Handbook of Satisfiability
Armin Biere, Marijn Heule, Hans van Maaren and Toby Walsh (Eds.)
IOS Press, 2021

doi:10.3233/FAIA200985

Chapter 2

CNF Encodings

Steven Prestwich

2.1. Introduction

Before a combinatorial problem can be solved by SAT methods, it must usually be transformed (*encoded*) to *conjunctive normal form* (CNF): a conjunction of clauses $\bigwedge_i c_i$, each clause c_i being a disjunction of literals $\bigvee_j l_j$, and each literal l_j being either a Boolean variable v or its negation $\overline{v}$. CNF has the advantage of being a very simple form, leading to easy algorithm implementation and a common file format. Unfortunately there are several ways of encoding most problems and few guidelines on how to choose among them, yet the choice of encoding can be as important as the choice of SAT solver.

This chapter reviews theoretical and empirical work on CNF encodings. In Section 2.2 we describe techniques for transforming from a formula in propositional logic to CNF (and 3-CNF), in particular the Tseitin encodings and some variants, the encoding of extensional and intensional constraints, and the DIMACS file format for CNF. In Section 2.3 we present case studies in CNF encoding, illustrating the techniques of this section and other modelling aspects. In Section 2.4 we discuss what makes one encoding better than another. Finally, Section 2.5 concludes the chapter.

Some of the topics mentioned in this chapter are covered in greater depth elsewhere in this book, but we cover them briefly to aid the discussion. For the sake of readability we shall omit some details from examples of SAT encodings. In particular, instead of $\bigwedge_{j=1}^{m} (\bigvee_{i=1}^{n} v_{i,j})$ we shall simply write $\bigvee_i v_{i,j}$ and leave it to the reader to quantify the variable i over its range, and to form conjunctions of clauses by quantifying the free variable j over its range. For full details please consult the various papers cited below.

2.2. Transformation to CNF

Propositional logic formulae can be transformed to CNF in several ways, but this may lose a great deal of structural information. Some of this information can be computed from the CNF encoding and can be used to boost the performance of both DPLL [Li00, OGMS02] (the Davis-Putnam-Logemann-Loveland

[DP60, DLL62] family of backtracking-based SAT algorithms) and local search [Seb94], but some of it is intractable to compute [LM98]. To avoid this difficulty, solvers for non-CNF problems have been devised [AG93, VG88, KMS97, MS06, Ott97, PTS07, Sta01, TBW04]. These techniques can yield great improvements on certain structured problems.

However, CNF currently remains the norm. The reason might be that tried and tested heuristics, based on problem features such as clause lengths and the number of occurrences of literals, would need to be adapted to non-CNF formulae. In this section we describe methods for transforming propositional logic formulae and constraints into CNF.

2.2.1. Transformation by Boolean algebra

A propositional logic formula can be transformed to a logically equivalent CNF formula by using the rules of Boolean algebra. For example consider the propositional formula

$$(a \to (c \wedge d)) \vee (b \to (c \wedge e))$$

(taken from [TBW04]). The implications can be decomposed:

$$((a \to c) \wedge (a \to d)) \vee ((b \to c) \wedge (b \to e))$$

The conjunctions and disjunctions can be rearranged:

$$\begin{array}{l}((a \to c) \vee (b \to c)) \wedge ((a \to c) \vee (b \to e)) \wedge \\ ((a \to d) \vee (b \to c)) \wedge ((a \to d) \vee (b \to e))\end{array}$$

The implications can be rewritten as disjunctions, and duplicated literals removed:

$$(\overline{a} \vee \overline{b} \vee c) \wedge (\overline{a} \vee \overline{b} \vee c \vee e) \wedge (\overline{a} \vee \overline{b} \vee c \vee d) \wedge (\overline{a} \vee \overline{b} \vee d \vee e)$$

Finally, subsumed clauses can be removed, leaving the conjunction

$$(\overline{a} \vee \overline{b} \vee c) \wedge (\overline{a} \vee \overline{b} \vee d \vee e)$$

This example reduces to a compact CNF formula, but in general this method yields exponentially large formulae.

2.2.2. Transformation by Tseitin encoding

Conversion to CNF is more usually achieved using the well-known Tseitin encodings [Tse83], which generate a linear number of clauses at the cost of introducing a linear number of new variables, and generate an equisatisfiable formula (satisfiable if and only if the original formula is satisfiable). Tseitin encodings work by adding new variables to the CNF formula, one for every subformula of the original formula, along with clauses to capture the relationships between these new variables and the subformulae. On the above example the Tseitin encoding would introduce a variable f_1 with definition

$$f_1 \leftrightarrow (c \wedge d)$$

to represent subformula $(c \wedge d)$. This definition can be reduced to clausal form:

$$(\overline{f_1} \vee c) \wedge (\overline{f_1} \vee d) \wedge (\overline{c} \vee \overline{d} \vee f_1)$$

