Proc. SymCon'03

Conditional Symmetry in the All-Interval Series Problem

Ian P. Gent[1], Iain McDonald[1], and Barbara M. Smith[2]

[1] School of Computer Science, University of St Andrews,
St Andrews,Fife, KY16 9SS, UK
{ipg,iain}@dcs.st-and.ac.uk
[2] School of Computing and Engineering, University of Huddersfield
Huddersfield HD1 3DH, UK
b.m.smith@hud.ac.uk

Abstract. We discuss the notion of a "conditional symmetry", which we define as a symmetry which is only valid subject to some condition which can be tested during search. We use the all-interval series problem as our running example, and describe a set of conditional symmetries that arise in it. We show two different ways to use SBDS, unchanged, to break conditional symmetries. Unfortunately, both techniques give very bad results on the all-interval series. Finally, we present a reformulation of the all-interval series which eliminates all standard and conditional symmetries. We show that this encoding allows dramatically improved search, by a factor of 50 in runtime on the best existing technique.

1 Conditional Symmetries

When we consider symmetries in CSPs, we normally think of a bijective function which is fully known before search starts. But not all symmetries are like this. We can have *conditional symmetries*, i.e. symmetries which only come into play if some condition is satisfied during search. For example, in [7] we are shown a method of breaking symmetries between identical objects in planning problems. Similarly in [3], methods of dealing with symmetries that emerge as a result of planning decisions are presented.

Definition 1. *Consider a set of constraints θ, and a bijective mapping $g_\theta : A \rightarrow A$ where A is a (partial) assignment. If g_θ requires θ to be true before it becomes a symmetry then g_θ is said to be a* conditional symmetry.

For any given CSP which has conditional symmetries acting on it, there will be a set Θ of pre-conditions, where each pre-condition $\theta \in \Theta$ will have an associated bijective mapping g_θ. While there may be many pre-conditions, some of them could be mutually exclusive. For instance θ_i might insist that $var_a = val_1$ and θ_j might insist that $var_a = val_2$. In this case, at most one of the mappings g_{θ_i} and g_{θ_j} can be a symmetry.

Since conditional symmetries only act in certain parts of the search space, we need to pay careful attention to how we break these symmetries. We may need to perform

some no-good recording as was done in [2][1]. However, it is clear that a simple adaptation of SBDS will work. After the first term, the following constraint is the standard SBDS constraint [4], while the first term asserts the condition holds at the relevant point in the search space.

$$\theta \wedge A \wedge g_\theta(A) \wedge (var \neq val) \Rightarrow g_\theta(var \neq val) \quad (1)$$

2 The All-Interval Series Problem

The all-interval series problem is to find a permutation of the n integers from 0 to $n-1$ such that the differences between adjacent numbers are also a permutation, of the numbers from 1 to $n-1$. It is problem Prob007 in CSPLib [6]. It is a simple example of the graceful graphs problem [8] in which the graph is a line.

We can model this using n integer variables $x_0, x_1, ..., x_{n-1}$ where x_i represents the number in position i in the permutation. Following Choi and Lee [1], we use auxiliary variables $d_i = |x_i - x_{i+1}|$ for $0 \leq i \leq n-2$ to represent the differences between adjacent numbers. The variables $d_0, ..., d_{n-2}$ are required to be all different.

Since the variables x_i are required to form a permutation of the numbers 0 to $n-1$, we can introduce dual variables y_j, $0 \leq j \leq n-1$, linked to the x_i variables by channelling constraints (implemented by IlcInverse in Solver). The variable y_j represents the number j, and the value assigned to the variables specifies the position of this number in the permutation. This in itself is sufficient to ensure that the values assigned to the x_i variables are all different. (However, we still include the all different constraint on the x variables.) Similarly we introduce dual p_j variables for the differences, so that $p_j = i \equiv d_i = j$. Choi and Lee also introduce an implied constraint on the y_j variables: since the difference $n-1$ must occur, and can only arise by putting the numbers 0 and $n-1$ next to each other, we must have that $|y_0 - y_{n-1}| = 1$. This is the only constraint we have used from a potentially complete model based on the y_j variables: Choi & Lee used the complete model, in combination with a model using the x_i and d_i variables, but we believe that the additional constraint propagation that they found is due (mainly or solely) to this implied constraint. We do not enforce generalized arc consistency on the allDifferent constraint on the d_i variables; while doing so reduces the number of fails slightly we did not note a reduction in running time compared to either an allDifferent constraint treated as binary $\neq$ constraints, or channelling constraints between the d_i variables and the p_j variables. Throughout this paper we use the lexicographic ordering heuristic: we don't see significantly improved performance using other heuristics.

