Modelling the Golomb Ruler Problem

Barbara M. Smith[1], Kostas Stergiou[2], and Toby Walsh[2]

[1] The APES Research Group, School of Computer Studies, University of Leeds, Leeds, United Kingdom. Email: bms@scs.leeds.ac.uk
[2] The APES Research Group, Department of Computer Science, University of Strathclyde, Glasgow, United Kingdom. Email: {ks,tw}@cs.strath.ac.uk

Abstract. The Golomb ruler problem has been proposed as a challenging constraint satisfaction problem. We consider a large number of different models of this problem, both binary and non-binary. The problem can be modelled using quaternary constraints, but in practice using a set of auxiliary variables and ternary constraints gives better results. A binary encoding of the problem gives a smaller search tree, but is impractical because it takes far longer to run. We compare variable ordering heuristics and consider the use of implied constraints to improve propagation. We believe that more case studies such as this are essential to reduce the skill currently required for successful modelling.

1 Introduction

In his AAAI-98 invited talk, Gene Freuder identified modelling as one of the major hurdles preventing the uptake of constraint satisfaction technology. The availability of non-binary constraints can increase the number of possible models of a problem amnd so makes modelling still more difficult. In this paper, we report a case study in modelling a challenging problem. We identify a large number of different models, both binary and non-binary, and compare them theoretically and empirically. We believe that many more studies like this will be needed to turn the art of modelling into a science.

2 The problem

Peter van Beek has proposed the Golomb ruler problem as a challenging constraint satisfaction problem for the CSPLib benchmark library (available as `prob006` at `http://csplib.cs.strath.ac.uk`). The problem specification given there is: "A Golomb ruler may be defined as a set of m integers $0 = x_1 < x_2 < ... < x_m$, such that the $m(m-1)/2$ differences $x_j - x_i$, $1 \leq i < j \leq m$, are distinct. Such a ruler is said to contain m marks and is of length a_m. The objective is to find optimal (minimum length) or near optimal rulers."

The longest currently known optimal ruler has 21 marks and length 333. van Beek reports that even quite small problems (with fewer than fifteen marks) are very difficult for complete methods such as backtracking search, and that their difficulty lies both in proving optimality and in finding a solution, since the problems have either a unique solution or just a handful of solutions.

3 Modelling the problem

To represent these as constraint satisfaction problems, we use m variables, $x_1, ..., x_m$, each with a domain $\{1 .. L\}$, where L is an upper limit on the length of the ruler. We post monotonicity constraints, $x_i < x_{i+1}$ for $1 \leq i < m$. There are three obvious ways of modelling the distinctness constraint on the differences between the marks:

quaternary constraints: we post $O(m^4)$ constraints of the form, $x_j - x_i \neq x_l - x_k$, for all $i < j, k < l$;

ternary and binary constraints: we introduce $m(m-1)/2$ auxiliary variables, d_{ij}, for all $i < j$, constrained to equal $x_j - x_i$ by means of $m(m-1)/2$ ternary constraints; we then post $O(m^4)$ binary not-equals constraints between all pairs of auxiliary variables;

ternary and all-different constraints: we again introduce $m(m-1)/2$ auxiliary variables, d_{ij}, constrained to equal $x_j - x_i$ by means of $m(m-1)/2$ ternary constraints; however, we now post a single all-different constraint on the $m(m-1)/2$ auxiliary variables.

The problem also has a reflection symmetry which we can break by adding the constraint that $x_2 - x_1 < x_m - x_{m-1}$ (or equivalently, $d_{12} < d_{m-1,m}$).

4 Theoretical comparison

As in other studies (e.g. [7, 6]), we will compare the levels of consistency achieved by generalized arc consistency on the different representations. The quaternary constraints express directly a relationship between the original variables which is only represented indirectly via the auxiliary variables in the ternary models. Hence, replacing quaternary constraints with auxiliary variables might be expected to reduce the level of consistency achieved. However, generalized arc consistency on the representation using auxiliary variables and an all-different constraint can be either stronger or weaker than generalized arc consistency on the representation using quaternary constraints.

Theorem 1 *Generalized arc consistency on the representation using auxiliary variables and an all-different constraint is incomparable to generalized arc consistency on the representation using quaternary constraints.*

Proof: Consider a Golomb ruler with $x_1 = \{0\}$, $x_2 = \{1, 2\}$, and $x_3 = \{4\}$. The representation with auxiliary variables and an all-different constraint is generalized arc consistent. However, enforcing generalized arc consistency on the constraint $x_3 - x_2 \neq x_2 - x_1$ in the quaternary representation prunes the value 2 from the domain of x_2.

