

Quadratic Direct Encoding vs. Linear Order Encoding

A One-out-of-N Transformation on CNF

Norbert Manthey and Peter Steinke

Knowledge Representation and Reasoning Group
Technische Universität Dresden, 01062 Dresden, Germany
`norbert@janeway.inf.tu-dresden.de`

Abstract. The translation from finite CSPs into SAT has been studied intensively. Common encodings for variable domains are the compact, direct and order encoding. The direct encoding needs a quadratic number of clauses for encoding a domain in SAT whereas the order encoding uses only linearly many. We introduce a conversion for domains from the direct encoding to the order encoding on the CNF level by extracting the encoded domains and replacing the related clauses by the order encoding and a mapping between them. The transformation keeps the model for the original formula and transforming domains with more than 8 elements results in less clauses in the formula. Experiments showed that our SAT solver *riss* can solve the transformed instances with domains greater than 7 from the SAT Competition 2009 Application with 3.4 % less run time. On 105 crafted instances *riss* could solve 55 instead of 58 instances, but the average run time decreased by 30 %. Results for PrecoSAT, clasp and march_hi will be presented as well, showing that the transformation boosts the performance of CDCL solvers.

1 Introduction

The satisfiability problem (SAT) and the constraint satisfaction problem (CSP) are two well known and intensely studied problems in computer science. There are numerous applications for both problems, e.g. bio informatics [11] or scheduling [2] for SAT and configuration problems [22] for CSP. Recent successes on SAT solver is based on the algorithmic level, like conflict-directed clause learning [17] (CDCL), on the implementation, like the two-watched-literal unit propagation [14] and cache utilization [10], as well as on new studies of heuristics [1]. Annual competitions push the development of solvers further. Consequently, developers of CSP solvers also encode CSPs into SAT and use SAT solvers to solve their problems (e.g. [18,6]).

Encoding a CSP into SAT can be realized in many ways. The main difference between encodings is the way how a domain of a CSP variable is encoded into propositional variables. The *direct encoding* [21] uses one propositional variable per possible assignment for the CSP variable. For assigning a value to the CSP

domain at least one of the according propositional variables and at most one variable has to be satisfied. The latter part requires a quadratic number of clauses. On the CNF level the encoded domain has the same properties as a one-out-of-N constraint, which is a special case of cardinality constraints in SAT. The *order encoding* [19] also needs a propositional variable for each CSP domain value, but these variables represent different information. The domain is assumed to be ordered so that a variable p_i states that the corresponding CSP variable has at least the value of the i -th element of its domain. To encode this information, only linearly many clauses are necessary. Additionally, the first propositional variable has always to be assigned *true*, because the CSP variable is assigned at least the smallest value of the domain. It is not necessary to encode this variable. Thus, only $n-1$ variables are necessary to encode the same domain with the order encoding. There are also hybrid encodings that encode a domain in both ways and a mapping between the two encodings, making it more easy to convert a CSP in CNF [16,6].

As far as we know, the direct encoding is most frequently used, although the order encoding seems to be more beneficial [19]. We are not aware of a transformation on the CNF level from the direct encoding to the order encoding. Hence, the contribution of this paper is exactly this transformation. We present an algorithm that extracts CSP domains out of a CNF and show that the used clauses can be replaced by new clauses that encode the same domain with the order encoding. The introduction of new variables for the order encoding requires also to add a mapping from the order encoding back to the direct encoding. Since the number of clauses of the order encoding and the mapping is linear in the size of the domain and the direct encoding requires quadratically many clauses, the transformation results in a new formula with less clauses, if the domain size is larger than a certain threshold. We show, that this threshold is eight elements.

This paper is structured in the following way: Encodings from CSP to SAT are introduced briefly in Section 2. The transformation is presented in Section 3. In Section 4 we give experimental results and finally conclude the work in Section 5.

2 Preliminaries

A finite *Constraint Satisfaction Problem* (CSP) is represented by a tuple (V, D, C) , where V is the set of variables and each variable v is assigned a domain d from the set of domains D , where a domain is a set of integers. The variable v can represent only values from d . C is the set of constraints that describe the problem. To satisfy the CSP, an assignment for all variables of V has to be found that does not violate a constraint of C . The reader will find more information about CSPs in [3].