There are 4 obvious symmetries in the problem: the identity, reversing the series, negating each element by subtracting it from $n-1$, and doing both. Puget and Régin, in their note in CSPLib [9], identify just the negation symmetry and eliminate it by adding the constraint $x_0 < x_1$. However, since there are only three symmetries other than identity, we can easily use SBDS [4], and define the three functions necessary.

Table 1 shows the results of using our model in Ilog Solver 5.2. The experiments were run on a Intel Celeron 1.7Ghz with 128MB RAM running Redhat Linux version

[1] It should be noted that this paper ends with, "We are currently extending [the work done in this paper to] ideas in 'quasi-symmetric' problems"

	No Symmetry Breaking				SBDS			
n	Solutions	Fails	Choice Points	Cpu (sec)	Solutions	Fails	Choice Points	Cpu (sec)
3	4	0	3	< 0.01	1	1	1	< 0.01
4	4	4	7	< 0.01	1	3	3	< 0.01
5	8	11	18	< 0.01	2	4	5	0.01
6	24	30	53	0.01	6	12	17	< 0.01
7	32	132	163	0.02	8	42	49	< 0.01
8	40	520	559	0.04	10	173	182	0.01
9	120	1906	2025	0.17	30	626	655	0.09
10	296	7853	8148	0.72	74	2675	2748	0.27
11	648	34200	34847	3.44	162	11665	11826	1.23
12	1328	159687	161014	15.34	332	55909	56240	6.96
13	3200	784823	788022	73.46	800	276499	277298	34.00
14	9912	4127283	4137194	410.91	2478	1486241	1488718	184.15

Table 1. Solving the all interval series problem with and without SBDS.

9.0. We give results with and without symmetry breaking using SBDS. As we now expect where there are a small number of symmetries, we get a significant win by using SBDS, the three symmetries achieving a run time improvement of more than a factor of two for $n \geq 10$. Taking into account cpu speed, these running times are worse than Puget and Régin's: at $n = 14$ they report a run time of 199.6s on a Pentium 800MHz. Our run times are significantly better than Simonis and Beldiceanu's [10], who report 311s for the $n = 12$ problem on a 233MHz machine, compared to our 15.34s and 6.96s without and with SBDS.

As well as SBDS, we investigated the use of constraints to break the symmetry. All symmetry can be broken in a number of ways, but the following method was the most effective we tested, and also closely related to the formulation presented later in the paper. We simply insist that the sequence $0, n-1, 1$ occurs, in that order, in the solution. If not already in this form, any all-interval series can be converted to one with this property by negation, reversal, or both.[2] Since we have the dual variables y_j, this constraint can be simply added by stating $y_{n-1} - y_0 = 1, y_1 - y_{n-1} = 1$. Table 2 shows results using this model. (Results for SBDS are repeated from Table 1 for comparison.) We see that as intended we return the same number of solutions as with SBDS, but run times are twice as fast or more. While exact comparisons are impossible this method seems close in run time with Puget and Régin's. We will show later that we can make dramatic improvements on these runtimes.

3 Conditional Symmetry in All-Interval Series

We have identified a conditional symmetry in the all-interval series problem. We can cycle a solution to the all interval series problem about a pivot to generate another

[2] In fact in general insisting that 1 and $n-1$ are adjacent breaks the negation symmetry (sometimes called complement symmetry) in any graceful graph.

