An Alternative Eager Encoding of the All-Different Constraint over Bit-Vectors

Pavel Surynek [1,2]

Abstract. A novel eager encoding of the ALLDIFFERENT constraint over bit-vectors is presented in this short paper. It is based on 1-to-1 mapping of the input bit-vectors to a linearly ordered set of auxiliary bit-vectors. Experiments with four SAT solvers showed that the new encoding can be solved order of magnitudes faster than the standard encoding in a hard unsatisfiable case.

1 INTRODUCTION AND MOTIVATION

Models of many real-life problems require a subset of modeling variables to be pair-wise distinct. This requirement is known as an ALLDIFFERENT constraint [5] in the constraint programming context. As the SAT solving technology [1], [3], [6] is becoming a tool of choice in many practical applications, efficient manipulation with the ALLDIFFERENT constraint in SAT solvers is of interest. Unlike other works on translating the ALLDIFFERENT constraint into SAT that use direct encoding of variable's domains [4], we study how to encode the constraint over the set of bit-vectors which essentially use binary encoding. We present a new eager encoding that maps the given set of bit-vectors to a linearly ordered set of auxiliary bit-vectors. We show that the new encoding is more efficient for hard unsatisfiable cases of the constraint on which SAT solvers struggle with the existent encoding for bit-vectors [2].

2 BACKGROUND – STANDARD MODEL

Suppose to have a set of bit-vectors $\{\mathcal{B}^1, \mathcal{B}^2, \ldots, \mathcal{B}^n\}$ each of length l. Bit-vectors are interpreted as non-negative integers. The ALLDIFFERENT constraint over $\mathcal{B}^1, \mathcal{B}^2, \ldots, \mathcal{B}^n$ - denoted as ALLDIFFERENT($\{\mathcal{B}^1, \mathcal{B}^2, \ldots, \mathcal{B}^n\}$) - requires that numbers represented by the bit-vectors are all distinct. The standard encoding [2] basically follows the scheme where pair-wise differences are encoded:

$$\text{ALLDIFFERENT}(\{\mathcal{B}^1, \mathcal{B}^2, \ldots, \mathcal{B}^n\}) \equiv \bigwedge_{i,j=1 \wedge i<j}^{n} \mathcal{B}^i \neq \mathcal{B}^j$$

Trivially it is possible to encode the individual inequalities as follows. Let the $\mathbb{i}$-th bit of the k-th bit-vector with $\mathbb{i} \in \{1,2,\ldots,l\}$ and $k \in \{1,2,\ldots,n\}$ be denoted as $\mathcal{B}^k_{\mathbb{i}}$.

$$\mathcal{B}^i \neq \mathcal{B}^j \equiv \bigvee_{\mathbb{i}=1}^{l} \left(\neg\mathcal{B}^i_{\mathbb{i}} \vee \mathcal{B}^j_{\mathbb{i}}\right) \wedge \left(\mathcal{B}^i_{\mathbb{i}} \vee \neg\mathcal{B}^j_{\mathbb{i}}\right)$$

However, if unfolded into the CNF representation though the distributive rule it results into too many clauses which is impractical. Therefore encoding using auxiliary propositional variables is used. It follows the standard technique of Tseitin's hierarchical encoding. A fresh propositional variable is introduced for each inequality between individual bits of the involved bit-vectors. That is, there is a new variable $a^{i,j}_{\mathbb{i}}$ for every $i, j \in \{1,2,\ldots,n\}$ with $i < j$ and $\mathbb{i} \in \{1,2,\ldots,l\}$. The auxiliary variable indicates if the corresponding bits in the inequality between bit-vectors differ or not. Thus, the following clauses are included to express this interpretation:

$$\bigwedge_{\mathbb{i}=1}^{l} \left(\neg a^{i,j}_{\mathbb{i}} \vee \mathcal{B}^i_{\mathbb{i}} \vee \mathcal{B}^j_{\mathbb{i}}\right) \wedge \left(\neg a^{i,j}_{\mathbb{i}} \vee \neg\mathcal{B}^i_{\mathbb{i}} \vee \neg\mathcal{B}^j_{\mathbb{i}}\right)$$

