Comma-Free Codes

Justin Pearson

Department of Information Technology
Division of Computer Systems
Uppsala University, Box 337, 751 05 Uppsala, Sweden
justin@docs.uu.se

Abstract. In this short paper Comma-Free Codes are introduced and symmetry breaking techniques are applied to their search. Comma-Free Codes have a lot of symmetry: both in the values and in the variables. Symmetry Breaking via Dominance Detection is applied to find Comma-Free Codes. To check if a current partial solution is symmetrically equivalent to a previously found no-good, graph isomorphism is used. In particular the powerful and fast graph isomorphism package `nauty` is used. Experimental results show that for difficult instances SBDD+Nauty out performs lexicographic ordering.

1 Comma-Free Codes

A *comma-free code* [7–9] over an alphabet A is a set, $C \subseteq A^*$, of words over A such that given any two words, $w, v \in C$, any sub-word, u, of the concatenation, wv, is not in the code. Here we will be only interested in codes where all the words have the same length. See figure 1 for an example of a comma-free code.

00001 00101 00110 11001 11010 11110

Fig. 1. A Comma-Free Code of 6 words of length 5 over the alphabet $\{0, 1\}$

Comma-Free Codes were originally inspired by biology. Genetic information in the cell is stored in DNA, which for a computer scientist is a string of letters from the alphabet A, C, T and G. Via a transcription mechanism proteins are constructed which are chains of amino acids from an alphabet of 20 different acids. One question that exercised biologists was: How are each of the 20 different acids coded as strings of DNA? One proposal is that the 20 acids are coded as comma-free codes, so as to aid transcription. As it turns out there is a comma-free code of size 20 of words of length 3 over a four letter alphabet. But in reality nature is not so simple: the genetic code is not a comma-free code and in fact is not even a code in the mathematical sense. Figure 2 shows a comma free code over alphabet size 4 with words of length 3.

Comma-Free Codes are still interesting from a mathematical point of view. One property of interest in the theory of comma-free codes is: given an alphabet

112 113 114 212 213 214 223 224 312 313
314 323 324 334 412 413 414 423 424 434

Fig. 2. Comma-Free code with word length 3 over an alphabet of size 4.

size and a word length what is the maximal number of words a comma-free code can contain?

If a word, $w = w_1 \dots w_k$, belongs to a comma-free code then any sub-word of

$$w_1 \dots w_k w_1 \dots w_k$$

should not be in the code. Two words are said to be in the same *cyclic equivalence class* if one is a cyclic permutation of the other. If a cyclic equivalence class of a word contains k members (where k is the word length) then it is called *complete*. Words from incomplete equivalence classes cannot be in comma-free codes (see [8] and the references therein for details). Thus an upper bound on the size of comma-free codes can be found by counting the number of complete classes. Let $W_k(n)$ denote the maximum number of words of length k that a code can contain over an alphabet of size n. The bound

$$W_k(n) \leq \frac{1}{k} \sum_{d|k} \mu(d) n^{k/d}$$

is derived in [7, 8] where μ is the Möbius function defined by:

$$\mu(d) = \begin{cases} 1 & \text{if } d = 1 \\ 0 & \text{if } d \text{ has any square factor} \\ (-1)^r & \text{if } d = p_1 p_2 \dots p_r \\ & \text{where } p_1, \dots, p_r \text{ are distinct primes.} \end{cases}$$

This bound is not always achieved.

2 Symmetry Breaking via Dominance Detection (SBDD)

Essentially SBDD [2] prevents symmetrically equivalent no-goods being explored. In more detail, given a failure during search (where a failure is some partial assignment that cannot be extended) the search procedure should guarantee that no symmetrically equivalent partial assignment is ever explored. Symmetry breaking during search (SBDS [6]) achieves this by adding symmetric no-goods: that is given some partial assignment for each symmetrically equivalent partial assignment a constraint is added forbidding that symmetrically partial assignment. In SBDD, on the other hand, instead of adding no-goods an extra procedure is added to the search procedure which checks if the current partial assignment has already been seen before (dominated) as a no-good.

One way of implementing SBDD is by maintaining a database of previously seen partial assignments and checking the current assignment against all the previous partial assignments for equality modulo symmetry.

Various modifications can be applied to SBDD in reducing the number of no-goods that need to be stored. In particular during depth-first search no-goods below a completed node in the search tree can be removed.