Consider a Golomb ruler with $x_1 = \{0\}$, $x_2 = \{1, 2\}$, $x_3 = \{3\}$, and $x_4 = \{4, 5\}$. The representation using quaternary constraints is generalized arc consistent. However, enforcing generalized arc consistency on the representation with

auxiliary variables and an all-different constraint shows that the problem is insoluble since the auxiliary variables d_{12}, d_{23} and d_{34} have domains $\{1,2\}$ and thus cannot be all different. □

More surprisingly, replacing the single all-different constraint with a clique of binary not-equals constraints can also be enough to counter the loss of consistency resulting from the removal of the quaternary constraints. As the auxiliary variables are shared between constraints, reasoning about them can achieve extra pruning. In fact, generalized arc consistency on the representation using auxiliary variables and a clique of binary not-equals constraints is incomparable to generalized arc consistency on the representation using quaternary constraints.

Theorem 2 *Generalized arc consistency on the representation using auxiliary variables and binary not-equals constraints is incomparable to generalized arc consistency on the representation using quaternary constraints.*

Proof: Consider the first Golomb ruler in the previous proof. The representation with auxiliary variables and binary not-equals constraints is generalized arc consistent. However, enforcing generalized arc consistency on the representation using quaternary constraints prunes the value 2 from the domain of x_2.

Consider a Golomb ruler with $x_1 = \{0\}$, $x_2 = \{1\}$, $x_3 = \{3\}$, $x_4 = \{7,8\}$, and $x_5 = \{8,9\}$. The representation using quaternary constraints is generalized arc consistent. However, enforcing generalized arc consistency on the representation with auxiliary variables and binary not-equals constraints shows that the problem is insoluble since the auxiliary variable d_{45} has all its possible values (1, 2 or 3) removed from its domain by the constraints with the auxiliary variables d_{12}, d_{23} and d_{13}. □

However, it is not surprising that, in the ternary representation, replacing the single all-different constraint with a clique of binary not-equals constraints reduces the level of consistency achieved.

Theorem 3 *Generalized arc consistency on the representation using auxiliary variables and an all-different constraint is strictly stronger than generalized arc consistency on the representation using auxiliary variables and binary not-equals constraints.*

Proof: It is trivially stronger as generalized arc consistency on an all-different constraint is stronger than arc consistency on a clique of binary not-equals constraints. To show strictness, consider again the second Golomb ruler in the first proof. The representation of this problem using auxiliary variables and binary not-equals constraints is generalized arc consistent. However, enforcing generalized arc consistency on the representation with auxiliary variables and an all-different constraint shows that the problem is insoluble. □

We can show what needs to be added to the ternary representation to make GAC in that case stronger than for the quaternary constraints. In the example given in the proof of Theorem 1, of a ruler with $x_1 = \{0\}$, $x_2 = \{1,2\}$, $x_3 =$

$\{4\}$, the value 2 would be deleted from the domain of x_2 if we added to the ternary representation the ternary constraints $2x_j - x_i - x_k \neq 0$ for all i, j, k with $i < j < k$. These are the only constraints that need be added to ensure that GAC on the ternary constraints prunes at least as many values as GAC on the quaternary constraints. If i, j, k, l are all different, the constraint $x_j - x_i \neq x_l - x_k$ is generalized arc consistent unless at least three of the variables have a singleton domain, and one of the values of the remaining variable, say x_i, violates the constraint; this value will be deleted. But in that case, d_{kl} also has a singleton domain, and arc consistency on $d_{ij} \neq d_{kl}$ would delete the only possible value for d_{kl} from the domain of d_{ij}. Then making the ternary constraint $d_{ij} = x_j - x_i$ generalized arc consistent will delete the same value of x_i: so we would get the same result as with the quaternary constraint. Hence, it is only when the quaternary constraint is actually a ternary constraint, because two of the variables coincide, that the ternary representation has a weaker effect, as shown in Theorem 1. Furthermore, this can only happen when the constraint is $x_j - x_i \neq x_k - x_j$ where $k > j > i$, which is equivalent to the constraint $2x_j - x_i - x_k \neq 0$ given earlier. Other constraints $x_j - x_i \neq x_k - x_i$ or $x_k - x_j \neq x_k - x_i$ where $k > j > i$ are taken care of by the ordering constraints, since $x_k > x_j > x_i$.

This shows that in theory ternary constraints can substitute for (and in fact improve on) the original quaternary constraints.

To conclude, we have shown that theory alone cannot choose between the representations with quaternary constraints and those with auxiliary variables, on the basis of the levels of consistency achievable. However, the results of the next section show that introducing auxiliary variables can be very worthwhile in practice.