	Symmetry Breaking Constraints				SBDS			
n	Solutions	Fails	Choice Points	Cpu (sec)	Solutions	Fails	Choice Points	Cpu (sec)
3	2	0	1	< 0.01	1	1	1	< 0.01
4	1	1	1	< 0.01	1	3	3	< 0.01
5	2	3	4	< 0.01	2	4	5	0.01
6	6	6	11	< 0.01	6	12	17	< 0.01
7	8	27	34	< 0.01	8	42	49	< 0.01
8	10	107	116	0.01	10	173	182	0.01
9	30	363	392	0.04	30	626	655	0.09
10	74	1382	1455	0.16	74	2675	2748	0.27
11	162	5544	5705	0.71	162	11665	11826	1.23
12	332	24308	24639	2.71	332	55909	56240	6.96
13	800	112822	113621	13.24	800	276499	277298	34.00
14	2478	556648	559125	68.19	2478	1486241	1488718	184.15

Table 2. Breaking symmetry in the all-interval series problem with constraints and with SBDS.

solution. The location of this pivot is dependent on the assignments made and so these symmetries are conditional. Here is a solution for $n = 11$. Differences are written underneath the numbers:

```
0 10 1 9 2 8 3 7 4 6 5
 10 9 8 7 6 5 4 3 2 1
```

The difference between the first number (0) and last number (5) is 5. This means we can split the sequence between the 8 and 3, losing the difference 5. We can join the rest of the sequence on to the start, because the $5-0$ will now replace $8-3$. This yields the solution:

```
3 7 4 6 5 0 10 1 9 2 8
 4 3 2 1 5 10 9 8 7 6
```

In this case the pivot is between the values 8 and 3. For this specific conditional symmetry of pivoting between x_5 and x_6 we have:

$$\theta = \{|x_0 - x_{10}| = |x_5 - x_6|\}$$
$$g_\theta = \{x_0 \mapsto x_5, x_1 \mapsto x_6, x_2 \mapsto x_7, x_3 \mapsto x_8, x_4 \mapsto x_9, x_5 \mapsto x_{10}, x_6 \mapsto x_0,$$
$$x_7 \mapsto x_1, x_8 \mapsto x_2, x_9 \mapsto x_3, x_{10} \mapsto x_4\}$$

The difference between first and last terms must always duplicate a difference in the sequence, so this operation can be applied to any solution. We show below that for $n > 4$ this always gives a new solution, i.e. not equivalent to any of the 4 symmetries acting on the original. This can be combined with the standard symmetries to give 4 conditional symmetries.

4 Breaking Conditional Symmetry Using SBDS

We now show that we can break conditional symmetries in the all interval series problem using SBDS. We have not written a new implementation of (1), but instead reuse our existing implementation of SBDS in Ilog Solver [4]. The basic task of an SBDS function in this implementation is to describe how the symmetry it represents acts on a given assignment. That is, the function is passed $var = val$ and returns $g(var = val)$. Upon backtracking, the SBDS search algorithm rules out the *negation* of the assignments returned by these functions. At a state A in search where the assignment $var = val$ has been proved inconsistent and $g(A)$ stored in a cache, SBDS posts the constraint $g(A) \Rightarrow g(var \neq val)$, the other conditions $A \wedge var \neq val$ being true from context.

For a conditional symmetry g_θ we wish to post $\theta \wedge g_\theta(A) \Rightarrow g_\theta(var \neq val)$. We do this by implementing an SBDS function that returns the pre-condition θ as well as the symmetric equivalent of $var = val$. That is, it returns

$$\theta \wedge g_\theta(var = val)$$

SBDS uses this to post $g_\theta(A) \rightarrow (\neg\theta \vee g_\theta(var \neq val))$, and by simple rearrangement this is equivalent to the intended $\theta \wedge g_\theta(A) \Rightarrow g_\theta(var \neq val)$.

For the all-interval series problem of size n, there are $n-1$ conditional symmetries corresponding to the original pivoting operation, and $4n-4$ in total when the original symmetries of the problem are combined with the conditional symmetries. Using the 5 original conditional symmetries for $n = 6$, we saw only a slight reduction in search which did not repay overheads. This is disappointing as the addition of a single SBDS function often can reduce search by almost a factor of two.