Similarly it would introduce a definition

$$f_2 \leftrightarrow (c \wedge e)$$

which reduces to clauses

$$(\overline{f_2} \vee c) \wedge (\overline{f_2} \vee e) \wedge (\overline{c} \vee \overline{e} \vee f_2)$$

Applying these definitions the original formula is reduced to

$$(\overline{a} \vee f_1) \wedge (\overline{b} \vee f_2)$$

Next two more variables would be introduced with definitions

$$f_3 \leftrightarrow (\overline{a} \vee f_1)$$

and

$$f_4 \leftrightarrow (\overline{b} \vee f_2)$$

and clauses

$$(\overline{f_3} \vee \overline{a} \vee f_1) \wedge (a \vee f_3) \wedge (\overline{f_1} \vee f_3) \wedge (\overline{f_4} \vee \overline{b} \vee f_2) \wedge (b \vee f_4) \wedge (\overline{f_2} \vee f_4)$$

The formula is now reduced to

$$(f_3 \vee f_4)$$

Finally, a variable would be introduced with definition

$$f_5 \leftrightarrow (f_3 \vee f_4)$$

and clauses

$$(\overline{f_5} \vee f_3 \vee f_4) \wedge (\overline{f_3} \vee f_5) \wedge (\overline{f_4} \vee f_5)$$

Tseitin CNF encodings are linear in the size of the original formula as long as the Boolean operators that appear in the formula have linear clausal encodings. The operators *and*, *or*, *not*, *nand*, *nor* and *implies* all have linear clausal encodings. In practice we might choose not to define some variables, for example $(f_3 \vee f_4)$ is already in clausal form so we need not define f_5. We might also choose to expand some non-clausal subformulae.

Alternatives to the Tseitin encodings have been described, mainly in the context of Boolean circuits, but the ideas are relevant to SAT in general. Consider the following example (taken from [TBW04]). Suppose we have a formula $X \vee (p \wedge q)$ where X is a large unsatisfiable formula and p, q are Boolean variables. A Tseitin encoding will include auxiliary variables x, y, z defined by $x \leftrightarrow X$, $y \leftrightarrow (p \wedge q)$ and $z \leftrightarrow (x \vee y)$. Now suppose that DPLL first selects variables y, z and sets them to true. Then unit propagation will set p, q to true. Next suppose that it selects variable x and sets it to false. Then DPLL only needs to find any variable assignment that falsifies X (which should be easy) and the problem is solved. However, if x was instead set to true then DPLL might be forced to perform a lengthy search to refute X, before backtracking to set x to false. Under the assignments $[\overline{x}, y, z]$ the X variables have become *don't care variables*: their values do not affect the satisfiability of the problem and they have become *unobservable*. A survey of transformation techniques for avoiding this source of inefficiency is given in [Vel05].

2.2.3. Transformation from CNF to 3-CNF

3-CNF formulae have exactly three literals in each clause. These problems are interesting in their own right, and random 3-CNF problems have been extensively studied. However, any SAT problem can be transformed into 3-CNF. SAT is known to be NP-complete, so this transformation proves the completeness of 3-CNF, by showing that 3-CNF is as expressive as CNF.

A well-known transformation is as follows. Suppose we have a SAT instance with clauses $c_1, \ldots, c_m$ over variables $u_1, \ldots, u_n$, where each clause c_i may contain any positive number of literals. This can be transformed to 3-CNF as follows. For a clause $c_i = (z_1 \vee \ldots \vee z_k)$ there are four cases:

- $(k = 1)$ Create two new variables $y_{i,1}, y_{i,2}$ and replace c_i by the clauses $(z_1 \vee y_{i,1} \vee y_{i,2})$, $(z_1 \vee \overline{y_{i,1}} \vee y_{i,2})$, $(z_1 \vee y_{i,1} \vee \overline{y_{i,2}})$ and $(z_1 \vee \overline{y_{i,1}} \vee \overline{y_{i,2}})$.
- $(k = 2)$ Create one new variable $y_{i,1}$ and replace c_i by the clauses $(z_1 \vee z_2 \vee y_{i,1})$ and $(z_1 \vee z_2 \vee \overline{y_{i,1}})$.
- $(k = 3)$ Leave clause c_i unchanged.
- $(k > 3)$ Create new variables $y_{i,1}, y_{i,2}, \ldots, y_{i,k-3}$ and replace c_i by the clauses $(z_1 \vee z_2 \vee y_{i,1})$, $(\overline{y_{i,1}} \vee z_3 \vee y_{i,2})$, $(\overline{y_{i,2}} \vee z_4 \vee y_{i,3})$, $\ldots$, $(\overline{y_{i,k-3}} \vee z_{k-1} \vee z_k)$.

This transformation does not seem to have many practical applications, but it may be useful for generating 3-CNF benchmarks, or as a preprocessor for a specialised algorithm that only accepts 3-CNF problems.

