# Value Ordering for Finding All Solutions

**Barbara M. Smith**
Cork Constraint Computation Centre, Ireland.
b.smith@4c.ucc.ie

**Paula Sturdy**
University of Huddersfield, U.K.
p.sturdy@hud.ac.uk

## Abstract

In finding all solutions to a constraint satisfaction problem, or proving that there are none, with a search algorithm that backtracks chronologically and forms $k$-way branches, the order in which the values are assigned is immaterial. However, we show that if the values of a variable are assigned instead via a sequence of binary choice points, and the removal of the value just tried from the domain of the variable is propagated before another value is selected, the value ordering can affect the search effort. We show that this depends on the problem constraints; for some types of constraints, we show that the savings in search effort can be significant, given a good value ordering.

## 1 Introduction

It is well-known that the order in which variables are instantiated can make an enormous difference to the search effort in solving a constraint satisfaction problem, whether just one solution is required, or all solutions. However, value ordering is relatively neglected, partly because no cheap general-purpose value ordering heuristics are known, and partly because it has been accepted that value ordering is not important if all solutions are required, or there is no solution.

Frost and Dechter [1995] showed that, with backjumping, the value ordering can have an effect on the search for all solutions, but say: "With backtracking the order in which values are chosen makes no difference on problems which have no solution, or when searching for all solutions." Their argument is that when a node corresponding to a variable is created in the search tree, the children correspond to the values of that variable, and that the subtrees rooted at the children nodes are explored independently. To find all solutions, or to prove that there are none, every subtree must be explored, and the order makes no difference to the overall search.

This argument assumes that the search tree is formed by $k$-way branching [Mitchell, 2003], i.e. when a variable with $k$ values in its domain is selected for instantiation, $k$ branches are formed. Constraint solvers such as ILOG Solver and ECL$^i$PS$^e$ by default use a search strategy similar to the MAC algorithm [Sabin and Freuder, 1997], and in particular use 2-way, or binary, branching. When a variable $x$ is selected for instantiation, its values are assigned via a sequence of binary choices. If the values are assigned in the order $\{v_1, v_2, ..., v_k\}$, the first choice point creates two alternatives, $x = v_1$ and $x \neq v_1$. The first (the left branch) is explored; if that branch fails, or if all solutions are required, the search backtracks to the choice point, and the right branch is followed instead. Crucially, the constraint $x \neq v_1$ is propagated, before a second choice point is created between $x = v_2$ and $x \neq v_2$, and so on. (The MAC algorithm also allows the possibility that on backtracking after trying $x = v_1$, a different variable could be assigned next, not just a different value: that possibility is not considered here.)

With binary branching, the subtrees resulting from successive assignments to a variable are not explored independently: propagating the removal of a value from the current variable's domain on the right branch can lead to further domain reductions. This propagation can affect the search when future values of the variable are considered: indeed, sometimes a future value can be removed from the domain. Hence, the order in which the values are assigned can affect the search.

[Smith, 2000] gives an example of the value ordering affecting the search to find all solutions. Here, we present a more extensive investigation and explain how value ordering can affect search effort. We show that, depending on the problem constraints, the saving in search effort over $k$-way branching can be around 50%, given a good value ordering.

## 2 Search Trees for a Golomb Ruler Problem

First, we examine in detail the effect of the value ordering when finding all solutions, using a variant of the Golomb ruler problem (prob006 in CSPLib). A Golomb ruler with $m$ marks may be defined as a set of $m$ integers $0 = x_0 < x_1 < ... < x_{m-1}$, such that the $m(m-1)/2$ differences $x_j - x_i$, $0 \leq i < j \leq m-1$, are distinct. The length of the ruler is $x_{m-1}$, and the objective is to find a minimum length ruler. Modeling the problem is discussed in [Smith *et al.*, 2000].

To create a problem with no solution, we set $x_{m-1} = minl - 1$, where $minl$ is the minimum possible length. The model is chosen so that the search trees (using ILOG Solver's default binary branching) for one instance of the problem are small enough to display, while the value ordering makes a significant difference to the number of backtracks.