Bit-vectors $\mathcal{B}^i$ and $\mathcal{B}^j$ differ if they differ in at least one position; that is, following clauses should be included: $\bigvee_{\mathbb{i}=1}^{l} a^{i,j}_{\mathbb{i}}$. Notice that auxiliary variables are linked to the original bits only in one direction. If $a^{i,j}_{\mathbb{i}}$ is set to $TRUE$ then $\mathcal{B}^i_{\mathbb{i}}$ and $\mathcal{B}^j_{\mathbb{i}}$ are forced to differ. However, if $a^{i,j}_{\mathbb{i}}$ is $FALSE$ then $\mathcal{B}^i_{\mathbb{i}}$ and $\mathcal{B}^j_{\mathbb{i}}$ are left unconstrained.

Proposition 1 (STANDARD ENCODING SIZE). *The standard encoding of the* ALLDIFFERENT *constraint requires* $l \cdot n$ *propositonal variables to represent the bit-vectors and* $l \cdot \frac{n(n+1)}{2}$ *auxiliary propositional variables; that is,* $\mathcal{O}(l \cdot n^2)$ *variables altogether. The number of clauses is* $1 + l \cdot n(n+1)$*; that is,* $\mathcal{O}(l \cdot n^2)$. ■

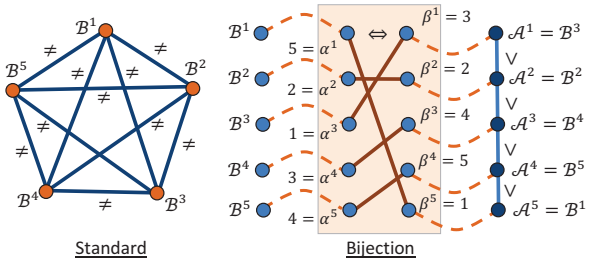

Figure 1. *Illustration of the standard and the bijection* ALLDIFFERENT *encodings.* In the bijection encoding, a 1-to-1 mapping of the bit-vectors is found first. Then the values of bit-vectors are forced to be linearly ordered according to their position in the mapping.

3 ALTERNATIVE BIJECTION ENCODING

We observed that a SAT solver struggles over the standard encoding especially in the unsatisfiable case according to our preliminary experiments. Therefore we developed an alternative encoding that is more suitable for this case. It maps the original bit-vectors to a linearly ordered set of auxiliary bit-vectors. First, a 1-to-1 mapping (*bijection*) between sets of bit-vectors needs to be modeled to enable this encoding style (see Figure 1 for illustration).

[1] Charles University in Prague, Malostranské náměstí 2/25, 118 00 Praha, Czech Republic, email: pavel.surynek@mff.cuni.cz
[2] Kobe University, 5-1-1 Fukae-minamimachi, Higashinada-ku, Kobe 658-0022, Japan.

This work is supported by the Japan Society for the Promotion of Science (contract no. P11743) and by the Czech Science Foundation (contract no. GAP103/10/1287).

Let the new linearly ordered bit-vectors be denoted as $\mathcal{A}^1$, $\mathcal{A}^2, \ldots, \mathcal{A}^n$. Additionally bit-vectors $\alpha^1, \alpha^2, \ldots, \alpha^n$ and $\beta^1, \beta^2, \ldots$, β^n of size $\lceil \log_2 n \rceil$ are introduced to model the bijection. The bit-vector α^k indicates what $\mathcal{A}^i$ with $i \in \{1,2,\ldots,n\}$ the original $\mathcal{B}^k$ will be mapped to. Bit-vectors β^j are used to enforce that at most one original bit-vector is mapped to a single ordered bit-vector. The following integer constraints are to establish the bijection:

$$\bigwedge_{i,k=1}^{n} \alpha^k = i \implies \mathcal{B}^k = \mathcal{A}^i \wedge \beta^i = k$$