2.2.4. Extensional constraints in CNF

One way of modelling a problem as SAT is first to model it as a constraint satisfaction problem (CSP), then to apply a standard encoding from CSP to SAT. A finite-domain CSP has variables $v_1 \ldots v_n$ each with a finite domain $\mathrm{dom}(v_i)$ of values, and constraints prescribing prohibited combinations of values (alternatively, constraints may prescribe *allowed* combinations of assignments). The problem is to find an assignment of values to all variables such that no constraint is violated. An example of a constraint is $(v_3 = 2, v_4 = 2)$ which prohibits the combination of assignments $v_3 = 2$ and $v_4 = 2$. This is a *binary* constraint because it contains only two variables, and a binary CSP contains only binary constraints.

The most natural and widely-used encoding from CSP to SAT is the *direct encoding* (also called the *sparse encoding*). A SAT variable $x_{v,i}$ is defined as true if and only if the CSP variable v has the domain value i assigned to it. The direct encoding consists of three sets of clauses. Each CSP variable must take at least one domain value, expressed by *at-least-one* clauses:

$$\bigvee_i x_{v,i}$$

No CSP variable can take more than one domain value, expressed by *at-most-one* clauses:

$$\overline{x_{v,i}} \vee \overline{x_{v,j}}$$

Conflicts are enumerated by *conflict* clauses:

$$\overline{x_{v,i}} \vee \overline{x_{w,j}}$$

An alternative to the direct encoding is the *support encoding* (defined only for binary CSPs) [Gen02, Kas90] in which conflict clauses are replaced by *support* clauses. Suppose that $S_{v,j,w}$ are the *supporting values* in $\mathrm{dom}(v)$ for value $j \in dom(w)$: these are the values in $\mathrm{dom}(w)$ that do not conflict with $w = j$. Then add a support clause

$$\overline{x_{w,j}} \vee \left(\bigvee_{i \in S_{v,j,w}} x_{v,i} \right)$$

A third alternative is the *log encoding*, first defined for Hamiltonian circuit, clique and colouring problems [IM94] and since used for other problems including planning [EMW97] and vertex colouring [VG07]. Its motivation is to exponentially reduce the number of SAT variables with respect to the direct encoding. Each CSP variable/domain value bit has a corresponding SAT variable, and each conflict has a corresponding clause prohibiting that combination of bits in the specified CSP variables. Define variables $x_{b,v}$ where $x_{b,v} = 1$ if and only if bit b of the domain value assigned to v is 1 (here we number the bits from 0 with the 0^{th} bit being least significant). For example to prohibit the combination $[p = 2, q = 1]$ where p, q have domains $\{0, 1, 2\}$ and therefore require two bits each, we must prohibit the combination $[\overline{x_{0,p}}, x_{1,p}, \overline{x_{0,q}}, x_{1,q}]$ by adding a conflict clause $(x_{0,p} \vee \overline{x_{1,p}} \vee x_{0,q} \vee \overline{x_{1,q}})$. No at-least-one or at-most-one clauses are required, but if the domain size of v is not a power of 2 then we must add clauses to prohibit combinations of bits representing non-domain values. For example here we must prevent the bit pattern $[x_{0,p}, x_{1,p}]$ representing $p = 3$ by adding a clause $\overline{x_{0,p}} \vee \overline{x_{1,p}}$, and similarly for q.

To illustrate the three encodings we use a simple graph colouring problem as an example, with two adjacent vertices v and w and three available colours $\{0, 1, 2\}$. Each vertex must take exactly one colour, but because they are adjacent they cannot take the same colour. This problem has six solutions: $[v = 0, w = 1]$, $[v = 0, w = 2]$, $[v = 1, w = 0]$, $[v = 1, w = 2]$, $[v = 2, w = 0]$ and $[v = 2, w = 1]$. The direct encoding of the problem contains at-least-one clauses

$$x_{v,0} \vee x_{v,1} \vee x_{v,2} \qquad x_{w,0} \vee x_{w,1} \vee x_{w,2}$$

at-most-one clauses

$$\begin{array}{lll} \overline{x_{v,0}} \vee \overline{x_{v,1}} & \overline{x_{v,0}} \vee \overline{x_{v,2}} & \overline{x_{v,1}} \vee \overline{x_{v,2}} \\ \overline{x_{w,0}} \vee \overline{x_{w,1}} & \overline{x_{w,0}} \vee \overline{x_{w,2}} & \overline{x_{w,1}} \vee \overline{x_{w,2}} \end{array}$$

and conflict clauses

$$\overline{x_{v,0}} \vee \overline{x_{w,0}} \qquad \overline{x_{v,1}} \vee \overline{x_{w,1}} \qquad \overline{x_{v,2}} \vee \overline{x_{w,2}}$$