The chosen model is not the quickest way to solve the problem, since it does expensive constraint propagation in order to reduce search. Auxiliary variables represent the $m(m-1)/2$ differences between the marks on the ruler, defined by $d_{i,j} = x_j - x_i$, where $i < j$; generalized arc consistency (GAC) is enforced on these ternary constraints using ILOG Solver's table constraints. We also enforce GAC on the allDifferent constraint on the difference variables. Since $x_i$ is the sum of $i$ difference variables, its minimum value is set to be the sum of the first $i$ integers. Similarly, the length $x_{m-1-i}$ to $x_{m-1}$ is at least the sum of the first $i$ integers, and this reduces the maximum value of $x_{m-1-i}$. We add the constraint $d_{0,1} < d_{m-2,m-1}$ to break the reflection symmetry.

The search variables are $x_0, x_1, ..., x_{m-1}$, assigned in that order. We compare two value ordering heuristics; choosing either the smallest value in the domain, or the largest. Choosing the largest value would not be a sensible strategy for finding a minimum length Golomb ruler, since the length is not known in advance, but here it shows the effect on search of changing the value ordering.

Figure 1 shows the search trees resulting from proving that there is no 6-mark Golomb ruler with length 16, with the two selected value orderings. The difference is striking, given the conventional view that the value ordering only reorders the search, and does not affect the overall search effort.

The black circles show where a failure is detected and the search backtracks. When values are assigned in increasing order, on backtracking to take the right branch the domain of the variable can sometimes be reduced, or eliminated altogether. We examine a case in detail. Initial constraint propagation reduces the domains of the search variables to $x_0$ : 0; $x_1$ : $\{1..5\}$; $x_2$ : $\{3..10\}$; $x_3$ : $\{6..13\}$; $x_4$: $\{10..14\}$; $x_5$ : 16. On backtracking from $x_1 = 1$, the right branch has $x_1 \in \{2..5\}$. Constraint propagation reduces this to $x_1 = 2$: there are 14 pairwise differences between the marks in this instance (excluding $d_{0,5}$, which has already been assigned) and just 14 possible values for these differences, from 1 to 14. If $x_1 \neq 1$, then $x_4 \leq 13$, because of the symmetry constraint. Hence, the value 14 can only be assigned to the difference $x_5 - x_1$, which means that $x_1 = 2$. Enforcing GAC on the allDifferent constraint on the difference variables makes this inference, and the branch $x_1 \in \{3..5\}$ is not considered further. On the other hand, when values are assigned in decreasing order, the domain of $x_1$ on the right branch always contains the value 1, and the allDifferent constraint is already GAC.

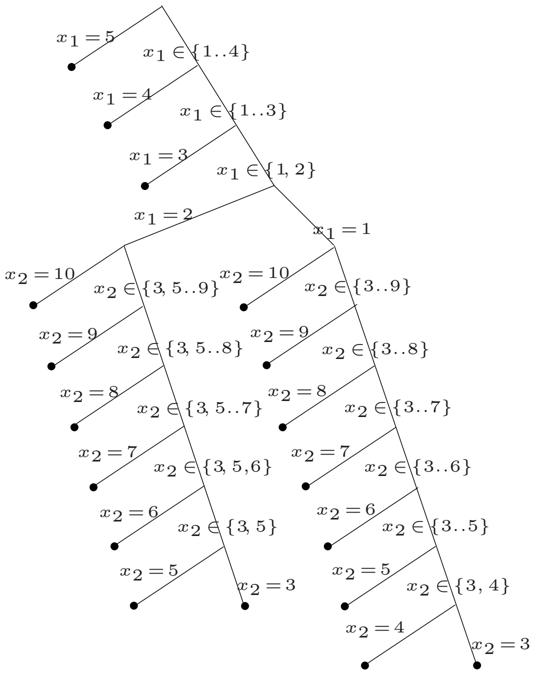
| $n$ | Increasing order | Decreasing order | $k$-way branching |
|---|---|---|---|
| 6 | 8 | 18 | 18 |
| 7 | 71 | 97 | 113 |
| 8 | 304 | 456 | 525 |
| 9 | 1,748 | 2,809 | 3,252 |
| 10 | 6,759 | 11,395 | 13,592 |
| 11 | 117,251 | 206,510 | 227,815 |

Table 1: Number of backtracks to prove that there is no Golomb ruler with $n$ marks of length $minl-1$, where $minl$ is the minimum possible length, using binary branching with increasing or decreasing value ordering, or $k$-way branching.