In implementing a search procedure for comma-free codes each no-good and partial assignment will be converted into a graph. In implementing SBDD with the optimisation of removing redundant no-goods the dominance checking procedure has to check if the graph of the previously found no-good is isomorphic to a subgraph of the current no-good. This procedure is of course NP-complete [4]. In the implementation of SBDD used in this paper time is traded for space. All the no-goods are kept, none are thrown away as would be done with an optimised implementation of SBDD. This allows graph isomorphism to be used as a dominance check since only no-goods at the same level in the search tree are checked. The complexity of graph isomorphism is not known, but in practice it is often polynomial.

3 Comma-Free Codes: Graph Isomorphism and Symmetry Breaking

To model and find comma-free codes using a CSP, a code is represented as a list of lists where each list represents a word. There are two obvious symmetries with this representation. First, the order of the words does not matter: that is any permutation of the words is still a comma-free code. Secondly given any comma-free code, C, over an alphabet A and any bijection, $f : A \to A$, on the alphabet, applying the bijection to every word in the code (giving the code $\{f(w)|w \in C\}$ where $f(w) = f(w_1) \cdots f(w_k)$ when $w = w_1 \cdots w_k$) is still a comma-free code.

One way of combating the first symmetry is to order the code words lexicographically using the `lex-chain` [1] global constraint.

The second symmetry (the value symmetry) is harder to break and even harder to break in the presence of the first symmetry. One such approach would be to reformulate the problem as a matrix model and use two lexicographic chain constraints (see [3]) but this would still not break all the symmetry in the problem.

The search procedure implemented to solve the CSP searches for a complete word at a time in the code. To check if the current partial assignment is symmetrically equivalent to a previously found no-good at the same level graphs are constructed for both the partial assignment and the no-good such that the graphs are isomorphic if and only if the assignments are symmetrically equivalent. Isomorphism of the graphs is checked using the `nauty` [10] system, which is able to return a canonical graph such that two graphs are isomorphic if and only if they have the same canonical graph. Hence if the no-goods are stored as `nauty` canonical graphs then a new partial assignment can be converted into a canonical graph and checked against stored no-goods.

The graph used for isomorphism testing is constructed as follows. Given a partial assignment of n code words of the form:

$$\begin{aligned} W_1 &= w_1^1 \dots w_1^k \\ &\vdots \\ W_n &= w_n^1 \dots w_1^k \end{aligned}$$

where each w_i^j is a letter from the alphabet of the code (because of the way the search procedure works there will be no code words to level n which have only some entries grounded), then a coloured graph with two colours is constructed as follows:

- The set of nodes of the graph is the disjoint union of the sets A (the domain or the alphabet of the code) and the set $\{w_i^j | 1 \leq i \leq n \wedge 1 \leq j \leq k\}$;
- There are two colours of the graph: the nodes in A being one and the nodes w_i^j being the other;
- For every graph, regardless of the code, there are edges from w_i^j to w_i^{j+1} for all $1 \leq i \leq n$ and all $1 \leq j < k$;
- For all $1 \leq i \leq n$ and $1 \leq j \leq k$ if $w_i^j = d$ then there is an edge from d to w_i^j.

In figure 3 an example graph is given for the code $\{001, 101\}$. It is then possible

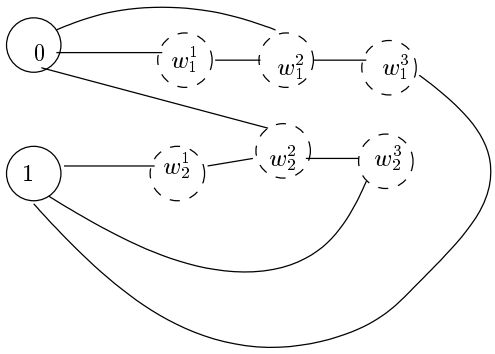

Fig. 3. Coloured graph for the code $\{001, 101\}$

to prove that two codes of the same length are symmetrically equivalent (both in value symmetry and the ordering of the words) if the two graphs are coloured graph isomorphic. Such an isomorphism gives a bijection on the code words. The separation of the colours gives a bijection on the domain elements. The edges between w_i^j and w_i^{j+1} forces any isomorphism f such that

$$f(w_i^j) = w_{i'}^{j'}$$

forces $f(w_i^{j+1})$ to be $w_{i'}^{j'+1}$. Finally edges between the domain elements and the nodes w_i^j force the isomorphism on domain elements to be an isomorphism of the code words.