5 Experimental results

Table 1 shows the number of branches explored and the CPU time used to find an optimal Golomb ruler, using ILOG Solver, for given numbers of marks. Solver's inbuilt minimization functions were used to find a ruler with minimal length for each number of marks. The table shows the number of branches explored during the search (the number of fails) and the CPU time (on a Silicon Graphics O2), both in finding the optimal solution (F) and in proving optimality thereafter (P). With the all-different constraint, Régin's filtering algorithm [4] was used for efficiency. In all three representations, the search variables are those representing the marks, i.e. x_i, $i = 1, ..., m$, and are ordered lexicographically.

Although the theory of the last section is concerned with generalized arc consistency applied to both the quaternary and ternary constraints, we have not used GAC in these experiments, except on the all-different constraint. However, in section 7 below, we compare GAC on the ternary constraints with arc consistency, and show that in terms of CPU time it is not competitive. The same is likely to be true of GAC on the quaternary representation, especially as, on the quaternary constraints in this problem, GAC will not often be able to prune

Marks	Stage	Quaternary		Ternary + Not-Equals		Ternary + All-Diff	
		Branches	CPU	Branches	CPU	Branches	CPU
4	F	1	0.003	1	0.002	1	0.003
	P	1	0.001	1	0.001	1	0.001
5	F	7	0.006	3	0.003	2	0.006
	P	5	0.004	7	0.002	2	0.002
6	F	15	0.020	6	0.007	6	0.012
	P	63	0.042	39	0.015	10	0.006
7	F	116	0.170	28	0.023	26	0.033
	P	594	0.801	327	0.198	84	0.125
8	F	756	2.03	130	0.104	98	0.124
	P	4852	14.0	2605	2.27	599	1.23
9	F	7271	31.7	1622	1.70	816	1.56
	P	33679	168	17823	22.1	2924	9.69
10	F	78503	657	21507	27.9	9757	24.3
	P	-	-	-	-	13707	68.3
11	F	-	-	-	-	31666	94.5
	P	-	-	-	-	-	-

Table 1. Branches explored and CPU time (seconds) used to find a minimal length ruler (F) or prove that none shorter exists (P). A - means that the run was cut off after 10^5 branches.

any values. (Furthermore, in Solver, GAC is harder to implement for quaternary constraints than for ternary constraints.)

Table 1 shows that in practice the quaternary representation is much less efficient than the other two, both in terms of branches explored and CPU time. The representation with the all-different constraint explores a much smaller search tree, although it is not always quicker than the not-equals representation, especially for the smaller problems. The experiments show that introducing auxiliary variables is in practice well worthwhile in the Golomb ruler problem.

6 Branching heuristics

We next investigate the effect of variable ordering heuristics. We concentrate on the auxiliary variable representation with the single all-different constraint as this is the most efficient in terms of CPU time, from those compared in Table 1. Table 2 compares lexicographic ordering with two versions of the heuristic which chooses next the variable with smallest remaining domain (SD); this heuristic has often been found to give good results, although mainly in binary constraint satisfaction problems. In one version of the smallest-domain heuristic, both the original and the auxiliary variables are used as search variables. In the other version (restricted SD) only the original variables are used as search variables. The lexicographic ordering selects the original variables in order, starting from x_2 (since x_1 is already assigned to 0). When all the original variables have been assigned, constraint propagation will have already assigned the auxiliary vari-

Marks	Stage	SD		Restricted SD		Lexicographic	
		Branches	CPU	Branches	CPU	Branches	CPU
4	F	1	0.003	1	0.003	1	0.003
	P	1	0.001	1	0.001	1	0.001
5	F	2	0.006	2	0.006	2	0.006
	P	2	0.002	2	0.002	2	0.002
6	F	5	0.011	7	0.011	6	0.012
	P	8	0.007	15	0.009	10	0.006
7	F	32	0.034	42	0.044	26	0.033
	P	166	0.194	137	0.135	84	0.118
8	F	269	0.281	129	0.151	98	0.124
	P	1893	3.48	840	1.35	599	1.23
9	F	2310	3.46	1051	1.63	816	1.56
	P	22059	59.1	4388	10.8	2924	9.69
10	F	-	-	10084	20.4	9757	24.3
	P	-	-	21916	79.9	13707	68.3
11	F	-	-	42946	107	31666	94.5

Table 2. Branches explored and CPU time (seconds) used to find a minimal length ruler (F) or prove that none shorter exists (P). A - means that the run was cut off after 10^5 branches.

ables as well, and therefore a restricted form of this heuristic does exactly the same thing.

If the difference variables are used as the search variables, instead of the original variables, and are assigned in the order $d_{12}, d_{13}, d_{14}, ..., d_{1m}, ..,$ constraint propagation will ensure that the variables $x_2, x_3, ..., x_m$ are assigned values simultaneously. As before, the remaining difference variables will also be assigned values as the result of constraint propagation from the x_i variables. Hence, lexicographic ordering of the auxiliary variables gives identical results to lexicographic ordering of the original variables, in both ternary representations.