4.1 A better implementation

There are two main disadvantages with our initial implementation of conditional symmetries. Firstly, the functions created are instance specific i.e. different functions are needed for different values of n. This is partly due to the fact that for the all interval series problem of size n, there are a variable number $(n-1)$ of these conditional symmetries. This number also adds to the run-time overheads. Secondly, the pre-condition θ needs to be posted just once for each conditional symmetry. However, our implementation posts the pre-condition θ redundantly as many as $n-1$ times. We now show that for the all-interval series problem, we can eliminate these two disadvantages.

Note that the location of the pivot can be represented as a variable. The value of this variable could then be used to show how far the series needed to be cycled. Thus we included a new variable $pivot$, and then wrote an SBDS function that returned a constraint whose effect depends on $pivot$. We use Solver's ability to index an array by an integer variable. For example, the symmetric equivalent to $vars[i] = j$ is $vars[(i + pivot) \mod n] = j$ under the original conditional symmetry.[3] The advantage of this is that we (i) have an SBDS function that is not instance specific and (ii) have a single SBDS function that breaks all $n-1$ conditional symmetries. However, unlike our first approach, it is not clear if this implementation technique generalises.

[3] For simpler implementation we added $n-1$ variables onto the end of $vars$, constrained so that $vars[i+n] = vars[i]$, and then returned the SBDS constraint as $vars[i + pivot] = j$.

	Conditional Symmetry Breaking				Conditional Symmetry Breaking + SBDS			
n	Solutions	Fails	Choice Points	Cpu (sec)	Solutions	Fails	Choice Points	Cpu (sec)
3	2	1	2	< 0.01	1	1	1	< 0.01
4	2	6	7	< 0.01	1	3	3	< 0.01
5	4	14	17	< 0.01	1	5	5	< 0.01
6	12	40	51	0.02	3	14	16	0.01
7	16	143	158	0.03	4	39	42	< 0.01
8	20	524	543	0.11	5	167	171	0.08
9	60	1908	1967	0.42	15	581	595	0.39
10	148	7798	7945	2.47	37	2451	2487	2.07
11	324	33741	34064	9.6	81	10494	10574	12.44
12	664	157000	157663	51.67	166	50050	50215	81.71
13	1600	772858	774457	340.97	400	244984	245383	500.26
14	4956	4067503	4072458	2355.05	1239	1316655	1317893	3682.79

Table 3. Results with conditional symmetry breaking. This used in conjunction with SBDS breaks all symmetry to get the correct number of unique solutions.

Unfortunately, we still seem to have the same problems with the original implementation in that the pruning seems to take place late and there are significant overheads. Table 3 shows results breaking just the original conditional symmetry, and the 7 non-trivial combinations of a conditional and regular symmetry. While in the full case, we do see a reduction in search and number of solutions, run time increases 20-fold compared to the use of SBDS in Table 1.

5 Reformulating to Eliminate Symmetry

Unfortunately, so far we can only report bad results for conditional symmetry breaking using SBDS. However, in the special case of all-interval series we now show that we can reformulate the problem to eliminate all symmetry including conditional symmetry, and that this gives a 50-fold runtime improvement on the best previous work.

In fact, we do more than reformulate the problem. We present a new problem. From each solution of this problem we can read off eight solutions to the all-interval series problem, corresponding to all combinations of conditional and ordinary symmetries. A curiosity of the new problem is that we actually *introduce* symmetry, in that the sequence can be rotated arbitrarily. However, it is easy to break this symmetry and in doing so we also break the conditional symmetry in the original. In fact, the conditional symmetry in the all-interval series corresponds to a rotation in the new problem. Since we can break all rotations simply, we break the conditional symmetry automatically. Indeed, it is notable that we have eliminated the problem of conditional symmetry in an entirely unconditional way.

The new problem is as follows. Consider a cycle formed by n nodes, with the n differences between consecutive nodes satisfying the constraint that every difference from 1 to $n-1$ appears at least once, and one difference appears exactly twice. From any solution to this we can form two all-interval series, by breaking the cycle at either one of the

repeated differences. For example, `0 6 1 3 2 5 4` is a solution to the new problem. The repeated difference is 1. We can split the sequence between the 3 and 2 or between 5 and 4, giving two all-interval series `2 5 4 0 6 1 3` and `4 0 6 1 3 2 5`. Note that there is no requirement that the repeated difference be between first and last elements.