It is crucial, that domains of bit-vectors α^k and β^j consists of exactly n values to ensure that the bijection is modeled correctly (extra values are forbidden). The individual integer implication is encoded with a single auxiliary propositional variable e_i^k as follows:

$$e_i^k \vee \bigvee_{\mathbb{i}=1}^{\lceil \log_2 n \rceil} \neg \mathrm{lit}(i_{\mathbb{i}}, \alpha_{\mathbb{i}}^k) \quad \text{where} \quad \mathrm{lit}(i_{\mathbb{i}}, \alpha_{\mathbb{i}}^k) = \begin{cases} \alpha_{\mathbb{i}}^k & \text{iff } i_{\mathbb{i}} = 1 \\ \neg\alpha_{\mathbb{i}}^k & \text{iff } i_{\mathbb{i}} = 0 \end{cases}$$

$$\bigwedge_{\mathbb{i}=1}^{l} \left(\neg e_i^k \vee \neg \mathcal{B}_{\mathbb{i}}^k \vee \mathcal{A}_{\mathbb{i}}^i\right) \wedge \left(\neg e_i^k \vee \mathcal{B}_{\mathbb{i}}^k \vee \neg \mathcal{A}_{\mathbb{i}}^i\right)$$

$$\bigwedge_{\mathbb{i}=1}^{\lceil \log_2 n \rceil} \neg e_i^k \vee \mathrm{lit}(k_{\mathbb{i}}, \beta_{\mathbb{i}}^i)$$

Finally there are integer constraints enforcing the ordering:

$$\bigwedge_{i=1}^{n-1} \mathcal{A}^i < \mathcal{A}^{i+1}$$

The individual inequality is encoded as a strict lexicographic ordering over the two bit-vectors. Now, l fresh propositional variables $\not\mathbb{i}_{\mathbb{i}}^i$ with $\mathbb{i} \in \{1,2,\ldots,l\}$ are introduced to indicate the first bit where $\mathcal{A}^i$ is less than $\mathcal{A}^{i+1}$. The ordering itself then just means that there exists such a first bit where bit-vectors differ: $\bigvee_{\mathbb{i}=1}^{l} \not\mathbb{i}_{\mathbb{i}}^i$.

$$\bigwedge_{\mathbb{j}=1}^{\mathbb{i}-1} \left(\neg \not\mathbb{i}_{\mathbb{i}}^i \vee \neg \mathcal{A}_{\mathbb{j}}^i \vee \mathcal{A}_{\mathbb{j}}^{i+1}\right) \wedge \left(\neg \not\mathbb{i}_{\mathbb{i}}^i \vee \mathcal{A}_{\mathbb{j}}^i \vee \neg \mathcal{A}_{\mathbb{j}}^{i+1}\right)$$

$$\left(\neg \not\mathbb{i}_{\mathbb{i}}^i \vee \neg \mathcal{A}_{\mathbb{i}}^i\right) \wedge \left(\mathcal{A}_{\mathbb{i}}^{i+1} \vee \neg \not\mathbb{i}_{\mathbb{i}}^i\right)$$

Proposition 2 (BIJECTION ENCODING SIZE). *The bijection encoding requires* $2l \cdot n$ *propositional variables to represent the bit-vectors,* $2n\lceil log_2 n \rceil$ *variables to represent the bijection, and* $n^2 + l(n-1)$ *auxiliary propositional variables; that is* $\mathcal{O}(n \cdot max\{n, l\})$ *propositional variables altogether.*

The number of clauses is $n^2(1 + l + \lceil \log_2 n \rceil) + (n-1)\frac{2l(l+1)}{2}$; *that is,* $\mathcal{O}(n^2 \cdot max\{\lceil \log_2 n \rceil, l\} + n \cdot l^2)$. ■

Table 1. Comparison of sizes of the standard and the bijection encoding.