It is clear from Table 2 that lexicographic ordering gives much better results than smallest domain ordering. With the smallest domain ordering, it is a good idea to search only on the original variables, and not on the auxiliary variables as well. With lexicographic ordering, however, it is immaterial which set of variables are used as the search variables.

In the quaternary representation, smallest domain ordering again explores more branches than lexicographic ordering, but the difference is not so marked as with the ternary representation shown in Table 2.

Hence, our results show that in this problem it is better to build up the ruler progressively, using either the original or the auxiliary variables, than to move back and forth along the ruler as the domain sizes change. The reason for this may be that, in the former case, the smaller differences will be assigned first, and this will allow the all-different constraint to increase the lower bound on the remaining auxiliary variables and hence prune more values.

7 Binary encodings

An alternative strategy for solving these non-binary models is to encode them into binary problems using one of the standard encodings such as the hidden variable or the dual encoding [1, 6]. In the case of Golomb rulers, the double encoding introduced in [6] is more practical than the dual encoding. The double encoding combines together all the constraints from the dual and the hidden variable encodings. In a dual encoding, the dual variables associated with either the all-different constraint or the clique of not-equals constraints have such large domains that we cannot afford to enforce arc consistency. In a double encoding, whilst we introduce dual variables associated with the ternary constraints, we can ignore the dual variables associated with the all-different constraint or the clique of not-equals constraints as they are redundant. This makes it computationally feasible to use the double encoding. Finally, whilst encodings of models with ternary constraints are practical, encodings of the quaternary constraints have domains which are prohibitively large. We therefore looked at four new models.

hidden variable encoding + all-different constraint: each ternary constraint is replaced by a hidden variable with domain of size $O(L^2)$; we also post a single all-different constraint between the auxiliary variables;

hidden variable encoding + not-equals constraint: each ternary constraint is replaced by a hidden variable with domain of size $O(L^2)$; we also post $O(m^4)$ binary not-equals constraints between each pair of auxiliary variables; this model contains purely binary constraints;

double encoding + all-different constraint: each ternary constraint is replaced by a hidden variable with domain of size $O(L^2)$; we also post compatibility constraints between hidden variables that share variables, and a single all-different constraint between the auxiliary variables;

double encoding + not-equals constraint: each ternary constraint is replaced by a hidden variable with domain of size $O(L^2)$; we also post compatibility constraints between hidden variables that share variables, and $O(m^4)$ binary not-equals constraints between each pair of auxiliary variables; this model contains purely binary constraints.

We did not consider it feasible to find optimal rulers using the binary encodings, as in Tables 1 and 2, since this normally requires setting L, the maximum length of the ruler, to some large value initially and then gradually reducing the maximum length allowed until no ruler can be found. Unfortunately, because the domain sizes of the hidden variables in the binary encodings are $O(L^2)$, the early stages of the optimization, when L is large and it should be very easy to find a ruler of length less than L, become very time-consuming. Instead, we have used the known optimal rulers to compare the encodings. We first find a ruler with length $\leq L_{min}$, where L_{min} is the optimal length, and then show that there is no ruler with length $\leq L_{min} - 1$.

Table 3 gives results from the models in which all the constraints are binary, including the all-different constraint, which is treated as a collection of binary $\neq$

constraints. These models are compared with the ternary representation, with the all-different constraint treated similarly. Two versions of the ternary representation are shown: in one, Solver's standard constraint propagation based on arc consistency is used on the ternary constraints. In the other, generalized arc consistency is used for the ternary constraints: these constraints are expressed using Solver's IlcTableConstraint function which implements Bessière and Régin's GAC-schema [2].

Marks	Length	Hidden + $\neq$		Double + $\neq$		3-ary + GAC + $\neq$		3-ary + $\neq$	
		Branches	CPU	Branches	CPU	Branches	CPU	Branches	CPU
4	6	1	0.008	1	0.009	1	0.003	1	0.002
4	5*	2	0.005	2	0.007	2	0.002	2	0.001
5	11	2	0.034	2	0.064	2	0.012	2	0.003
5	10*	9	0.042	9	0.079	9	0.014	9	0.004
6	17	5	0.150	5	0.473	5	0.052	5	0.005
6	16*	36	0.304	32	0.912	36	0.109	42	0.018
7	25	18	0.645	17	3.70	18	0.251	21	0.014
7	24*	286	3.13	237	11.9	286	1.18	341	0.206
8	34	45	2.34	45	21.2	45	0.964	54	0.043
8	33*	2015	31.4	1461	117	2012	12.1	2648	2.32
9	44	708	22.0	506	147	705	8.51	1098	1.20
9	43*	12822	302	8846	1180	12815	115	18723	23.4

Table 3. Branches explored and CPU time (seconds) used to find a ruler of no more than the specified length or prove that none exists, using lexicographic ordering. * indicates that there is no ruler of this length.