Now let us consider the symmetry of this new problem. Because we included the difference between first and last elements of the sequence, we introduce new symmetry because we can rotate the cycle. We break this simply by insisting the first element is 0. Next, we note that 0 and $n-1$ must be adjacent, and since we can reverse any sequence, we insist that the second element is $n-1$. Finally, where does the difference $n-2$ come in? It can only be by putting $n-2$ at the end, i.e. before 0 in the cycle, or 1 after $n-1$. But if we put $n-2$ before 0, we would have $n-2, 0, n-1$, which by negation and reversal would be $0, n-1, 1$. So we can insist that the sequence starts $0, n-1, 1$.[4] This makes the reformulated problem in full:

Definition 2 (Reformulation of All-interval series problem). *Given $n \geq 3$, find a vector $(s_0, ..., s_{n-1})$, such that:*

1. *s is a permutation of $\{0, 1, ..., n-1\}$; and*
2. *the interval vector $(|s_1 - s_0|, |s_2 - s_1|, ...|s_{n-1} - s_{n-2}|, |s_{n-1} - s_0|)$ contains every integer in $\{1, 2, ..., n-1\}$ with exactly one integer repeated; and*
3. *$s_0 = 0$, $s_1 = n-1$, $s_2 = 1$.*

In the appendix prove some useful lemmas about this new version of the problem. The two most important ones show that:

- For $n > 4$, there are exactly 8 times as many solutions to the original all-interval series problem as to the reformulated one.
- We can add a derived constraint, that the repeated difference is even if n is congruent to 0 or $1 \mod 4$.

To code this formulation, we simply changed the all different constraint to an IlcDistribute constraint, subject to every difference occurring at least once: the fact that there are n differences automatically encodes that one will appear twice. We experimented with different levels of propagation on the IlcDistribute constraint, and while search was reduced slightly, it did not repay the extra overheads, so we used the minimum. As the differences are no longer all different, we cannot use the dual p variables, and we also omitted the dual y variables as they were no longer cost effective. Finally we added the constraint on the parity of the repeated difference. This did reduce run time by about a third. Again we used the lexicographic heuristic.[5]

Table 4 shows results using the reformulated encoding. For $n > 4$ we correctly obtain half the number of solutions reported by SBDS in Table 1 and an eighth of that with no symmetry breaking, and the same number of solutions as in the right hand side

[4] Thus, in effect we have added the same constraints we tested in Table 2, that the sequence $0, n-1, 1$ occurs.

[5] Our code is actually Solver 4.4 code compiled and run under Solver 5.2.

n	Solutions	Fails	Choice Points	Cpu (sec)	Speedup	Fails/Solution
3	1	0	0	0.01	-	0
4	1	0	0	< 0.01	-	0
5	1	0	0	< 0.01	-	0
6	3	1	3	< 0.01	-	0.33
7	4	1	4	< 0.01	-	0.25
8	5	9	13	< 0.01	-	1.80
9	15	14	28	0.01	9	0.93
10	37	69	105	0.02	13	1.97
11	81	278	358	0.02	61	3.43
12	166	858	1,023	0.06	116	5.17
13	400	3,758	4,157	0.28	121	9.40
14	1,239	19,531	20,769	1.78	103	15.76
15	3,199	91,695	94,893	8.85	-	28.66
16	6,990	389,142	396,131	36.94	-	55.67
17	17,899	2,093,203	2,111,101	215.61	-	116.95
18	63,837	13,447,654	13,511,490	1,508.26	-	212.15
19	181,412	79,270,906	79,452,317	9,778.94	-	436.97
20	437,168	435,374,856	435,812,023	53,431.50	-	995.90

Table 4. Run times for reformulated version of the all-interval series problem. Where meaningful, the column for speedup indicates the factor by which these run times improve those of SBDS in Table 1, on the same machine. The final column indicates the number of fails encountered per solution: from $n = 9$ this almost doubles for each step in n.

of Table 3. Where we have meaningful comparisons, from about $n = 11$ to 14, we seem to have a speedup of about 100 times compared to using SBDS in Table 1. This represents a roughly 50-fold speedup on Puget and Régin's results [9] and our own from Table 2. It is disappointing that the speedup does not improve with n. Given the growth in run time, this corresponds only to an ability to solve $n + 3$ in time similar to solve n using SBDS.[6] Nevertheless, it is clear that this formulation is by a very wide margin the best way to count solutions to the all-interval series problems.