#bit-vectors (16-bits)	Standard		Bijection	
	#Variables	#Clauses	#Variables	#Clauses
64	67584	133056	9968	176943
128	266240	536448	28400	690031
256	1056768	2154240	90096	2756591

4 EXPERIMENTAL EVALUATION

As shown in Table 1, the bijection encoding has fewer variables while the number of clauses is slightly higher than in the standard encoding. Nevertheless, we also need runtime comparison. A setup where a *transition-phase* behavior was observed is presented. We used 32 bit-vectors consisting of 6 bits. Additionally, there was a lower bound and an upper bound per each bit-vector. If $d \in \mathbb{N}$, $d \leq 34$ is a given domain size, then the lower bound $b_L^k \in \mathbb{N}$ and the upper bound $b_U^k \in \mathbb{N}$ for the bit-vector $\mathcal{B}^k$ were generated randomly as follows: b_L^k was selected uniformly from $[0..34-d]$ and b_U was set to $b_U^k + d$. Thus, $b_L^k \leq \mathcal{B}^k \leq b_U^k$ is enforced for each k. Finally, a single ALLDIFFERENT over 32 bit-vectors was added.

Three SAT solvers were used in the evaluation: MINISAT [3], GLUCOSE [1], and CRYPTOMINISAT [6]. The runtime was measured for different domain sizes d ranging from 2 to 34 - Figure 2. For small d unsatisfiability could be checked easily; for large d the same could be done for satisfiability. The most interesting behavior occurred around $d = 13$ which represent difficult cases.

None of the tested SAT solvers was able to solve all the instances over the standard encoding in the time limit of 1 hour (wall clock limit per instance). The best performing over the standard encoding was GLUCOSE which solved 29 instances out of 33 and was also the fastest. Over the bijection encoding, MINISAT and CRYPTOMINISAT solved all the instances and very importantly the runtime of CRYPTOMINISAT was always below 2 seconds. GLUCOSE also performed relatively well compared to the standard encoding with 30 solved instances.

Generally, the standard encoding can be solved faster in the satisfiable case. However, the bijection encoding is significantly better in the hard unsatisfiable case. This is because it can be checked more easily for this encoding if there are enough values in domains of bit-vectors to establish the required pair-wise difference (at least by some SAT solvers). A single linearly ordered set of bit-vectors is matched into the domains while in case of the standard encoding all the orderings (permutations) of the original bit-vectors may be checked.

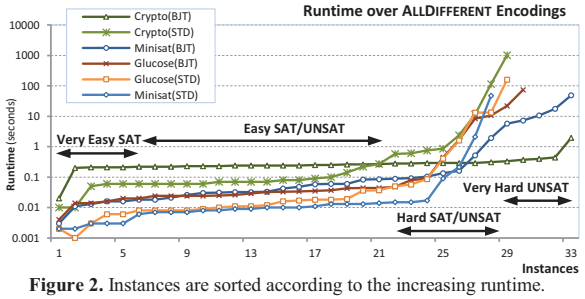

Figure 2. Instances are sorted according to the increasing runtime.

5 CONCLUSION

A new encoding for the ALLDIFFERENT constraint over bit-vectors based on 1-to-1 mapping has been proposed. It has fewer variables and it is more efficient in difficult unsatisfiable cases than the existent encoding [2] that uses pair-wise differences. In the future work, it would be also interesting to investigate how the presented eager encodings performs with respect to the strong ALLDIFFERENT propagators [5] integrated with the solver lazily via the SMT framework and also how it performs in applications.

REFERENCES

[1] G. Audemard, L. Simon, 'Predicting Learnt Clauses Quality in Modern SAT Solver', *Proceedings of IJCAI 2009*, (2009).

[2] A. Biere, R. Brummayer, 'Consistency Checking of All Different Constraints over Bit-Vectors within a SAT Solver', *Proceedings of FMCAD 2008*, 1-4, (2008).

[3] N. Eén, N. Sörensson, 'An Extensible SAT-solver', *Proceedings of SAT 2003*, 502-518, (2003).

[4] P. Nightingale, I. Gent, 'A New Encoding of AllDifferent into SAT', *CP 2004 Workshop on Modelling and Reformulating CSPs*, (2004).

[5] J.-C. Régin, 'A Filtering Algorithm for Constraints of Difference in CSPs', *Proceedings of AAAI 1994*, 362-367, (1994).

[6] M. Soos, K. Nohl, C. Castelluccia, 'Extending SAT Solvers to Cryptographic Problems', *Proceedings of SAT 2009*, 244-257, (2009).