Fewer branches are explored in the double than in the hidden encoding. The hidden encoding gives very similar results, in terms of the size of the search tree, to the ternary representation with generalized arc consistency. As expected, GAC on the ternary representation reduces the size of the search tree considerably, compared with just arc consistency. However, the CPU times tell a different story. Both the binary encodings, and especially the double, are far worse than the ternary models. This is not surprising as arc consistency takes longer to enforce in the binary encodings due to the large domain sizes of the hidden variables. The worst-case time complexity of maintaining generalized arc consistency on the ternary constraints is $O(L^3)$, whilst that for arc consistency on the hidden variable representation is $O(L^4)$. Furthermore, in terms of CPU time, enforcing generalized arc consistency on the ternary constraints is not worthwhile; however, this may be partly due to the fact that a general algorithm is being used. An algorithm specialized for these constraints might give better results.

Table 4 gives an equivalent comparison when we replace the clique of binary not-equals constraints with a single all-different constraint. The results are similar: the double encoding gives the best results in terms of the number of branches explored, but is not practicable as the CPU time is far longer

than with the other representations. The quickest results come from using the ternary representation without GAC, even though this gives the largest search tree. Comparison of Tables 3 and 4 shows that in all four models it is worthwhile to use a single all-different constraint rather than binary not-equals constraints.

Marks	Length	Hidden + AllDiff		Double + AllDiff		3-ary + AllDiff + GAC		3-ary + AllDiff	
		Branches	CPU	Branches	CPU	Branches	CPU	Branches	CPU
4	6	0	0.007	0	0.007	0	0.003	0	0.002
4	5*	1	0.001	1	0.001	1	0.001	1	0.002
5	11	1	0.037	1	0.064	1	0.014	1	0.004
5	10*	1	0.028	1	0.027	1	0.010	1	0.003
6	17	3	0.156	3	0.473	3	0.059	3	0.008
6	16*	11	0.199	10	0.563	11	0.079	11	0.011
7	25	10	0.660	10	3.65	10	0.267	13	0.019
7	24*	88	2.13	83	8.33	88	0.809	94	0.134
8	34	29	2.37	29	21.3	28	0.996	32	0.053
8	33*	567	19.2	501	83.8	567	7.57	620	1.30
9	44	361	19.6	293	144	360	8.10	465	1.05
9	43*	2746	159	2398	712	2741	63.4	3268	10.7
10	55	2064	151	1793	892	2062	61.6	2750	9.35
10	54*	13883	1240	-	-	13879	493	15606	77.7

Table 4. Branches explored and CPU time (seconds) used to find a ruler of no more than the specified length or prove that none exists, using lexicographic ordering. * indicates that there is no ruler of this length. - means that the run was not attempted.

8 Implied constraints

So far, we have considered different ways of representing the basic problem, and the effect of putting more or less effort into constraint propagation through some form of arc consistency. Another route to improved performance is to add constraints to the model: the constraints we can consider are not required to ensure that solutions found are correct, but are implied by constraints already in the model. Such implied constraints are derived from two or more existing constraints and so can be viewed as partially achieving some higher level of consistency than arc consistency. The intention in adding implied constraints is that propagating them will lead to more, or at least earlier, pruning of values from the domains of variables than would otherwise occur.

In the Golomb ruler problem, one set of implied constraints has already been mentioned, namely the constraints $2x_j - x_i - x_k \neq 0$, for $i < j < k$. As discussed in section 4, these constraints, in theory, give additional pruning in the ternary representation and allow GAC on the ternary representation to be stronger than GAC on the quaternary representation. These constraints are implied by the constraints $d_{ij} \neq d_{jk}$, $d_{ij} = x_j - x_i$ and $d_{jk} = x_k - x_j$.

Although implied constraints have often been found to lead to significantly faster solution time (e.g. [3, 5]), choosing useful implied constraints is something of an art. We can state some conditions for good implied constraints. First, to allow useful constraint propagation, we require either constraints for which specialized and efficient constraint propagation algorithms are available, or else constraints of small arity. Secondly, we should be able to envisage circumstances in which a candidate implied constraint will lead to values being deleted from the domains of variables which would not be deleted by the existing constraints.