The support encoding contains the same at-least-one and at-most-one clauses, plus support clauses

$$\begin{array}{lll} x_{v,1} \vee x_{v,2} \vee \overline{x_{w,0}} & x_{v,0} \vee x_{v,2} \vee \overline{x_{w,1}} & x_{v,0} \vee x_{v,1} \vee \overline{x_{w,2}} \\ x_{w,1} \vee x_{w,2} \vee \overline{x_{v,0}} & x_{w,0} \vee x_{w,2} \vee \overline{x_{v,1}} & x_{w,0} \vee x_{w,1} \vee \overline{x_{v,2}} \end{array}$$

The log encoding contains only conflict clauses

$$\begin{array}{l} x_{v,0} \vee x_{v,1} \vee x_{w,0} \vee x_{w,1} \\ \overline{x_{v,0}} \vee x_{v,1} \vee \overline{x_{w,0}} \vee x_{w,1} \\ x_{w,0} \vee \overline{x_{w,1}} \vee x_{w,0} \vee \overline{x_{w,1}} \end{array}$$

and *prohibited value clauses* to forbid non-domain values

$$\overline{x_{v,0}} \vee \overline{x_{v,1}} \qquad \overline{x_{w,0}} \vee \overline{x_{w,1}}$$

There are also further encodings. *Hierarchical encodings* recursively subdivide domains [Vel07], and *reepresentative-sparse encoding* [BHN14] are based on hierarchical and direct encodings. A hybrid of the log and support encodings called the *log support encoding* was devised by [Gav07]. Variables and prohibited value clauses as in the log encoding, and some of the conflict clauses are replaced by support clauses. A variant based on Gray codes was also proposed. The *binary transform* of [FP01] is a variant of the log encoding that avoids the use of prohibited value clauses. The direct encoding without at-most-one clauses is often used with SAT local search algorithms, and has been called the *multivalued encoding* [Pre04]. SAT solutions under this encoding represent *bundles* of solutions to the CSP, formed by taking the Cartesian product of the CSP domain values allowed in the SAT solution, and to extract a single solution we simply select any allowed value for each CSP variable. For binary CSPs the *maximal encoding* [Pre04] eliminates all but the maximal bundles (in which no additional domain values are allowed) by adding *maximality clauses* to the multivalued encoding:

$$x_{v,i} \vee \bigvee_{w,j} x_{w,j}$$

where w, j range over the assignments to other variables $w = j$ that conflict with $v = i$. The maximal encoding breaks a form of dominance between CSP solutions and can speed up all-solution search. On the above example the maximality clauses are:

$$x_{v,0} \vee x_{w,0} \qquad x_{v,1} \vee x_{w,1} \qquad x_{v,2} \vee x_{w,2}$$

For CSPs with integer linear constraints, a better alternative is the *order encoding* [AM04, CB94, TTKB09]. In this encoding a SAT variable $v_{x,a}$ represents a *primitive comparison* $x \leq a$ for integer variable x and upper bound a. Other comparisons (linear constraints) are transformed into primitive comparisons via a method described in [TTKB09], illustrated here by an example. For integer variables x, y, z each with domain $\{0, 1, 2, 3\}$ the comparison $x+y < z-1$ becomes

$$\begin{array}{lll} v_{x,-1} \vee v_{y,-1} \vee \overline{v_{z,1}} & v_{x,-1} \vee v_{y,0} \vee \overline{v_{z,2}} & v_{x,-1} \vee v_{y,1} \vee \overline{v_{z,3}} \\ v_{x,0} \vee v_{y,-1} \vee \overline{v_{z,2}} & v_{x,0} \vee v_{y,0} \vee \overline{v_{z,3}} & v_{x,1} \vee v_{y,-1} \vee \overline{v_{z,3}} \end{array}$$

(Variables corresponding to non-existent domain values such as -1 can be eliminated.) The order relation is represented by *axiom clauses* of the form

$$\overline{v_{x,a}} \vee v_{x,a+1}$$

The order encoding was proved to transform a tractable CSP into a tractable SAT problem by [PJ11]. Improved variants were developed by [AS14] to handle problems with large domains, incorporating techniques from the log and support encodings, and Multi and Binary Decision Diagrams. The *compact order encoding* [TTB11] combines ideas from the order and log encodings, and the *representative-order encoding* [Ngu17] uses the order encoding in partitions of the variables, which are representated by selected variables.