Table 1 gives the results for larger instances of the same problem, and also shows the effect of using an implementation of $k$-way branching in ILOG Solver. $k$-way branching nearly doubles the number of backtracks compared with binary branching using increasing value order. Although decreasing order is much worse than increasing order, it still allows significantly less search than $k$-way branching for the larger instances.

The runtimes for $k$-way branching are usually shorter than for binary branching with decreasing value ordering, even when the search tree is larger. For $n$ = 11, $k$-way branching takes 1390 sec. (on a 600MHz Celeron PC), compared to 1410 sec. for binary branching with decreasing value ordering. With increasing value ordering, binary branching is significantly faster (980 sec.).

The explanation given earlier for the immediate failure of the right branch at the choice point between $x_1 = 2$ and $x_1 \in \{3..5\}$ depends on the global allDifferent constraint.



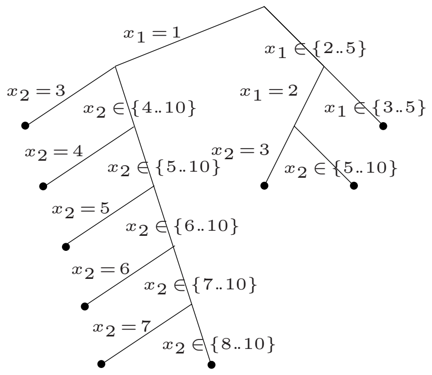


Figure 1: Search trees formed in proving that there is no Golomb ruler with 6 marks of length 16, using (top) decreasing value order and (bottom) increasing value order.

However, even if the allDifferent constraint is treated as a clique of binary $\neq$ constraints and the constraints $d_{i,j} = x_j - x_i$ are not made GAC, choosing the smallest value in the domain requires fewer backtracks to prove insolubility than choosing the largest value. The crucial constraints are $x_0 < x_1 < ... < x_{m-1}$ and the symmetry constraint $d_{0,1} < d_{m_2,m-1}$, which is equivalent to $x_1 < 16 - x_4$ in the example analysed. With monotonic binary constraints such as these, the removal of the largest or smallest value in the domain of one of the variables can reduce the domain of the other, and removing any other value has no effect [van Hentenryck *et al.*, 1992]. Given a constraint $x < y$ with $x$ assigned before $y$, trying the values of $x$ in increasing order means that on backtracking, the smallest value in the domain of $x$ is removed on the right branch and this removal in turn reduces the domain of $y$. Since the variables are assigned in lexicographic order in this case, this explains why assigning the values in increasing order has the largest effect on the search.

## 3 Graceful Labeling of a Graph

In this section, we consider further the effect of value ordering when there are monotonic constraints, and the interaction of the variable ordering and the value ordering.

The problem is that of determining all graceful labelings of the graph shown in Figure 2. A labeling $f$ of the nodes of a graph with $q$ edges is *graceful* if $f$ assigns each node a unique label from $\{0, 1, ..., q\}$ and when each edge $xy$ is labeled with $|f(x) - f(y)|$, the edge labels are all different.

A possible CSP model has a variable for each node, $x_1, x_2, ..., x_n$, each with domain $\{0, 1, ..., q\}$ and a variable for each edge, $d_1, d_2, ..., d_q$, each with domain $\{1, 2, ..., q\}$. The constraints are: if edge $k$ joins nodes $i$ and $j$ then $d_k = |x_i - x_j|$; $x_1, x_2, ..., x_n$ are all different, as are $d_1, d_2, ..., d_q$. Since the edge variables must be assigned a permutation of the values 1 to $q$, it is worthwhile to enforce GAC on the allDifferent constraint. The allDifferent constraint on the node variables is looser (9 variables and 17 possible values in the example of Figure 2) and this is expressed by $\neq$ constraints. The search variables are $x_1, x_2, ..., x_n$.