Satisfiability Testing (SAT) is a subset of CSP. All variables in V have a binary domain with the result that they can be mapped to propositional variables. In the following, propositional variables p and negated propositional variables $\neg p$ are called *literals*. A constraint in SAT is a disjunction of literals and is called

clause. Special clauses are the *unit clause*, which contains only a single literal, and the *binary clause* with two literals. For industrial applications, the CDCL algorithm [17] seems to be the most powerful algorithm, because solvers using this algorithm dominated the last competitions. In the sequel the focus will be on this kind of SAT solvers.

In [21], Walsh studies the relationship between CSP and SAT. He furthermore compares the power of algorithms that are used to solve these two problems theoretically. SAT has been proven to be NP complete [5]. There are time polynomial mappings from SAT to CSP and from finite domain CSP to SAT [21]. Still, there is no competitive comparison of SAT and CSP or a detailed case analysis to choose the appropriate representation of a problem. In this work we will focus on the encoding from CSP to SAT. There are several ways to encode a CSP into SAT [7,8]. Encoding a domain from a CSP variable into SAT is done most prominently in following three ways:

1. log encoding (also called compact encoding) [7]
2. direct encoding (also called sparse encoding) [21]
3. order encoding [19]

There are also systems that combine the latter two encodings [16,6] and call it hybrid encoding. The log encoding represents the elements of a CSP domain logarithmically. The direct encoding and the order encoding use a linear representation. Motivated by this property, the presented transformation focusses on the direct and order encoding.

To encode a CSP variable v with a domain d by using the direct encoding results in $n = |d|$ propositional variables p_1, \dots, p_n . A variable p_i is assigned *true*, if the related CSP variable v is assigned the value of element i of d . Therefore, the following two things have to be stated:

1. At least one propositional variable has to be satisfied
2. At most one propositional variable has to be satisfied

In the following, the specified constraint is called *one-out-of- N* constraint and has the property that exactly one variable of the specified set of variables is assigned *true*. Encoding the first statement is usually done by a single clause that connects all variables of the constraint positively. The second statement is encoded with binary clauses: $1 \leq i \leq n, i < j \leq n : [\neg p_i, \neg p_j]$, resulting in $\frac{n(n-1)}{2}$ clauses. Encoding a domain with the direct encoding requires quadratically many clauses.

The order encoding orders the element of the domain artificially to overcome the quadratic growth. By encoding this order it is already ensured that a CSP variable cannot represent two values of its domain at the same time. In order to encode a CSP variable v with an ordered domain d , $n = |d|$ propositional variables have to be introduced. A variable p_i is assigned to *true*, if v has a value greater or equal to the i -th element of d . Only linearly many clauses are necessary for encoding the order of the elements in a domain, namely: $1 \leq i < n : [p_i, \neg p_{i+1}]$. Furthermore it is stated that the variable represents at

least the smallest value of the domain by adding a clause $[p_1]$. Due to this clause, p_1 has to be assigned *true* and can be removed from the formula, resulting in only $n - 1$ variables. The introduced order reduces the space complexity of the encoding from quadratic to linear in the number of clauses. Given the meaning of the propositional variables the statement $v = i$ is encoded by $p_i \wedge \neg p_{i+1}$.

As an example the variable v with the domain $d = 1, 2, 3, 4$ is encoded with the two presented encodings. The direct encoding uses the variables a_1, \dots, a_4 , where a_i is set to true iff $v = i$. The following clauses encode this domain in SAT:

$$[a_1, a_2, a_3, a_4], [\neg a_1, \neg a_2], [\neg a_1, \neg a_3], [\neg a_1, \neg a_4], [\neg a_2, \neg a_4], [\neg a_3, \neg a_4], [\neg a_3, \neg a_4]$$

The first clause states the at-least constraint. The CSP variable v has to have a value. All the remaining clauses force that v can have at-most one value. The clauses encode the implications $i \neq j : a_i \rightarrow \neg a_j$. Assuming that $v = 2$, a_2 is set to *true* and the remaining variables will be set to *false*. The order encoding for this example uses the variables b_1, \dots, b_4 , where b_i is set to true iff $v \geq i$. The domain is encoded by:

$$[b_1], [b_1, \neg b_2], [b_2, \neg b_3], [b_3, \neg b_4]$$

Again, the first clause states that the variable v has a value. The remaining clauses represent the order of the domain elements from 1 to 4. If v has value j , then each variable that represents $i < j : v \geq i$ has also to be set to *true*. The encoded implications represent exactly this behavior. Furthermore, all the propositional variables that represent an equation $i > j : v \geq i$ have to be set to *false*. These implication are expressed in the same binary clauses. Let $v = 2$, then b_2 is assigned *true*, whereas b_3 is assigned *false*. Consequently, b_4 is also assigned *false*. To satisfy the very first clause of the order encoding, b_1 is assigned *true*.