2.2.5. Intensional constraints in CNF

Not all constraints are conveniently expressed extensionally by listing conflicts, and we may wish to SAT-encode constraints that are usually expressed intentionally. For example the *at-most-one* constraint states that at most one of a set of variables $x_1 \dots x_n$ can be true. We shall use this example to illustrate alternative encodings of an intensional constraint. The often-used *pairwise encoding* treats the constraint extensionally, and simply enumerates $O(n^2)$ at-most-one clauses:

$$\overline{x_i} \vee \overline{x_j}$$

where $i < j$. The *ladder encoding* described by [GN04] is more compact: define new variables $y_1 \dots y_{n-1}$ and add *ladder validity clauses*

$$\overline{y_{i+1}} \vee y_i$$

and *channelling clauses* expanded from

$$x_i \leftrightarrow (y_{i-1} \wedge \overline{y_i})$$

The ladder adds $O(n)$ new variables but only $O(n)$ clauses. The *binary encoding* was proposed by [FPDN05] and later by [Pre07a] who named it the *bitwise encoding* (we use the former name here). Define new Boolean variables b_k where $k = 1 \dots \lceil \log_2 n \rceil$. Add clauses

$$\overline{x_i} \vee b_k \quad \left[\text{or } \overline{b_k}\right]$$

if bit k of $i-1$ is 1 [or 0]. This encoding has $O(\log n)$ new variables and $O(n \log n)$ binary clauses: more literals but fewer new variables than the ladder encoding. The pairwise encoding is acceptable for fairly small n, while for large n the ladder encoding gives good results with DPLL, and the binary encoding gives good results with local search algorithms. A *relaxed ladder encoding* defined in [Pre07a] also gave improved performance with local search. Further at-most-one encodings are surveyed in [NM15] and we briefly mention them here. In the *commander encoding* [KK07] the SAT variables are partitioned into subsets, each represented by a *commander variable* to reduce the number of clauses. The *product encoding* [Che10] is recursive and based on Cartesian products, and requires few clauses and variables. The *bimander encoding* [NM15] is based on the commander encoding but replaces the commander variables by variables from the binary encoding.

A useful generalisation of the at-most-one constraint is the *cardinality* constraint, which states that at most (or at least) k of a set of variables $x_1 \dots x_n$ can be true. This can be compactly SAT-encoded and a survey of such encodings was given by [MSL07], who also show that SAT solver performance on such encodings can be improved by effectively ignoring the auxiliary variables that they introduce. [BB03a] used a similar arithmetic-based method to that of [War98] for linear inequalities, but with a unary representation of integer variables to improve propagation. [Sin05] described two improved encodings: one smaller and based on a sequential counter, the other with better propagation and based on a parallel counter. [ANORC09, ANORC11b] based their encoding on sorting networks.

[CZI10] based a more compact encoding on a different type of sorting network. [OLH+13] used a different type of arithmetic network to obtain another compact encoding, which performed well on MaxSAT problems.

Integer linear inequalities (or *pseudo-Boolean constraints*) further generalise cardinality constraints and are of the form:

$$\sum_i w_i x_i \leq k$$

for some integer weights w_i (interpreting false as 0 and true as 1); the $\leq$ relation may be replaced by $<$, $\geq$, $>$ or $=$. These constraints can be handled either directly by generalising SAT to linear pseudo-Boolean problems, or indirectly by finding SAT encodings. [War98] SAT-encoded hardware arithmetic circuits to model these constraints. [ES06] described three encoding methods based on Binary Decision Diagrams, arithmetic networks and sorting networks. [ANORC11a] used Binary Decision Diagrams to obtain an encoding with improved propagation.

Another intensional constraint is the *parity* constraint $(\bigoplus_{i=1}^{n} v_i) \leftrightarrow p$ which states that p is true if and only if there is an even number of true variables in the set $\{v_1, \ldots, v_n\}$. This can be encoded simply by enumerating all possible combinations of v_i truth values, together with their parity p, but this creates exponentially many clauses and is only reasonable for small constraints. A more practical linear method due to [Li00] decomposes the constraint by introducing new variables f_j:

$$(v_1 \oplus f_1) \leftrightarrow p \quad (v_2 \oplus f_2) \leftrightarrow f_1 \quad \ldots \quad (v_{n-3} \oplus f_{n-3}) \leftrightarrow f_{n-2} \quad v_n \leftrightarrow f_{n-1}$$

These binary and ternary constraints are then put into clausal form by enumerating cases, for example $(v_1 \oplus f_1) \leftrightarrow p$ becomes:

$$(\overline{v_1} \vee \overline{f_1} \vee p) \wedge (v_1 \vee \overline{f_1} \vee \overline{p}) \wedge (\overline{v_1} \vee f_1 \vee \overline{p}) \wedge (v_1 \vee f_1 \vee p)$$

A commander encoding-style hybrid of the enumeration and linear methods was used in [Pre07b] to improve local search performance.

2.2.6. The DIMACS file format for CNF

A file format for SAT problems in CNF (as well as vertex colouring and clique problems) was devised in the DIMACS Challenge of 1993 [JT96], and has been followed ever since. (A format for general SAT was also proposed but does not appear to have been generally adopted.) Having a common file format has facilitated the collection of SAT benchmark problems in the SATLIB web site [1], the Beijing challenges [2] and the regular SAT solver competitions [3], which have stimulated a great deal of research into efficient algorithms and implementations. This is similar to the use of AMPL files for MIP problems, and file formats have also been introduced into Constraint Programming for solver competitions.