4 Results

An implementation of SBDD with graph isomorphism was compared against lexicographically ordering the words. Although lexicographic ordering will not break all the symmetry (that is both the value symmetry and the interchangeability of the words in the code) in many cases it performs well. It is not until relatively difficult instances that SBDD with graph isomorphism competes with lexicographic ordering.

In figure 4 results are presented for codes of words of length 3 over a domain of size 4 using Sicstus Prolog on a Pentium 4 2.53Ghz machine (with 512Mb of memory) running Linux; note that in this case 20 is the maximal code length. The search looks for one code satisfying the constraints. The number of backtracks does not refer to the total number of backtracks, but the backtracks at the word level. At each level in the search tree a complete word is found, thus on a backtrack a new code word is found. To find each code word at each level in the tree Sicstus' normal labelling procedure is used. Figure 6 shows the Sicstus code used without any symmetry breaking; it is the backtracks of `labeling_cmp` that are reported in figures 4 and 5. After code length 16, SBDD with graph isomorphism wins. Also in figure 5 results are shown for code words of length 3 over an alphabet of size 5.

Code Length	Time: SBDD	Backtracks	Time: lex	Backtracks
13	1.17	34	0.84	18
14	4.4	96	2.41	98
15	259.09	3145	159.29	4198
16	328.37	3145	208.6	4198
17	1891.27	13608	3729.96	57680
18	2272.76	13608	4591.89	57680
19	2741	13608	5768.78	57680
20	3234.31	13608	7247.42	57680

Fig. 4. Results on domain size 4, word length 3, all times reported in seconds.

5 Conclusion

Although it might seem that losing the optimisations possible with SBDD on a depth-first search requires many no-goods to be stored and checked, by using graph isomorphism these no-goods can be checked relatively quickly. Also since the symmetry groups in general would be large and constructing the graphs is

Code Length	Time: SBDD	Backtracks	Time: lex	Backtracks
22	36.02	704	33.58	1092
23	40.16	704	40.43	1092
24	44.74	704	47.74	1092
25	47.93	704	54.07	1092
26	841.82	2926	878.23	10827

Fig. 5. Results: Domain size 5, word length 3.

```
%labeling_cmp(Words) Words is a list of lists, that is a list of
%code words.
    labeling_cmp([]).
    labeling_cmp([W|T]) :-
%labeling is a call to Sicstus finite domain labelling  procedure on
%the code word W
        labeling([],W) ,
        labeling_cmp(T).
```

Fig. 6. Sicstus code used for labelling without any additional symmetry breaking

quite simple this method avoids generating many no-goods as would be done with SBDS or its optimisations [5].

The technique of using graph isomorphism could also be applied to Balanced Incomplete Block Designs and the Social Golfer and this is work in progress. This work was partially supported by a STINT institutional grant IG2001-67 of STINT the grant 221-99-369 of the Swedish Research Council.

References

1. Mats Carlsson and Nicolas Beldiceanu. Arc-consistency for a chain of lexicographic ordering constraints. Technical Report T2002-18, Swedish Institute of Computer Science, 2002.
2. T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In T. Walsh, editor, *Proceedings of CP'01*, volume 2293 of *LNCS*, pages 93–107. Springer-Verlag, 2001.
3. P. Flener, A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In P. Van Hentenryck, editor, *Proceedings of CP'02*, volume 2470 of *LNCS*, pages 462–476. Springer-Verlag, 2002.
4. Michael R. Garey and David S. Johnson. *Computers and Intractability.* W.H. Freeman and Company, 1979.
5. Ian P. Gent, Warwick Harvey, and Tom Kelsey. Groups and constraints: Symmetry breaking during search. In P. Van Hentenryck, editor, *Proceedings of CP'02*, volume 1520 of *LNCS*, pages 415–430. Springer-Verlag, 2002.
6. I.P. Gent and B.M. Smith. Symmetry breaking during search in constraint programming. In *Proceedings of ECAI'00*, pages 599–603, 2000.

7. S.W. Golomb and L.R. Welch. Comma-free codes. *Candian Journal of Mathematics*, 10:202–209, 1958.
8. B.H. Jiggs. Recent results in comma-free codes. *Canadian Journal of Mathematics*, 15:178–187, 1963.
9. Nguyen Huong Lam. Completing comma-free codes. *Theoretical Computer Science*, 301:399–415, 2003.
10. Brendan McKay. `nauty` user's guide (version 2.2). Available via `http://cs.anu.edu.au/people/bdm/nauty/`.