results were observed.

Size 12	Backtracking Steps	Execution Time	Solutions
input_order	109242	315 s	1328
most_constrained	112135	325 s	1328
first_fail	108994	312 s	1328

Table 3: Results for cycle constraint with different variable selection methods

Improvements

The program could be significantly improved by using a specific constraint for the absolute distance calculation. Such a constraint is not provided as a built-in in CHIP, but could be easily added. If full lookahead is used for the distance constraint, further improvements in the constraint propagation are possible and the number of backtracking steps should be reduced.

Summary

In this note we have presented a CHIP program to solve the problem prob007 of the CSPLIB. We show that for any problem size one solution can be constructed systematically and generated without any backtracking. Finding all solutions for the problem sizes 6,8,10 and 12 is quite efficient with a simple CHIP program. The use of a global constraint to express the permutation conditions in the problem significantly reduces the execution time and the number of backtracking steps.

Bibliography

[BC 94] N. Beldiceanu, E. Contejean
Introducing Global Constraints in CHIP
Journal of Mathematical and Computer Modelling, Vol 20, No 12, pp 97-123, 1994

[Hoo98] H. Hoos
Stochastic Local Search | Methods, Models, Applications
PhD thesis, Computer Science Department
Technical University of Darmstadt, Germany, 1998