However, although these are desirable, they are not sufficient to ensure a reduction in solution time. Adding constraints brings an overhead, and a worthwhile implied constraint must achieve sufficient additional pruning to more than offset this overhead. It is hard to predict whether a given candidate implied constraint will do this, without experimenting. For instance, the constraints $2x_j - x_i - x_k \neq 0$ meet the conditions, since they are of small arity and we have given in section 4 an example of a value being deleted by a constraint of this form which would not be deleted by the existing ternary constraints. However, experiments with the ternary model with an all-different constraint, which has already been shown to give the best results in terms of CPU time, show that adding these constraints makes no difference to the number of branches explored, if a lexicographic ordering is used. Hence, adding these constraints increases the CPU time. If a smallest-domain ordering is used, the number of branches explored decreases significantly with these constraints, but not sufficiently to beat the lexicographic ordering. Overall, despite their theoretical advantage, they are not worth using.

Below, we consider other implied constraints, and their effect when added to the model which has so far proved best, i.e. the ternary representation, with auxiliary variables, a single all-different constraint and lexicographic ordering.

8.1 Ordering Constraints on the Auxiliary Variables

For any set of three indices i, j, k with $i < j < k$, it must be true that $d_{ij} < d_{ik}$ and $d_{jk} < d_{ik}$. These constraints are implied by the constraints defining the difference variables and the ordering constraints $x_i < x_j < x_k$ (which in turn are implied by the constraints $x_i < x_{i+1}$ for $1 \leq i < m$). In this case, the implied constraints are binary, hence cheap to propagate, and we can see that they could lead to domain reductions not achievable otherwise. For instance, suppose we are trying to find the minimal length ruler with 5 marks and are currently looking for one with length 11 (which is the minimal length). The initial domain of each of the variables x_2, x_3, x_4 is $\{0..11\}$. The ordering constraints on the original variables, and the fact that $x_1 = 0$ and $x_5 = 11$, will reduce their domains to x_2: $\{1..8\}$, x_3: $\{2..9\}$, x_4: $\{3..10\}$. The initial domain of the variables d_{23}, d_{24}, d_{34} will be $\{1..11\}$. The reduction in the domains of the original variables will reduce the domains of these variables too: d_{23} to $\{1..8\}$, d_{24} to $\{1..9\}$ and d_{34} to $\{1..8\}$. The all-different constraint cannot make any further reductions, whether or not we invoke the full filtering algorithm. However, if we have the constraints $d_{23} < d_{24}$ and $d_{34} < d_{24}$, 1 is removed from the domain of d_{24}. So this example

shows that in theory we might get additional propagation from such constraints. Hence, on theoretical grounds, there seems good reason to suppose that adding these constraints would give improved performance.

The first column of Table 5 shows the best results obtained hitherto (already seen in Tables 1 and 2.) The second column shows the results of adding the ordering constraints on the difference variables to this model. The ordering constraints do reduce the size of the search tree explored, except for the smallest problems, although they do not in most cases reduce the CPU time. Hence, somewhat surprisingly, these constraints are not worthwhile, in conjunction with the rest of the search strategy.

Marks	Stage	Ternary + AllDiff		Ternary + AllDiff + order constraints	
		Branches	CPU	Branches	CPU
4	F	1	0.003	1	0.004
	P	1	0.001	1	0.001
5	F	2	0.006	2	0.008
	P	2	0.002	2	0.002
6	F	6	0.012	6	0.019
	P	10	0.006	10	0.009
7	F	26	0.033	26	0.054
	P	84	0.118	79	0.116
8	F	98	0.124	98	0.189
	P	599	1.23	556	1.22
9	F	816	1.56	780	1.84
	P	2924	9.69	2726	9.91
10	F	9757	24.3	9259	27.9
	P	13707	68.3	12527	69.9
11	F	31666	94.5	30797	112
	P	343220	2000	322579	2070

Table 5. Branches explored and CPU time to find a minimal length ruler (F) or prove that none shorter exists (P) with and without order constraints on the difference variables.

8.2 Improved Bounds on the Auxiliary Variables

The variables d_{ij} are currently set up with lower bound 1 and upper bound the maximum length allowed for the ruler. The ordering constraints described above improve these bounds on a *sequence* of differences; for instance, because $d_{12} < d_{13} < d_{14} ... < d_{1n}$, the lower bounds of these variables will be 1, 2, 3, ..., $n-1$. However, it is clear that these lower bounds are not in most cases attainable: we cannot, for instance, have $d_{13} = 2$, because that would mean $d_{12} = d_{23} = 1$.

How do we get tighter bounds, in general? If we consider that $d_{ij} = d_{i,i+1} + d_{i+1,i+2} + ... + d_{j-1,j}$, we see that d_{ij} is the sum of $j - i$ integers, which are

constrained to be all different. So d_{ij} must be at least equal to the sum of the first $j-i$ integers, i.e. $d_{ij} \geq (j-i)(j-i+1)/2$.