3 The Transformation

Using the order encoding seems to require less clauses. However, if a SAT solver has to solve a problem that is encoded in the direct encoding it is hard to re-encode the whole problem in another encoding. Thus, we focus on extracting One-out-of-N constraints, which possibly represents integer variables and their domain, and on replacing them by a hybrid representation of the order encoding. The presented approach is hybrid, because the variables representing the One-out-of-N constraint are not removed from the formula. The following sections describe the transformation from the direct encoding to the order encoding in propositional logic, illustrate its correctness and show how one-out-of-N constraints can be extracted from a formula in CNF efficiently. Furthermore, an example is give, that shows that the number of clauses can increase even more, as long as the hybrid approach is not chosen, but the direct encoding is replaced by the order encoding an the representation of all propositional variables of this domain is replaced.

3.1 Mapping the Order Encoding to the Direct Encoding

A one-out-of- N constraint of a variable v with n elements in its domain d will be described with the propositional variables a_1, \dots, a_n . Without the loss of generality we neglect the meaning of the elements of d and refer to them by their indices: $i \leq n : a_i = \text{true} \rightarrow v = i$. The order encoding will be described with the variables b_1, \dots, b_n and also the indexes of the domain elements are used: $i \leq n : p_i = \text{true} \rightarrow v \geq i$. Consequently, a statement about the variable can be mapped by the formula $v = i \equiv v \geq i \wedge \neg v \geq i+1$. On the CNF level, this mapping between the direct encoding and the order encoding is $a_i \equiv b_i \wedge \neg b_{i+1}$. Transforming this equation into CNF results in the clauses:

1. $[\neg a_i, b_i]$
2. $[\neg a_i, \neg b_{i+1}]$
3. $[a_i, \neg b_i, b_{i+1}]$

For each element of the ordered domain d these three clauses need to be generated. There are special cases for the first and the last element. In the order encoding, b_1 is always set to *true* with the result that the above clauses are reduced to $[\neg a_1, \neg b_2]$ and $[a_1, b_2]$ and represent the equivalence $a_1 \equiv \neg b_2$. As already discussed in Section 2, this effect is expected. Similarly, the clauses for the n -th element of d are also reduced, because it is the last element of the domain and there cannot be greater elements. Therefore, b_{n+1} is always set to *false*. The remaining clauses for the mapping of a_n are $[\neg a_n, b_n]$ and $[a_n, \neg b_n]$. These two clauses represent the equivalence $a_n \equiv b_n$.

The mapping is illustrated with an example of a variable v with its domain $d = \{1, 2, 3, 4\}$. The propositional variables for the direct encoding are a_1, a_2, a_3, a_4 and the order encoding uses the variable b_1, b_2, b_3, b_4 . Assuming v is set to 2. Consequently, a_2 has to be satisfied. Furthermore, b_2 is set to *true* and b_3 is set to *false* by using only the information of the variable. With the clauses that encode the domains in the two encodings, a_1, a_3, a_4, b_4 are assigned *false* and b_1 is assigned *true* (compare Section 2). Now the mapping $a_i \equiv b_i \wedge \neg b_{i+1}$ can be tested. The mapping produces exactly the truth values for the variables in the direct encoding as the encoding would do itself.

1. $a_1 \equiv \neg b_2 \equiv \neg \text{true} \equiv \text{false}$
2. $a_2 \equiv b_2 \wedge \neg b_3 \equiv \text{true} \wedge \neg \text{false} \equiv \text{true}$
3. $a_3 \equiv b_3 \wedge \neg b_4 \equiv \text{false} \wedge \neg \text{false} \equiv \text{false}$
4. $a_4 \equiv b_4 \equiv \text{false}$

Since encoding the mapping between the order encoding and the direct encoding is linear in the size of the domain, the quadratic direct encoding should be replaced to reduce the number of clauses in the formula. Let $H \equiv F \wedge D$ be a formula with the propositional variables p_1, \dots, p_m . D encodes a domain d of an integer variable v using the direct encoding, with $n = |d|$. F encodes the remaining problem. The formula H can be modified by a set of clauses G that represents the order encoding for d with the variables p_{m+1}, \dots, p_{m+n} and the

set of clauses M for the mapping between the order encoding G and the direct encoding D . We claim that all models for the formula $F \wedge G \wedge M$ represent all models for H . More precisely: every model for $F \wedge G \wedge M$ is also a model for H and every model I_1 for H can be extended to an interpretation $I_2 = I_1 \cup J$ which is a model for $F \wedge G \wedge M$.