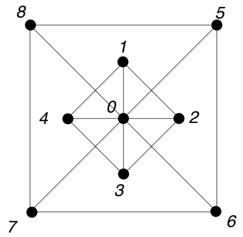


Figure 2: A graph that can be gracefully labeled. The node numbers correspond to the variables of the CSP.

Symmetrically equivalent solutions can be eliminated by adding constraints to the model [Petrie and Smith, 2003].

- $x_0 < 8$ (to eliminate the complement symmetry, i.e. the symmetry that replaces every value $v$ by $q - v$).
- $x_1 < x_2$; $x_1 < x_3$; $x_2 < x_4$ (to eliminate rotations and reflections in the cycle consisting of nodes $1, 2, 3, 4$).
- $x_5 < x_6$; $x_5 < x_7$; $x_6 < x_8$ (to eliminate symmetry in the cycle consisting of nodes $5, 6, 7, 8$).
- $x_1 < x_5$ (to prevent interchanging the two cycles).

What propagation can follow from the removal of a value from the domain of a variable in this case? The (binary) symmetry constraints are monotonic and so a value removal will propagate as in the Golomb Rulers problem. Given a clique of $\neq$ constraints, as on the node variables, removing a value from the domain of a variable does not affect the other variables. However, if there is a global allDifferent constraint, as on the edge variables, removing a value may affect other domains; for instance, if there are only as many values as variables and a value occurs in the domain of only two variables, removing it from the domain of one means that it must be assigned to the other. However, the immediate effect on the search variables (the node variables) is limited, since GAC is not maintained on the ternary constraints linking the edge and node variables.

We have tried several different variable orders for this problem. In each case, if binary branching is used with a value ordering that chooses a value from the middle of the domain, it takes the same number of backtracks to find all solutions as $k$-way branching. However, if the value chosen is either the smallest or the largest in the domain, binary branching takes fewer backtracks than $k$-way branching. This suggests that the reduction in search is mainly due to the symmetry constraints: any propagation due to the other constraints would not be restricted to the largest and smallest value in the domain.

Table 2 shows the results of solving this problem with different variable and value orders. As well as lexicographic and reverse lexicographic variable order, we use the orders that we found to be respectively best and worst, when assigning the values in increasing order [Sturdy, 2003]. (Incidentally, the comparison also demonstrates that the value order cannot compensate for a poor variable order in finding all solutions.)

It is not necessary to use the same order (increasing or decreasing) for all variables. The rightmost columns in Table 2 give the results for a heuristic that, for most variable orders, assigns the values of $x_1$ and $x_5$ in increasing order, and those of the other variables in decreasing order. However, if $x_5$ comes after $x_6$, $x_7$ and $x_8$ and before $x_1$ in the variable ordering, it is assigned in decreasing order, and if $x_1$ comes after $x_2$ to $x_8$ in the variable order, it is assigned in decreasing order. (Hence, when the variables are assigned in reverse lexicographic order, decreasing order for all variables is best.) Table 2 shows that the heuristic is at least as good as the better of increasing or decreasing order in all cases, and this was true for the other variable orders that we tried.

The heuristic was derived empirically by trying increasing or decreasing order for each variable, with a range of variable orders. We found that the effect of changing the value order for a single variable does not depend on the value order chosen for the other variables, so that the best order, for a given variable order, can be chosen independently for each variable. The heuristic can be related to the binary symmetry-breaking constraints: unless $x_1$ and $x_5$ are late in the variable order, assigning their values in ascending order will lead to reducing

| Variable order | $k$-way branching | | Increasing order | | Decreasing order | | Heuristic | |
|---|---|---|---|---|---|---|---|---|
| | bt. | sec. | bt. | sec. | bt. | sec. | bt. | sec. |
| $x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8$ | 6043 | 0.21 | 3126 | 0.21 | 3793 | 0.21 | 2929 | 0.21 |
| $x_8, x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0$ | 11963 | 0.46 | 10167 | 0.55 | 5304 | 0.37 | 5304 | 0.37 |
| $x_8, x_0, x_4, x_3, x_1, x_5, x_2, x_6, x_7$ | 3129 | 0.15 | 1839 | 0.14 | 2374 | 0.16 | 1730 | 0.14 |
| $x_1, x_7, x_5, x_3, x_2, x_8, x_0, x_6, x_4$ | 39469 | 1.57 | 31338 | 1.68 | 15871 | 1.04 | 15834 | 1.04 |