The format is as follows. At the start is a *preamble* containing information about the file. These optional comment lines begin with the letter "c":

[1] http://www.cs.ubc.ca/~hoos/SATLIB/
[2] http://www.cirl.uoregon.edu/crawford/beijing
[3] http://www.satcompetition.org/

```
c This is an example of a comment line.
```

Then a line of the form

`p cnf` *variables clauses*

states the number of Boolean variables and clauses in the file, where *variables* and *clauses* are positive integers and variables are numbered 1... *variables*. The rest of the file contains the clauses. Each clause is represented by a list of non-zero integers, followed by a zero which represents the end of the clause. The integers must be separated by spaces, tabs or newlines. A positive integer i represents a positive literal with variable number i, while a negative integer $-i$ represents a negative literal with variable number i. For example the line

```
1 5 -8 0
```

represents the clause $v_1 \vee v_5 \vee \overline{v_8}$. Clauses may run over more than one line, spaces and tabs may be inserted freely between integers, and both the order of literals in a clause and the order of clauses in the file are irrelevant.

2.3. Case studies

In this section we discuss concrete SAT models. We use simple problems to illustrate CSP-to-SAT encodings, alternative and non-obvious variable definitions, alternative combinations of clauses, the selective application of Tseitin encodings, the exploitation of subformula polarity, the use of implied and symmetry-breaking clauses, special modelling techniques for local search, and some common sources of error.

2.3.1. N-queens

We begin with an example of CSP-to-SAT modelling: the well-known n-queens problem. Consider a generalised chess board, which is a square divided into $n \times n$ smaller squares. The problem is to place n queens on it in such a way that no queen attacks any other. A queen *attacks* another if it is on the same row, column or diagonal (in which case both attack each other). How many ways are there to model this problem as a SAT problem? One or two models might spring to mind, but in a classic paper [Nad90] Nadel presents no fewer than nine constraint models (denoted by Q1–Q9), some representing families of models and one being isomorphic to a SAT problem. There are several ways of SAT-encoding a CSP, so we immediately have a rather large number of ways of modelling n-queens as SAT. We now discuss some of these ways, plus additional variants.

Nadel's model Q1 is perhaps the most obvious one. For each row define a variable with a finite domain of n values, denoting the column in which the queen on that row is placed. Add binary constraints to prevent attacks, by forbidding pairs of variables from taking values that place queens in the same column (the values are identical) or cause queens to attack along a diagonal (their absolute difference matches the absolute difference between the variable numbers). Row attacks are implicitly forbidden because each row variable takes exactly one value.

Model Q2 is very similar to Q1, but variables correspond to columns and values to rows. A direct encoding of Q1 or Q2 gives almost identical results, except for clauses ensuring that each row (Q1) or column (Q2) contains at least one queen. A further option is to use both sets of clauses, because in any n-queens solution there is a queen in each row and column. This is an example of adding implied clauses to a model: an implied clause is one that is implied by other clauses in the model, and therefore logically redundant. Implied clauses have been shown to speed up both backtracking [ABC$^+$02] and local search [CI96, KD95].

Model Q6 is already a SAT model. For each board square (i, j) define a Boolean variable $v_{i,j}$, which is true if and only if a queen is placed on it. To prevent partially-filled boards (including the empty board) from being solutions add clauses

$$\bigvee_i v_{i,j}$$

to ensure that each row contains a queen; alternatively we could ensure that each column contains a queen; or both. To prevent attacks add a clause

$$(\overline{v_{i,j}} \vee \overline{v_{i',j'}})$$

for each distinct pair of squares $(i, j), (i', j')$ that are on the same row, column or diagonal and therefore attack each other. It is worth pointing out here that there is no point in adding multiple versions of the same clause, for example:

$$(\overline{v_{2,5}} \vee \overline{v_{3,6}}) \qquad (\overline{v_{3,6}} \vee \overline{v_{2,5}})$$

This would occur if we only considered distinct attacking squares: $(i, j) \neq (i', j')$. Instead the attacking squares used in conflict clauses should satisfy $(i, j) \prec (i', j')$ for some total ordering $\prec$ on squares; for example the lexicographical ordering $(i, j) \prec (i', j')$ if and only if (i) $i < i'$ or (ii) $i = i'$ and $j < j'$. This type of needless duplication is easy to commit when SAT-encoding. Model Q6 is almost identical to the direct encoding of Q1 or Q2, modulo details of whether we ensure no more than one queen per row, column or both.