Proof sketch: $I \models F \wedge G \wedge M \Rightarrow I \models H \equiv F \wedge D$. The subformula F can be dropped from the sketch, because it is obvious that $I \models F$. For D we have to show that the mapping M is correct, which should be obvious since the definition of M is chosen so that this property is fulfilled. Then we know that I is a valid interpretation of the order encoding G and the mapping M , hence D has also to be satisfied. The other direction can be proved in the same way.

We illustrate this on the following example. Let our original formula be H .

$$H = [1, 2, 3], [\neg 1, \neg 2], [\neg 1, \neg 3], [\neg 2, \neg 3], [\neg 2, 4], [\neg 1, \neg 5]$$

Here the reader can find a one-out-of-N constrain encoded directly.

$$D = [1, 2, 3], [\neg 1, \neg 2], [\neg 1, \neg 3], [\neg 2, \neg 3]$$

Let F be the remaining formula so that $H \equiv D \wedge F$ holds.

$$F = [\neg 2, 4], [\neg 1, \neg 5]$$

According to our definition we can transform the direct encoded one-out-of-N constrain D into a order encoded one.

$$G = [6], [6, \neg 7], [7, \neg 8]$$

The mapping is given by the formula M .

$$\begin{aligned} M = & [\neg 1, 6], [\neg 2, 7], [\neg 3, 8], \\ & [\neg 1, \neg 7], [\neg 2, \neg 8], \\ & [1, \neg 6, 7], [2, \neg 7, 8], [3, \neg 8] \end{aligned}$$

Now we replace D by $G \wedge M$ and result in H' .

$$\begin{aligned} H' = & [6], [6, \neg 7], [7, \neg 8], \\ & [\neg 1, 6], [\neg 2, 7], [\neg 3, 8], \\ & [\neg 1, \neg 7], [\neg 2, \neg 8], \\ & [1, \neg 6, 7], [2, \neg 7, 8], [3, \neg 8], \\ & [\neg 2, 4], [\neg 1, \neg 5] \end{aligned}$$

Let $I \models H$ be an arbitrarily chosen model of H .

$$I = \{1, \neg 2, \neg 3, \neg 4, \neg 5\}$$

Now the mapping M transforms the direct encoding into the order encoding, hence we can extent I to I' , since there is only one model $I' \models M$ with $I \subset I'$ and $I \models H$.

$$I' = \{1, \neg 2, \neg 3, \neg 4, \neg 5, 6, \neg 7, \neg 8\}$$

The other way around, from I' the model for H can be obtained easily by removing the newly introduced variables again from I' .

3.2 Extracting One-out-of-N Constraints

Extracting one-out-of-N constraints is sufficient to find domains that have been encoded with the direct encoding, because they have the same properties. The extraction on a given CNF can be done efficiently. Each clause that contains at least three literals is considered as candidate. All binary clauses are used to verify whether a candidate is really a one-out-of-N constraint. Let $binary:literal \rightarrow clause$ be a mapping from a literal l to the set of binary clauses that contain l . The following steps have to be executed for the verification:

- For each candidate clause c :
 1. Mark $\neg l$ for all literals $l \in c$
 2. For each literal $l \in c$:
 - Check whether all marked literals l' can be found in $binary(\neg l)$
 3. If any check failed, c is withdrawn
 4. Otherwise c represents a one-out-of-N constraint
 5. Clear all marks

The check in Step 2 ensures, that all the binary clause that are necessary for the direct encoding of the domain are found in the formula. If all these binary clauses are found, the clause c together with all the used binary clauses encode a one-out-of-N constraint with respect to the literals in c .

A given CNF might also encode a domain without the binary clauses. This is possible, if all the constraints of the CSP are encoded as nogood and thus, the at-least-one clause is the only positive occurrence of the corresponding propositional variables. This case is not considered in the extraction, because in this case there is only a single clause that represents the domain and transforming these domains results only in bigger formulas.