Table 4 shows that the number of fails per solution roughly doubles for each increment in n. Thus, while sometimes regarded as the easiest problem in CSPLib, the all-interval series still seems to involve considerable combinatorial search. However, it remains possible that further ideas will show that the enumeration problem to be easy. One idea we have not explored is to generalise our deduction of the parity of the repeated difference. We have to partition the set of differences (and the repeated difference) into two equal sets: those where $x_{i+1} > x_i$ and those where $x_{i+1} < x_i$. Search in this number partitioning problem can be reduced using modulo arithmetic [5].

[6] Interestingly, this seems to correspond to the fact that three variables are set before search.

6 Conclusions

In this paper we have formally defined "conditional symmetry", a class of symmetry that has been observed previously in CSPs, but not defined in general. Many new questions are raised by this. Most importantly, how do we best deal with conditional symmetries? While we showed that it works, our results suggest that "hacking" SBDS to break conditional symmetries is not a long term solution. It may be possible to modify some symmetry breaking method to cope with conditional symmetry. Once an efficient method of dealing with conditional symmetries has been found, we need to be able to use this research to perform conditional symmetry breaking in planning problems (as already done by Ian Miguel in [7], and by Maria Fox and Derek Long in [3]) and scheduling problems.

We also introduced a reformulated version of the all-interval series problem which improves the state of the art for that problem by a factor of 50. The reasons for this remarkable speedup are not entirely clear, but we can make the following points. First, the reformulation entirely eliminates the conditional symmetries. Second, we introduce n symmetries by creating a cycle, but this and the original symmetries are broken just by setting 3 variables. This is probably the main reason for the speedup, as setting variables is obviously a particularly strong form of symmetry breaking constraints. Third, we get a small speedup from the parity constraint. (This in fact could be added in a different form to the original all-interval series, but we have not tried that.) We would love to be able to generalise the insights that led to a 50-fold improved reformulation, but sadly it seems very special purpose to the particular problem.

Our results show that CSP practitioners should look out for conditional symmetries when modelling. Just as with ordinary symmetries, we have shown that conditional symmetries can be broken in full leading to a reduced number of solutions. While this can be done in a number of ways, in this paper we saw the combination of reformulation and symmetry breaking constraints to be a remarkably effective technique. We conclude by suggesting that the topic of conditional symmetries in CSP is worthy of further study.

Acknowledgements

We have discussed conditional symmetries with many members of APES, whom we thank. We specially thank Evgeny Selensky, Ian Miguel, Graeme Bell, and reviewers for SymCon. This work is partly supported by EPSRC grants GR/R29666 and GR/R29673, by a Royal Society of Edinburgh SEELLD Support Fellowship and by an EPSRC PhD studentship.

References

1. C. Choi and J.H. Lee, *On the pruning behavior of minimal combined models for CSPs.*, Proceedings of the Workshop on Reformulating Constraint Satisfaction Problems (A. Frisch, ed.), 2002, Available from http://www-users.cs.york.ac.uk/~frisch/Reformulation/02/Proceedings/choi.ps.