Table 2: Finding all graceful labelings of the graph in Figure 2, with $k$-way branching or binary branching with different value orderings.

the domains of other variables on backtracking. The variables assigned in decreasing order are those that appear mainly on the right of $>$ constraints; on backtracking, the largest value in the domain will be removed, and again this will reduce the domains of other variables.

## 4 Langford's Problem

In the previous examples, assigning values in either increasing or decreasing order reduces the search effort considerably, in comparison with $k$-way branching, because of the monotonic binary constraints in the problem. In this section, a problem with no monotonic constraints is discussed.

Langford's problem (prob024 in CSPLib) can be stated as follows: *"A sequence of $n \times m$ integers consists of the integers 1 to $m$ each appearing $n$ times. There is one integer between consecutive occurrences of the integer 1, two integers between consecutive occurrences of the integer 2, and so on. Find all possible such sequences."*

Modeling this problem as a CSP is discussed in [Hnich *et al.*, 2004]. A possible model has $nm$ variables, one for each location in the sequence; its value represents the integer at this location. Thus, the value $i$ of $d_j$, $1 \leq j \leq nm$, is an integer in the interval $[1, nm]$, representing the fact that occurrence $(i$ div $n)$+1 of the integer $i$ mod $n$ occurs at location $j$. For example, in the (3,9) instance, i.e. $n$ =3, $m$ = 9, $d_1 = 1, d_3 = 2, d_5 = 3$ represent the 1st, 2nd and 3rd occurrences of the integer 1 appearing in positions 1, 3 and 5 of the sequence. A dual model of the problem has a variable $x_i, 1 \leq i \leq nm$, for each occurrence of each integer: its value is the position in the sequence of this occurrence. The problem can be viewed as a permutation problem: any valid sequence assigns a permutation of the $nm$ possible values to the variables $d_1, d_2, ..., d_{nm}$. To ensure that any solution is a permutation of the values, both sets of variables are included in the model, with channeling constraints between them, i.e. $d_i = j$ iff $x_j = i$. We also add constraints to break the symmetry: given any solution, another can be found by reversing the sequence. We assign values to $d_1, d_2, ..., d_{nm}$ in turn.

A value ordering heuristic for this problem can be devised by considering how removing a value from the domain of a variable $d_j$ on backtracking will affect other variables. The removal will also remove a value from the domain of $n - 1$ other variables; for example, in the (3,9) instance, if $d_1 \neq 1$, then $d_3 \neq 2$ and $d_5 \neq 3$. However, the values 2 and 3 are only assigned as a consequence of assigning the value 1, and these deletions will not lead to further propagation. The other possible propagation is from the channeling constraints: as discussed in [Hnich *et al.*, 2004], if a value appears in the domain of only one $d_j$ variable, it will be assigned to that variable. Hence, if there is a value that appears in the domains of the current variable and only one other search variable, it should be chosen: on backtracking, propagation will assign the value to the other variable.

In [Smith, 2000], *dualsdf* ordering was used, which chooses the value appearing in fewest domains, i.e. corresponding to the dual variable with smallest domain. *dualsdf* was shown to give better results than increasing value order, for finding all solutions. It will always choose a value appearing in the domain of only one other search variable, if there is one; hence, we can now explain its good performance. Its anti-heuristic, *dualldf*, chooses the value corresponding to the dual variable with *largest* domain, and will not choose a value appearing in only two domains unless there is no alternative. Whereas *dualsdf* can be expected to give as much search reduction as possible in finding all solutions, *dualldf* will rarely be able to reduce the domains of other variables on the right branch and so should be nearly as bad as $k$-way branching for this problem. The results in Table 3 confirm this.

| Instance | *dualsdf* | *dualldf* | $k$-way branching |
|---|---|---|---|
| (9,3) | 182 | 208 | 210 |
| (10,3) | 562 | 619 | 631 |
| (11,3) | 2381 | 2697 | 2736 |

Table 3: Number of backtracks to find all solutions to three instances of Langford's problem, using binary branching with different value orders, or $k$-way branching.