The chain of reasoning leading to this as an implied constraint is harder to disentangle than in the previous examples. We first have to write $x_j - x_i$ as $(x_j - x_{j-1}) + (x_{j-1} - x_{j-2}) + ... + (x_{i+1} - x_i)$ and then use the constraints defining the difference variables to derive $d_{ij} = d_{i,i+1} + d_{i+1,i+2} + ... + d_{j-1,j}$. Then, since there is an all-different constraint on all the difference variables, there is an all-different constraint on any subset. Hence, d_{ij} is constrained to be the sum of $j-i$ different integers. Solver, and other constraint programming tools as far as we are aware, do not provide a "sum of n all different" constraint, but we can use it, outside Solver, to derive the new lower bounds as above.

Similarly, note that $d_{12} + d_{23} + ...d_{n-2,n-1} + d_{n-1,n} = x_n$. So, $d_{ij} = x_n - (d_{12} + d_{23} + .. + d_{i-1,i} + d_{j,j+1} + ...d_{n-1,n})$. The differences on the right hand side of this expression are again a set of different integers, and there are $n-1-j+i$ of them. So we can maximise the RHS by minimising the sum of these consecutive differences. This gives $d_{ij} \leq x_n - (n-1-j+i)(n-j+i)/2$.

These constraints are cheap to implement: the first set are just unary constraints, the second set are binary constraints involving x_n. As x_n is reduced, these constraints get tighter, so that they are more effective the closer we get to the minimal length.

We can tighten the lower bounds on the difference variables still further. The lower bounds described above are based on the observation that the difference d_{ij} must be made up of the sum of $j-i$ different integers. However, this section of the ruler, i.e. from mark i to mark j, must itself form a Golomb ruler with $j-i+1$ marks, although not necessarily one of minimal length. So having found all minimal length rulers with fewer than n marks, we can use the lengths of these rulers, $l_2, l_3, ..., l_{n-1}$, while searching for the minimal length ruler with n marks. That is, $d_{ij} \geq l_{j-i+1}$, provided that $i > 1$ or $j < n$.

Table 6 shows the results of adding these bounds to the model, with and without the order constraints described in the last section. The first two columns of the table show that reducing the initial domains of the difference variables in this way leads to significant benefits. The search space explored is considerably reduced, and so is the CPU time: because of the smaller domains, the CPU time is less than before even if there is no reduction in the number of branches explored. The final column shows that also adding order constraints on the difference variables, as described in section 8.1, gives almost no further reduction in the number of branches explored, and is considerably more expensive in CPU time.

8.3 Dynamic Bounds on the Auxiliary Variables

As mentioned in section 8.2, there is an implicit "sum of n all different" constraint on each difference variable, which we have so far used to reduce their initial domains. Ideally, we should like to be able to use this constraint during search as well. For instance, if we have assigned the values 1 and 3 to x_2 and x_3, then

Marks	Stage	3-ary + AllDiff		+ tighter bounds		+ tighter bounds + order constraints	
		Branches	CPU	Branches	CPU	Branches	CPU
4	F	1	0.003	1	0.004	1	0.004
	P	1	0.001	1	0.001	1	0.001
5	F	2	0.006	2	0.006	2	0.007
	P	2	0.002	2	0.001	2	0.002
6	F	6	0.012	6	0.012	6	0.013
	P	10	0.006	6	0.004	6	0.005
7	F	26	0.033	26	0.032	26	0.038
	P	84	0.118	54	0.051	54	0.063
8	F	98	0.124	98	0.128	98	0.160
	P	599	1.23	385	0.503	385	0.635
9	F	816	1.56	718	1.17	718	1.47
	P	2924	9.69	1751	3.61	1751	4.48
10	F	9757	24.3	7971	17.1	7948	21.3
	P	13707	68.3	7812	24.0	7807	29.5
11	F	31666	94.5	29251	80.8	29190	108
	P	343220	2000	252985	1020	252577	1300

Table 6. Branches explored and CPU time to find a minimal length ruler (F) or prove that none shorter exists (P), with and without tighter bounds on the difference variables.

any difference variable d_{ij} where $i \geq 2$ must be at least the sum of $j-i$ different integers *excluding* 1, 2 and 3.

We have implemented a limited form of propagation for this constraint, by keeping track of the largest mark so far assigned a value, x_{max}, and the values which have so far been assigned to the difference variables. Whenever a difference variable is assigned, we calculate the sum of the $m-max$ smallest integers which have not yet been assigned to the difference variables, and post a constraint on this branch of the search tree that the value of $d_{max,m}$ must be at least this sum.

Table 7 shows the effect of adding this constraint propagation to the model giving the best results so far in term of CPU time (from Table 6). Except for the smallest problems, there are significant reductions in both the number of branches explored and CPU time. We have also solved the 12-mark problem using this model. Finding the optimal solution took 1,398,326 backtracks, and the total number of backtracks including the proof of optimality was 1,911,435: the total CPU time was about 2.5 hours. This is a significant reduction in search effort from the results by Jean-Francois Puget reported in CSPLib: he found a solution for this problem using ILOG Solver after 2,042,000 backtracks, and the total number of backtracks including the proof of optimality was 3,143,316.