Model Q5 is quite different. It associates a variable with each queen, and each variable has a domain ranging over the set of squares on the board. A direct encoding would define a Boolean variable $v_{q,i,j}$ for each queen q and each square (i, j), which is true if and only if that queen is placed on that square. Add clauses to prevent two queens from being placed on the same square:

$$(\overline{v_{q,i,j}} \vee \overline{v_{q',i,j}})$$

for all pairs q, q' such that $q < q'$ (to avoid generating two equivalent versions of each clause). Add clauses to prevent attacks:

$$(\overline{v_{q,i,j}} \vee \overline{v_{q',i',j'}})$$

for each pair of queens $q < q'$ and each distinct pair of attacking squares $(i, j) \neq (i', j')$. Alternatively we could enumerate each distinct pair of queens $q \neq q'$ and each pair of attacking squares such that $(i, j) \prec (i', j')$. However, it would be incorrect to enforce both $q < q'$ and $(i, j) \prec (i', j')$ because this would (for

example) prevent the attacking configuration of queen 1 at (1,1) and queen 2 at (2,2), but it would not prevent the attacking configuration of queen 2 at (1,1) and queen 1 at (2,2). Great care must be taken with these issues when SAT-encoding.

A possible drawback with Q5 is that it contains a great deal of symmetry: every solution has $n!$ SAT models because the queens can be permuted. To eliminate this symmetry we can add further clauses, for example by forcing queens to be placed in squares with increasing indices:

$$(\overline{v_{q,i,j}} \vee \overline{v_{q',i',j'}})$$

where $q > q'$ and $(i,j) \prec (i',j')$. N-queens itself also has 8 symmetries: any solution can be reflected 2 times and rotated 4 times. To break these symmetries and thus reduce the size of the search space we may add further clauses, but we shall not consider these details.

Nor shall we consider Nadel's other constraint models, two of which define a variable for each upward or downward diagonal instead of each row or column: these again lead to SAT-encodings similar to Q6. But there are other ways of encoding the attack constraints. For example, suppose we define the Boolean variables $v_{i,j}$ of Q6. We must ensure that at most one queen is placed on any given row, column or diagonal, which in Q6 was achieved by enumerating all possible pairs of attacking queens. This is the pairwise encoding of the at-most-one constraint, but instead we could use a more compact encoding such as the ladder encoding. Another possibility is to model support instead of conflict in model Q1, replacing several conflict clauses

$$\overline{v_{i,j}} \vee \overline{v_{i',j'}}$$

by a clause

$$\overline{v_{i,j}} \vee \left(\bigvee_{j' \in C_{i,j,i'}} v_{i,j'} \right)$$

where $C_{i,j,i'}$ is the set of columns whose squares on row i' are not attacked by a queen at (i,j). This clause states that if a queen is placed on square (i,j) then the queen on row j' must be placed on a non-attacking column: note that its correctness depends on clauses ensuring that exactly one queen can be placed in any row. This model corresponds to a support encoding of Q1. We could also exchange the roles of rows and columns to obtain a support encoding of Q2, or use conflict or support clauses from Q1 and Q2 to obtain hybrid models.

The n-queens problem illustrates the approach of first modelling a problem as a CSP, then SAT-encoding the CSP. It shows that, even for a very simple problem, there may be a choice of problem features to encode as variables. Even after making this fundamental choice there may be alternative sets of clauses giving a logically correct model. It provides examples of implied clauses and symmetry breaking clauses. It also illustrates that care must be taken to avoid accidentally excluding solutions by adding too many clauses.

2.3.2. All-interval series

For our next example we take the all-interval series (AIS) problem, an arithmetic problem first used by Hoos [Hoo98] to evaluate local search algorithms, and in-

spired by a problem occurring in serial musical composition. There is a constructive solution to the AIS problem that requires no search so it is inherently easy. However, the easily-found solution is highly regular and other solutions might be more interesting to musicians. The problem is to find a permutation of the numbers $\{0, \ldots, n-1\}$ such that the set of differences between consecutive numbers contains no repeats. For example, a solution for $n = 11$ (taken from [GMS03]) with differences written underneath the numbers is:

0		10		1		9		2		8		3		7		4		6		5
	10		9		8		7		6		5		4		3		2		1	

First define Boolean variables $s_{i,j}$ each of which is true if and only if the i^{th} number is the integer j. AIS is a sequence of integers so each number takes exactly one value:

$$\left(\bigvee_i s_{i,j}\right) \wedge (\overline{s_{i,j}} \vee \overline{s_{i,j'}})$$

We may model differences as follows: if the formula

$$\bigvee_j (s_{i,j} \wedge s_{i+1,j\pm k})$$

is true then the difference between numbers i and $i+1$ is k, so we can enforce distinct differences by the formula:

$$\overline{\bigvee_j (s_{i,j} \wedge s_{i+1,j\pm k})} \vee \overline{\bigvee_j (s_{i',j} \wedge s_{i'+1,j\pm k})}$$

for distinct i, i' and relevant k values. (The meaning of $j \pm k$ here is that clauses should be generated using both $j+k$ and $j-k$, wherever the resulting numbers are within the range of the second index of the s variable.) A CNF encoding of this formula may be obtained by expansion, giving a set of quaternary clauses of the form

$$\overline{s_{i_1,j_1}} \vee \overline{s_{i_2,j_2}} \vee \overline{s_{i_3,j_3}} \vee \overline{s_{i_4,j_4}}$$