3.3 Altering the Original Formula

Usually, the number of variables does not influence the power of a SAT solver as much as the number of clauses. Recent experiments showed, that the ratio between clauses and variables influences the performance of our solver *riss*. This effect might be caused by the fact that a higher ratio ships with a higher number of clauses. Therefore, we replace only domains of a size n , such that the number of clauses of the formula is reduced if the order encoding and the mapping are added and the direct encoding is removed. As a side effect, the newly introduced variables for the order encoding decrease the ratio between clauses and variables. As already presented, the direct encoding uses $1 + \frac{n(n-1)}{2}$ clauses, the order encoding needs n clauses, and the mapping requires $2 + 3(n-2)$ clauses.

Table 1 shows the number of clauses that are necessary to encode a domain d with n elements in the direct encoding or the order encoding. The last column shows the difference between two neighboring cells for each line. This difference states the newly introduced clauses if one increases the domain size from the

| Size (n) | 3 | 4 | 5 | 6 | 7 | 8 | 9 | diff |
|-----------------|---|----|----|----|----|----|----|-----------|
| Direct Encoding | 4 | 7 | 11 | 16 | 22 | 29 | 38 | $+(n-1)$ |
| Order Encoding | 2 | 3 | 4 | 5 | 6 | 7 | 8 | +1 |
| Mapping | 7 | 10 | 13 | 16 | 19 | 22 | 25 | +3 |
| Difference | 5 | 6 | 6 | 5 | 3 | 0 | -5 | $4-(n-1)$ |

Table 1: Conversion size for several domain sizes

previous size by one element. Therefore, the n in this column represents the elements of the new domain and the value of the column itself represents the difference to the number of clauses that are additionally introduced to represent the new size compared to the old size. thus, one realizes that the order encoding needs only a constant number of clauses to encode the next domain size. The direct encoding needs linear many clauses to do so. Since the mapping needs only three more clauses per iteration, the difference in the number of clauses when using the direct encoding or using the order encoding combined with a mapping increases also linearly. As soon as the size of the domain becomes larger than four, the difference between the two encodings starts to decrease. When a size of eight is reached, the two encodings need the same amount of clauses. Table 1 furthermore shows that the difference in this row grows also linearly and thus the absolute numbers of clauses between the two encodings differ quadratically.

An illustration of the algorithm is given for the formula with the following clauses:

$$[\neg 1, 4], [1, \neg 4], [\neg 2, 5], [2, \neg 5], [\neg 3, 6], [3, \neg 6], [1, 2, 3], [\neg 1, \neg 2], [\neg 2, \neg 3], [\neg 1, \neg 3]$$

The extraction for encoded one-out-of-N constraints finds the clause $[1, 2, 3]$ as sole candidate. According to the algorithm in Section 3.2, the literals $\neg 1$, $\neg 2$ and $\neg 3$ are marked (Step 1). In step two each literal of the candidate is further analyzed: The list of binary clauses for the first literal $\neg 1$ contains $\neg 1, 4, \neg 2$ and $\neg 3$. For $\neg 2$ the literals $\neg 2, 5, \neg 1$ and $\neg 3$ are found. Finally, analyzing $\neg 3$ results in the literals $\neg 3, 6, \neg 1$ and $\neg 2$. Since the condition in step 2.1 is met for all literals of the candidate, the candidate represents a direct encoded domain $d = \{1, 2, 3\}$ with $|d| = 3$. As a next step, new variables 7, 8 and 9 are introduced and the clauses for the order encoded domain are created. Furthermore, the clauses for the mapping have to be generated. The mapping is $1 \equiv \neg 8$, $2 \equiv 8 \wedge \neg 9$ and $3 \equiv 9$. Finally, the transformed formula is created by removing the clauses for encoding d and adding the order encoding and the mapping:

1. $[\neg 1, 4], [1, \neg 4], [\neg 2, 5], [2, \neg 5], [\neg 3, 6], [3, \neg 6]$
2. $[7], [8, \neg 9]$
3. $[\neg 1, \neg 8], [1, 8], [\neg 2, 8], [\neg 2, \neg 9], [2, \neg 8, 9], [\neg 3, 9], [3, \neg 9]$

The first line is the part of the original formula that has not been altered. In the second line the clauses of the order encoding are given. The mapping from the order encoding and the new variables to the direct encoding and the original

variables are given in the third line. It can be seen that the number of clauses increases from 10 to 15, which is exactly the difference that has been specified in Table 1 for a domain with a cardinality of 3.

If no new variables would be introduced, but the semantic of the existing variables would be changed, the mapping $a_i \equiv a'_i \wedge \neg a'_{i+1}$ can result in an exponential blowup in the clauses. Thus, we have not considered to replace the semantics.