Hence, *dualsdf* reduces the search effort to find all solutions compared with $k$-way branching, and its good performance can be explained in relation to to the problem constraints. However, the reduction in search is much less than in the previous examples, and does not make binary branching cost-effective in terms of run-time.

## 5 Symmetry Breaking During Search

We have shown that the effect of the value order on the search for all solutions depends on propagating the removal of the value just tried, on the right branch at binary choice points. In Symmetry Breaking During Search [Gent and Smith, 2000], constraints to avoid assignments symmetric to those already considered are added dynamically during search, on the right branch. These additional constraints can reinforce the effect of the value order on search. We demonstrate this in the $n$-

queens problem, whose symmetries are described in [Gent and Smith, 2000].

The basic program (from the ILOG Solver User Manual) has a variable $x_i, i = 1, 2, ..., n$ for each row of the board, representing the queen on that row, with a value for each column. There are three allDifferent constraints, representing that the queens are on different columns and on different diagonals in each of the two possible directions. SBDS requires a specification of the effect of each symmetry on the assignment of a value to a variable. For instance, in the $n$-queens problem, the reflection in the top-left to bottom-right diagonal (symmetry *d1*) transforms the assignment $x_i = j$ to $x_j = i$. The first assignments made are $x_1 = 1$ and $x_3 = 2$ (the variable ordering heuristic chooses the variable with smallest domain, using the smallest value as a tie-breaker). On backtracking to take the alternative choice $x_3 \neq 2$, *d1* is the only symmetry remaining, given the assignment $x_1 = 1$. SBDS adds the symmetric equivalent of $x_3 \neq 2$, i.e. $x_2 \neq 3$, to the right branch. Both constraints, i.e. $x_3 \neq 2$ and $x_2 \neq 3$, are propagated before choosing another value.

When using SBDS to eliminate the symmetry in the $n$-queens problem, we compare four value ordering heuristics. First, we select values in increasing order, as a basis for comparison. Second, we use a heuristic which seems likely to give most scope for the SBDS constraints to prune the domains of future variables. It is not obvious how best to do this, but the value that attacks the most unattacked squares will free these squares on backtracking: it seems plausible that this will increase the likelihood that the SBDS constraints remove one. The opposite heuristic, choosing the value that attacks fewest unattacked squares, is similar to the *promise* heuristic introduced by Geelen [1992]; hence, we call these two heuristics *anti-promise* and *promise* respectively. Finally, we use *dualsdf*, as in Langford's problem, taking into account that $x_1, ..., x_n$ must be assigned a permutation of the values 1 to $n$; in a permutation problem, a global allDifferent constraint does at least as much domain pruning as channeling constraints would do, when a value is removed from the domain of one of the variables [Hnich *et al.*, 2004].

We first compare the four heuristics if the symmetry is *not* eliminated: the results are given in Table 4. The effect of the value ordering then depends on how the allDifferent constraints are treated. If they are treated as cliques of binary $\neq$ constraints, the value ordering can have no effect on the search effort. Table 4 gives the results for $k$-way branching only, since the four binary branching heuristics give the same number of backtracks, though a marginally longer runtime.

On the other hand, if the allDifferent constraints are treated as global constraints, *dualsdf* requires less search than $k$-way branching, and least search of the binary branching heuristics. However, the reduction in search, in comparison with $k$-way branching, is not great. Overall, if the symmetry is not eliminated, $k$-way branching and $\neq$ constraints give the fastest runtime. Maintaining GAC on the allDifferent constraints is time-consuming, especially when GAC has to be re-established after each individual value for a variable has been tried, as in binary branching.