It is possible that more effort put into propagating the "sum of n all different" constraint might pay dividends. On the other hand, what we have implemented so far is relatively lightweight, and given that the variables are being assigned in lexicographic order, recalculating the lower bound on the length of the rest of the ruler is an obvious way to try to prune values. Attempting to use the

Marks	Stage	3-ary + AllDiff + tighter bounds		+ dynamic bounds	
		Branches	CPU	Branches	CPU
4	F	1	0.004	1	0.004
	P	1	0.001	1	0.001
5	F	2	0.006	2	0.007
	P	2	0.001	2	0.002
6	F	6	0.012	6	0.013
	P	6	0.004	6	0.004
7	F	26	0.032	25	0.032
	P	57	0.051	50	0.049
8	F	98	0.128	80	0.111
	P	385	0.503	355	0.480
9	F	718	1.17	509	0.822
	P	1751	3.61	1564	3.41
10	F	7971	17.1	5428	12.0
	P	7812	24.0	6810	22.1
11	F	29251	80.8	19211	53.6
	P	252985	1020	224636	967

Table 7. Branches explored and CPU time to find a ruler of minimal length (F) or prove that none shorter exists (P), with and without dynamic lower bounds on the difference constraints.

constraint more extensively is likely to incur increased overheads, and possibly result in little or no extra pruning.

9 Conclusions

We began with a simple representation of the Golomb ruler problem using quaternary constraints on the variables representing the positions of the marks, and have considered many alternatives to this basic model. We showed theoretically that the level of consistency achieved by generalized arc consistency in the quaternary representation cannot be compared with the consistency level achieved in a ternary representation with auxiliary variables. However, experimental results (using AC rather than GAC on the quaternary and ternary constraints) show that the extra propagation through the auxiliary variables reduces both the number of explored branches and the CPU time. This is true whether using an all-different constraint over all the auxiliary variables or representing the all-different constraint as a clique of binary not-equals constraints. We have shown that the double encoding of non-binary into binary constraints reduces the number of explored branches significantly. However, the CPU time is much greater because of the cost of enforcing arc consistency on hidden variables with large domains. We have also considered several sets of implied constraints, in conjunction with the ternary representation of the problem, some of which can reduce both the number of branches explored and the CPU time significantly. Our final model is a dramatic improvement over the basic model which we started with:

for instance, the time taken to find an optimal 10–mark ruler has been reduced 50-fold.

What general lessons can be learned from this study? First, even simple problems can be modelled in many different ways. Counting all possible encodings, both binary and non-binary, this study alone has suggested fifteen possible models for the problem, although we have omitted some of these from our experiments (in particular the binary translations of the quaternary representations). If we include all possible ways of adding the implied constraints discussed in section 8, there would be many more.

Secondly, finding the best model involves a trade-off between the arity of the constraints and the efficiency with which we can reason about them. The most effective model in this case study was not the one involving quaternary constraints but that with ternary constraints which allows us to post a single, large, all-different constraint.

Thirdly, the addition of implied constraints may allow us to achieve higher levels of consistency. However, finding good implied constraints which will result in an overall improvement in solution times is not easy. Although we have suggested some guidelines, our experience with this problem indicates that often it will only be possible to tell by experiment whether a particular set of implied constraints is worthwhile.

Acknowledgments

The third author is supported by EPSRC award GR/K/65706. All authors wish to thank the other members of the APES research group.

References

1. F. Bacchus and P. van Beek. On the Conversion Between Non-Binary and Binary Constraint Satisfaction Problems. In *Proceedings AAAI'98*, pages 311–318, 1998.
2. C. Bessière and J.-C. Régin. Arc consistency for general constraint networks: preliminary results. In *Proceedings IJCAI'97*, volume 1, pages 398–404, 1997.
3. L. Proll and B. Smith. ILP and Constraint Programming Approaches to a Template Design Problem. *INFORMS Journal on Computing*, 10:265–275, 1998.
4. J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings AAAI'94*, volume 1, pages 362–367, 1994.
5. J.-C. Régin. Minimization of the number of breaks in sports scheduling problems using constraint programming. In *Proceedings of the DIMACS Workshop on Constraint Programming and Large Scale Discrete Optimization*, Sept. 1998.
6. K. Stergiou and T. Walsh. Encodings of Non-Binary Constraint Satisfaction Problems. In *Proceedings AAAI'99*, 1999.
7. K. Stergiou and T. Walsh. The Difference All-Difference Makes. In *Proceedings IJCAI'99*, 1999.