4 Experiments

The performance of SAT solvers decreases with an increasing number of clauses, which has been already evaluated experimentally. The presented transformation does not introduce more clauses if it is applied to domains that have a size greater equal eight. Therefore, the transformation has only been applied to formulas that contain this kind of domains. The SAT solver *riss* [12] has been extended and is able to transform the formula before or after preprocessing it. In the experiments, the transformation has been applied before preprocessing.

4.1 Encodings in Application Instances

First, we check the SAT Competition 2009 Application benchmark for instances that contain domains that have been encoded with the direct encoding. After applying the transformation to the instances, *riss* solved 177 instances, where 25 instances contained encoded domains. The average speedup on these instances is 3.4%. The instances with the encoded domains come from two families, namely the q-query family and the vmopc family. Especially the vmopc family seems to benefit from the encoding. More detailed results per instance are presented in Table 2. The column *Converted* shows the solving time for instances where the transformation has been applied. The second column *Original* shows the runtime for the original instances of the transformation. The number of transformed domains is given in the column *Domains* and the speedup for an instance is given in the last column *Speedup*.

4.2 Encodings in Crafted Instances

Since the application benchmark contained only 25 instances with encoded domains, we looked at crafted instances because they encode different problems than application instances and do not often use the Tseitin transformation [20], but encode variable domains. We found 105 instances that contained encoded domains of at least eight elements in the Crafted Benchmark of the SAT Competition 2009 and in the medium and hard crafted instances of the SAT Competition 2007. We applied the transformation to these instances and ran a set of SAT solvers on these instances.

| Instance | Converted | Original | Domains | Speedup |
|-----------------------------|-----------|----------|---------|---------|
| q_query_2_L324_coli.cnf | 390.55 | 414.23 | 646 | -5.72% |
| q_query_3_L100_coli.sat.cnf | 473.84 | 253.21 | 297 | 87.13% |
| q_query_3_L150_coli.sat.cnf | 662.75 | 756.65 | 447 | -12.41% |
| q_query_3_L200_coli.sat.cnf | 684.18 | 846.23 | 597 | -19.2% |
| q_query_3_L60_coli.sat.cnf | 84.71 | 113.24 | 177 | -25.2% |
| q_query_3_L70_coli.sat.cnf | 250.77 | 194.22 | 207 | 29.12% |
| q_query_3_L80_coli.sat.cnf | 496.14 | 424.89 | 237 | 16.77% |
| q_query_3_L90_coli.sat.cnf | 505.88 | 625.65 | 267 | -19.14% |
| q_query_3_l37_lambda.cnf | 3.59 | 9.04 | 108 | -60.28% |
| q_query_3_l38_lambda.cnf | 10.05 | 8.41 | 111 | 19.44% |
| q_query_3_l39_lambda.cnf | 24.61 | 12.92 | 114 | 90.40% |
| q_query_3_l40_lambda.cnf | 31.83 | 34.37 | 117 | -7.42% |
| q_query_3_l41_lambda.cnf | 63.31 | 49.46 | 120 | 27.98% |
| q_query_3_l42_lambda.cnf | 110.53 | 97.71 | 123 | 13.11% |
| q_query_3_l43_lambda.cnf | 296.08 | 149.25 | 126 | 98.37% |
| q_query_3_l44_lambda.cnf | 1054.14 | 950.41 | 129 | 10.91% |
| q_query_3_l45_lambda.cnf | 885.27 | 922.83 | 132 | -04.07% |
| q_query_3_l46_lambda.cnf | 908.24 | 852.52 | 135 | 06.53% |
| q_query_3_l47_lambda.cnf | 1040.25 | 1177.46 | 138 | -11.65% |
| q_query_3_l48_lambda.cnf | 901.41 | 977.47 | 141 | -07.78% |
| vmcpc_24.cnf | 18.06 | 86.66 | 48 | -79.16% |
| vmcpc_25.cnf | 24.46 | 62.21 | 50 | -60.67% |
| vmcpc_26.cnf | 18.48 | 45.86 | 52 | -59.70% |
| vmcpc_28.cnf | 84.50 | 423.97 | 56 | -80.06% |
| vmcpc_29.cnf | 1455.13 | 2147.83 | 58 | -32.25% |

Table 2: Comparison on the application benchmark

Table 3¹ summarizes the results of this experiment. The table provides an overview over the number of the solved original instances (Original instances) and the number of solved instances after the transformation (Transformed instances). Furthermore, the average runtime per instance is given for both versions of the instances. Our solver *riss* can solve only 53 instead of 57 instances within the timeout, but it shows a significant reduced average runtime for the solved instances. In average *riss* solves each instances 200 seconds faster, resulting in a speedup of 28.27%.