Table 5 compares the four heuristics when the symmetry in the problem is eliminated using SBDS. We cannot compare the heuristics with $k$-way branching in this case, because SBDS depends on binary branching. *Anti-promise*, intended to allow propagation of the SBDS constraints to reduce search, is indeed the best of the four heuristics. *Dualsdf* is often a rather poor second best, suggesting that the SBDS constraints are more significant than the allDifferent constraints in allowing domain reductions on the right branch. The experience with this problem class shows that it is possible to design value ordering heuristics for finding all solutions by considering the propagation of the constraints added on the right branch during search, whether the removal of the value just tried or SBDS constraints. The differences in search effort are relatively small, however, and the runtimes for *promise*, *anti-promise* and *dualsdf* are very similar; increasing order is fastest, being simplest to implement.

The original *promise* heuristic [Geelen, 1992] was designed to find a first solution quickly, and it is worth noting that our version is much better than *anti-promise* in this respect. For $n \leq 100$, *promise* can usually find a first solution in just a few backtracks, whereas *anti-promise* becomes increasingly successful at avoiding solutions. For instance, when $n = 100$, *promise* finds a solution in 2 backtracks, whereas *anti-promise* takes more than 21 million. However, *promise* is slightly worse than *anti-promise* for finding all solutions.

|  | $\neq$ constraints |  | GAC on allDifferent constraints |  |  |  |
|---|---|---|---|---|---|---|
|  | $k$-way branching |  | *dualsdf* |  | $k$-way branching |  |
| $n$ | bt. | sec. | bt. | sec. | bt. | sec. |
| 9 | 1111 | 0.04 | 854 | 0.03 | 858 | 0.03 |
| 10 | 5072 | 0.11 | 3831 | 0.14 | 3869 | 0.12 |
| 11 | 22124 | 0.41 | 16308 | 0.63 | 16478 | 0.54 |
| 12 | 103956 | 1.90 | 74514 | 3.05 | 75448 | 2.62 |
| 13 | 531401 | 9.79 | 366488 | 15.6 | 371298 | 13.5 |
| 14 | 2932626 | 53.9 | 1964642 | 84.8 | 1990925 | 73.8 |

Table 4: Number of backtracks (bt.) and runtime (on a 1.7GHz Pentium M PC) to find all solutions to the $n$-queens problem, if symmetry is not eliminated.

| $n$ | Increasing order | *anti-promise* | *promise* | *dualsdf* |
|---|---|---|---|---|
| 9 | 144 | 144 | 158 | 148 |
| 10 | 649 | 617 | 674 | 647 |
| 11 | 2433 | 2375 | 2375 | 2384 |
| 12 | 11305 | 10872 | 11547 | 11252 |
| 13 | 52140 | 51045 | 52825 | 51452 |
| 14 | 292869 | 279844 | 295522 | 291825 |
| 15 | 1562199 | 1521670 | 1541946 | 1547019 |

Table 5: Number of backtracks to find all non-isomorphic solutions to the $n$-queens problem, using different value ordering heuristics.

## 6 Conclusions

Mitchell [2003] showed that binary branching can do much less search than $k$-way branching, due to the possibility that on backtracking to a choice point, the next assignment tried need not be another value (if there is one) of the same variable. In practice, this is not provided by default in solvers such as ECL$^i$PS$^e$ and ILOG Solver, although the user can implement such a search strategy, and Sabin and Freuder [1997] allowed it in their description of the MAC algorithm.

We have shown that even when the search always assigns another value of the current variable on backtracking, binary branching offers another potential advantage. Before another value is assigned, the removal of the value just tried from the domain of the variable is propagated. This can lead to further domain reductions, which can in turn mean that a future value of the current variable is pruned, or that the search when a future value is assigned is reduced. As a result, given the same variable ordering, binary branching does no more search than $k$-way branching in finding all solutions, and may do considerably less, depending on the order in which values are tried.

The reduction in search effort is much less than Mitchell found, since if all the values of the current variable are tried in turn, the propagation of the removal of the value just tried will be subsumed by the propagation of the assignment of another value. Nevertheless, we have shown that binary branching can result in a 50% reduction in backtracks over $k$-way branching, with a reduction of around 30% in runtime.