2. Filippo Focacci and Michaela Milano, *Global cut framework for removing symmetries*, Principles and Practice of Constraint Programming - CP2001 (Toby Walsh, ed.), Springer, 2001, pp. 77–92.
3. Maria Fox and Derek Long, *Extending the exploitation of symmetries in planning*, Artificial Intelligence Planning Systems (Malik Ghallab, Joachim Hertzberg and Paolo Traverso, eds.), 2002, pp. 83-91.
4. I.P. Gent and B.M. Smith, *Symmetry breaking in constraint programming*, Proceedings of ECAI-2000 (W. Horn, ed.), IOS Press, 2000, pp. 599–603.
5. I.P. Gent and T. Walsh, *From approximate to optimal solutions: Constructing pruning and propagation rules*, Proceedings of IJCAI 97, 1997, pp. 1396–1401.
6. I.P. Gent and T. Walsh, *CSPLib: a benchmark library for constraints*, Tech. report, Technical report APES-09-1999, 1999, Available from http://www.csplib.org/. A shorter version appears in the Proceedings of the 5th International Conference on Principles and Practices of Constraint Programming (CP-99).
7. Ian Miguel, *Symmetry-breaking in planning: Schematic constraints*, SymCon'01: Symmetry in Constraints 2001 (Pierre Flener and Justin Pearson, eds.), 2001, pp. 17–24.
8. K.E. Petrie and B.M. Smith, *Symmetry breaking in graceful graphs*, Proc. CP-03 (F. Rossi, ed.), Springer, 2003.
9. J.-F. Puget and J.-C. Régin, *Solving the all-interval problem*, Available from http://www.4c.ucc.ie/~tw/csplib/prob/prob007/puget.pdf.
10. H. Simonis and N. Beldiceanu, *A note on CSPLIB prob007*, Available from http://www.4c.ucc.ie/~tw/csplib/prob/prob007/helmut.pdf.

Appendix: Theoretical Results on Reformulated Encoding

Lemma 1. *If $n > 4$ then the difference $n-2$ is never repeated.*

Proof. The sequence starts $0, n-1, 1$. The only way to repeat the difference $n-2$ is to place $n-2$ last, i.e. next to 0. So the sequence must be $0, n-1, 1, ..., n-2$. But now where can we get the difference $n-3$? Only from placing 1 and $n-2$ next to each other. This is impossible since $n > 4$ and there are terms between 1 and $n-2$.

Lemma 2. *Given a solution to Definition 2, no combinations of the symmetries mentioned earlier, or of rotating the sequence, leads to a distinct solution.*

Proof. In the new formulation, the conditional symmetry simply corresponds to a rotation of the cycle. The constraint $s_1 = 0$ disallows all rotations. Negating the vector places 0 after $n-1$, and no rotation can correct this, so this is made impossible by setting $s_1 = 0, s_2 = n-1$. The same holds for reversing the vector. The only remaining case is the combination of negating the vector and reversing it. Consider $n > 4$. Negation converts $0, n-1, 1, ...$ into $n-1, 0, n-2, ...$, reversal makes it $n-2, 0, n-1, ...$, and rotation to place 0 first makes this $0, n-1, ..., n-2$. But this is impossible for $n > 4$ by Lemma 1. In the exceptional cases of $n = 3$ and $n = 4$, the unique solutions are $0, 2, 1$ and $0, 3, 1, 2$, and the reader can check that the operations of negation, reversal, and anticlockwise rotation by 1, taken together lead to the identical solution.

Lemma 3. *For $n > 4$, there are 8 distinct all-interval series corresponding to any solution of Definition 2.*

Proof. There are two distinct all-interval series corresponding to the reformulated problem in which we do not apply any of the original problem symmetries. These are the ones in which we split the sequence at one or other of the repeated differences. Note that both contain the subsequence $0, n-1, 1$. Doing nothing, reversing, complementing, and complementing and reversing leads to four pairs of solutions containing the subsequences $0, n-1, 1$; $1, n-1, 0$; $n-1, 0, n-2$; and $n-2, 0, n-1$. These subsequences characterise the pairs of solutions: only one can appear in each solution. To see this, we note from Lemma 1 that $n-2$ does not appear next to 0 in the original solution to the reformulated problem since $n > 4$. This means that none of the subsequences can be converted to another by cutting and regluing at the selected difference: this would correspond to a rotation in the solution to the reformulated problem and would only be possible if $n-2$ was next to 0. [7]

Lemma 4. *The repeated difference is even iff n is congruent to 0 or 1 mod 4.*

Proof. The cycle starts with 0 and ends with 0. Therefore the total sum of all the differences in the cycle is even. Without the repeated difference, the sum of all the differences is just $n(n-1)/2$, which is even when n is $4k$ or $4k+1$, and odd otherwise. Therefore the remaining, repeated, difference has to be of the same polarity.

[7] The proof does rely on $n > 4$. For $n = 3, 4$ we can have $n-2$ next to 0, in the solutions 0,2,1 and 0,3,1,2. For $n = 3, 4$, there is a unique solution to this problem and 4 to the original all-interval series.