Another solving technique for crafted instances is the look-ahead algorithm for choosing a variable before branching. The runtime of this algorithm seems to be highly correlated with the number of variables in the problem. A well known look-ahead solver march [9] can solve one more instance if the transformation is applied, but needs much more time per instance, namely 24.8%. In contrast,

¹ More detailed results per instance can be found at <http://www.ki.inf.tu-dresden.de/~norbert/paperdata/CSPTSAT2011.html>

| Solver | <i>riss</i> | march_hi | MiniSAT 2.2 | clasp | PrecoSAT |
|-----------------------|-------------|----------|-------------|--------|----------|
| Original instances | 57 | 29 | 47 | 61 | 57 |
| Transformed instances | 54 | 30 | 48 | 62 | 53 |
| Original runtime | 703.76 | 467.09 | 529.64 | 424.88 | 425.43 |
| Transformed runtime | 505.90 | 621.60 | 561.07 | 505.82 | 403.06 |

Table 3: Comparison on the crafted benchmark

another well known CDCL based solver MiniSAT 2.2 [15] can also solve one more instance but its runtime does not increase significantly.

Two more SAT solvers are analyzed. The solver clasp [13] is the winner of the crafted benchmark track of the SAT Competition 2009. It is also able to solve one more instance, but its average runtime increases by 16 % in comparison to MiniSAT. A reason for this effect could be that clasp does more calculation to determine the next branching variable than other CDCL solvers. PrecoSAT [4] can solve three instances less and its runtime improves by 5%.

Concerning the runtime improvement, there seems to be a cut between solvers that do fast decisions, such as as *riss*, MiniSAT and PrecoSAT, and solvers that spend more time for the next decision, such as march and clasp. Since the transformation encodes the original problem in another way, the heuristics of the solver also output different decisions that lead to different guiding path and introduce a high variation in the solving time for each instances. Still, the conversion from the direct encoding to the order encoding boosts the performance of CDCL based SAT solvers.

5 Conclusion

In this paper we present a study on transformations on the CNF level. The contribution of this work is a transformation that reduces the number of clauses by replacing a quadratic direct encoding of a CSP variable domain with a linear order encoded domain and the corresponding linear mapping between the two encodings. We illustrated the correctness of the transformation and studied the influence of this mapping to the performance of several SAT solvers on the Crafted and Application Benchmark of the SAT Competition 2009 and medium and hard instances from the Crafted Benchmark of the SAT Competition 2007. We also show that transforming domains with more than eight elements results in a formula with less clauses. On our CDCL based SAT solver *riss* the runtime to solve an encoded instance improves by almost 30% in average on crafted instances and improves by 3.4% on application instances. The performance of other CDCL based SAT solvers, e.g. PrecoSAT or MiniSAT 2.2 improves only slightly. Since variables are added to the formula, solvers that do not choose the next decision literal quickly seem to loose performance after the transformation. The performance of clasp, the solver that won the crafted track of the SAT Competition 2009, could solve one more instance, but needed 16% more runtime

per instance, on average. In average the look ahead solver march uses 24.8% more runtime to solve an instance.

We choose to not replace the direct encoding completely, but replace the clauses that encode a domain and introduce some new variables and new clauses for the order encoding of this domain and the corresponding mapping. This scheme has also been used in [16,6]. Replacing the variables of the direct encoding completely would result in a formula with less variables and thus the complexity of the given problem is reduced. First attempts for this conversion failed, because the time for the replacement has been higher than solving the original instance. Furthermore, we are interested in more studies on the comparison of the order encoding and the direct encoding and the way these two encodings constrain the search space during the solving process.

Since not only domains are encoded into CNF but the whole CSP, there might be more transformations that reduce the number of clauses in a CNF. As experiments showed, less clauses seem to boost the performance of SAT solvers as we showed for the order encoding. We are not aware of any other transformation on CNF that re-encodes a part of the problem.

This work also shows, that the order encoding of CSP variables seems to be beneficial. To our knowledge, the direct encoding is used more than the order encoding, although Sugar [18], one of the leading CSP to SAT converter uses the order encoding and won several CSP competition tracks. This work tries to bridge the gap between the mostly used direct encoding in CNF and the beneficial order encoding.