We found the largest search savings in the problems with monotonic binary constraints, when assigning the values in increasing or decreasing order. A reasonable value ordering heuristic in that case seems to be that if a variable is constrained to be less than several other variables, and is assigned before them, its values should be tried in increasing order, and otherwise in decreasing order. In that way, on backtracking, removing the value just tried means reducing the range of the variable, and this propagates to the variables it constrains. Furthermore, propagating range reductions in monotonic binary constraints is cheap, so that the additional overhead in propagating the removal of a value before the assignment of the next is not large.

In other cases, although it is possible to choose good value orderings by considering how the problem constraints would propagate the removal of a value from the domain of a variable, the reduction in search effort is much less, and $k$-way branching is faster. The experience with these problems suggests that constraint propagation must be cheap and further domain reductions must be very likely in order for binary branching to be worthwhile. For instance, enforcing GAC on an allDifferent constraint fails on both counts: it is time-consuming, and removing a value from the domain of a single variable only occasionally leads to further domain reductions. As shown in the $n$-queens problem, when finding all solutions without breaking symmetry, it is better to use $k$-way branching than binary branching, if using global allDifferent constraints, or better still to use $k$-way branching with $\neq$ constraints.

Binary branching, with *any* value ordering, does no more search than $k$-way branching in finding all solutions, or proving that there are no solutions. However, even when binary branching does less search, $k$-way branching can be faster. Propagating the effect of removing the value just tried from the domain of the current variable can be time-consuming, and may not give sufficient reduction in search to be worthwhile. Our investigations have been concerned with finding all solutions; however, when the extra propagation done by binary branching is not worthwhile for finding all solutions, it is unlikely to be worthwhile for finding one solution either. Although binary branching on successive values of the current variable is the default search strategy in some constraint solvers, $k$-way branching may often be a better choice.

## Acknowledgements

We thank Gene Freuder, Deepak Mehta, Barry O'Sullivan and Nic Wilson for their helpful comments, and Peter van Beek for providing $k$-way branching code. This work was supported in part by the Science Foundation Ireland under Grant No. 00/PI.1/C075.

## References

[Frost and Dechter, 1995] D. Frost and R. Dechter. Look-ahead value ordering for constraint satisfaction problems. In *Proceedings IJCAI95*, pages 572–578, 1995.

[Geelen, 1992] P. A. Geelen. Dual Viewpoint Heuristics for Binary Constraint Satisfaction Problems. In B. Neumann, editor, *Proceedings ECAI'92*, pages 31–35, 1992.

[Gent and Smith, 2000] I. P. Gent and B. M. Smith. Symmetry Breaking During Search in Constraint Programming. In W. Horn, editor, *Proceedings ECAI'2000*, pages 599–603, 2000.

[Hnich *et al.*, 2004] B. Hnich, B. M. Smith, and T. Walsh. Dual Models of Permutation and Injection Problems. *JAIR*, 21:357–391, 2004.

[Mitchell, 2003] D. G. Mitchell. Resolution and constraint satisfaction. In F. Rossi, editor, *Principles and Practice of Constraint Programming - CP 2003*, LNCS 2833, pages 555–569. Springer, 2003.

[Petrie and Smith, 2003] K. E. Petrie and B. M. Smith. Symmetry Breaking in Graceful Graphs. Technical Report APES-56a-2003, APES Research Group, June 2003.

[Sabin and Freuder, 1997] D. Sabin and E. C. Freuder. Understanding and Improving the MAC Algorithm. In G. Smolka, editor, *Principles and Practice of Constraint Programming - CP97*, LNCS 1330, pages 167–181. Springer, 1997.

[Smith *et al.*, 2000] B. M. Smith, K. Stergiou, and T. Walsh. Using auxiliary variables and implied constraints to model non-binary problems. In *Proceedings AAAI-2000*, pages 182–187, 2000.

[Smith, 2000] B. M. Smith. Modelling a Permutation Problem. Research Report 2000.18, School of Computer Studies, University of Leeds, June 2000.

[Sturdy, 2003] P. Sturdy. Learning Good Variable Orderings. Technical Report APES-64-2003, APES Research Group, June 2003. Available from http://www.dcs.st-and.ac.uk/~apes/apesreports.html.

[van Hentenryck *et al.*, 1992] P. van Hentenryck, Y. Deville, and C.-M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.