Open questions are related to the different representations of the one-out-of-N constraint. If the at-most-one constraint is not encoded and thus there are not quadratically many binary clauses, transforming the formula might still improve SAT solving because the order encoding might have beneficial properties over the direct encoding. Furthermore, there might also be a similar representation of other cardinality constraints that do not use the direct encoding but a related idea to the order encoding.

References

1. Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern sat solver. In *Twenty-first International Joint Conference on Artificial Intelligence(IJCAI'09)*, pages 399–404, jul 2009.
2. R. Béjar and F. Manyà. Solving the round robin problem using propositional logic. In *Procs. 17th National Conf. on Artificial Intelligence and 12th Conf. on Innovative Applications of Artificial Intelligence*, 2000.
3. Nicolas Beldiceanu, Mats Carlsson, Sophie Demassey, and Thierry Petit. Global constraint catalogue: Past, present and future. *Constraints*, 12:21–62, March 2007.
4. Armin Biere. PrecoSAT system description. <http://fmv.jku.at/precosat/preicosat-sc09.pdf>, 2009.
5. S. A. Cook. The complexity of theorem-proving procedures. In *Procs. 3rd Annual ACM Symposium on Theory of Computing*, 1971.

6. Thibaut Feydy and Peter J. Stuckey. Lazy clause generation reengineered. In *CP'09: Proceedings of the 15th international conference on Principles and practice of constraint programming*, pages 352–366, Berlin, Heidelberg, 2009. Springer-Verlag.
7. Marco Gavanelli. The log-support encoding of csp into sat. In *CP'07: Proceedings of the 13th international conference on Principles and practice of constraint programming*, pages 815–822, Berlin, Heidelberg, 2007. Springer-Verlag.
8. Ian P. Gent. Arc consistency in sat. In *Proceedings of ECAI 2002*, pages 121–125. IOS Press, 2002.
9. Marijn Heule and Hans van Maaren. march_hi. SAT 2009 Competitive Event Booklet, <http://www.cril.univ-artois.fr/SAT09/solvers/booklet.pdf>, 2009.
10. Steffen Hölldobler, Norbert Manthey, and Ari Saptawijaya. Improving resource-unaware sat solvers. In Christian Fermüller and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 6397 of *Lecture Notes in Computer Science*, pages 357–371. Springer Berlin / Heidelberg, 2010.
11. I. Lynce and J. Marques-Silva. SAT in bioinformatics: making the case with haplotype inference. In *Procs. 9th SAT, LNCS 4121*, 2006.
12. Norbert Manthey. Solver Submission of riss 1.0 to the SAT Competition 2011. Technical Report 1, Knowledge Representation and Reasoning Group, Technische Universität Dresden, 01062 Dresden, Germany, January 2011.
13. Benjamin Kaufmann Martin Gebser and Torsten Schaub. clasp: A Conict-Driven Answer Set Solver. SAT 2009 Competitive Event Booklet, <http://www.cril.univ-artois.fr/SAT09/solvers/booklet.pdf>, 2009.
14. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. *Design Automation Conference*, pages 530–535, 2001.
15. Niklas Sörensson. Minisat 2.2 and minisat++ 1.1. http://baldur.iti.uika.de/sat-race-2010/descriptions/solver_25+26.pdf, 2010.
16. Olga Ohrimenko and Peter J. Stuckey. Modelling for lazy clause generation. In *CATS '08: Proceedings of the fourteenth symposium on Computing: the Australasian theory*, pages 27–37, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.
17. João P. Marques Silva and Karem A. Sakallah. GRASP: A new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design, ICCAD '96*, pages 220–227, Washington, DC, USA, 1996. IEEE Computer Society.
18. Naoyuki Tamura and Mutsunori Banbara. Sugar: A csp to sat translator based on order encoding. www.cril.univ-artois.fr/CPAI06/descriptionSolvers/Sugar.pdf.
19. Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear csp into sat. *Constraints*, 14(2):254–272, 2009.
20. G. S. Tseitin. On the complexity of derivations in the propositional calculus. *Studies in Mathematics and Mathematical Logic*, Part II:115–125, 1968.
21. Toby Walsh. Sat v csp. In *in Proc. CP-2000*, pages 441–456. Springer-Verlag, 2000.
22. Jules White, Brian Dougherty, Douglas C. Schmidt, and David Benavides. Automated reasoning for multi-step feature model configuration problems. In *Proceedings of the 13th International Software Product Line Conference, SPLC '09*, pages 11–20, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.