Alternatively, we may introduce Tseitin variable definitions

$$d_{i,k} \leftrightarrow \left[\bigvee_j (s_{i,j} \wedge s_{i+1,j\pm k})\right]$$

which means that $d_{i,k}$ is true if and only if the difference between numbers i and $i+1$ is k. We then replace the difference formulae by clauses

$$\overline{d_{i,k}} \vee \overline{d_{i',k}}$$

and CNF-encode the $d_{i,k}$ definitions. However, we can be more compact than the Tseitin method here. It was noted in [PG86] that only clauses corresponding to positive [negative] implications are logically necessary when transforming a subformula with negative [positive] *polarity*. The polarity of a subformula is

positive [negative] if it only occurs under an even [odd] number of negations. In this case each subformula

$$\bigvee_j (s_{i,j} \wedge s_{i+1,j\pm k})$$

only occurs negatively, so we only need clauses derived from the implication

$$d_{i,k} \leftarrow \left[\bigvee_j (s_{i,j} \wedge s_{i+1,j\pm k}) \right]$$

yielding clauses of the form

$$\overline{s_{i,j}} \vee \overline{s_{i+1,j\pm k}} \vee d_{i,k}$$

This is convenient, as the $\rightarrow$ implication would require either a larger expansion or more auxiliary variables.

To this encoding may be added several families of implied clauses. The $s_{i,j}$ variables encode a permutation of the integers $0 \ldots n$ so each integer occurs at most once:

$$\overline{s_{i,j}} \vee \overline{s_{i',j}}$$

The d variables encode a sequence of differences:

$$\bigvee_i d_{i,k}$$

and

$$\overline{d_{i,j}} \vee \overline{d_{i,j'}}$$

These differences form a permutation of the numbers $1 \ldots n$:

$$\overline{d_{i,j}} \vee \overline{d_{i',j}}$$

We have now arrived at Hoos's original AIS encoding [Hoo98]. A further study of AIS encodings was presented by [ABC+02]. They experimented with omitting the at-most-one clauses

$$\overline{s_{i,j}} \vee \overline{s_{i,j'}}$$

and

$$\overline{d_{i,k}} \vee \overline{d_{i,k'}}$$

which is safe because of the clauses ensuring that the two sets of variables encode permutations. Thus they treated the at-most-one clauses as implied clauses, and also introduced a less-obvious set of implied clauses:

$$\overline{d_{i,k}} \vee s_{i-1,m} \wedge \overline{d_{i,k}} \vee s_{i,m}$$

where $k > n/2$ (n is assumed to be even) and $n - k \leq m \leq k - 1$. In experiments with local search algorithms, different combinations of these implied clauses had either very good or very bad effects on performance, showing the unpredictable computational effects of implied clauses.

As pointed out by Gent et al. [GMS03] the AIS problem contains symmetry, which can be broken by adding further clauses. An AIS sequence may be reversed to give another sequence, so we may add clauses

$$\overline{s_{0,j}} \vee \overline{s_{1,j'}}$$

where $j' < j$ forcing the second number to be greater than the first. We may also subtract each value from $n-1$ to obtain another sequence so we can add clauses to ensure that the first value is no greater than $n/2$:

$$\overline{s_{0,j}}$$

where $j \leq n/2$. Gent et al. also use a clever technique to boost the performance of a constraint solver by a factor of 50. They point out a *conditional symmetry* of AIS, which is a symmetry that occurs during search when a subset of the variables have been assigned values. Breaking the conditional symmetry is not easy in the original AIS model, but they generalise AIS to a new problem containing additional symmetry (rotation of sequence values) that can easily be broken, which incidentally also breaks the symmetry and conditional symmetry in AIS.

The AIS example illustrates the unpredictable but potentially beneficial effects of adding implied clauses, the selective use of Tseitin variables, the exploitation of subformula polarity to reduce the size of the encoding, the use of non-obvious implied clauses, and the possibility of better models that might be found by ingenuity.

2.3.3. Stable marriages

In the above examples there was nothing very surprising about the variable definitions themselves: they mostly corresponded to assignments of constraint variables to domain values. In our next example, taken from a study by Gent et al. [GIM+01], the variable definitions are less obvious. In the stable marriage problem we have n men and n women. Each man ranks the women in order of preference, and women similarly rank men. The problem is to marry men and women in a *stable* way, meaning that there is no incentive for any two individuals to elope and marry each other. In the version of the problem considered, incomplete preference lists are allowed, so that a man [woman] might be unacceptable to some women [men]. A person is willing to leave his/her current partner for a new partner only if he/she is either unmatched or considers the new partner better than the current partner. A pair who mutually prefer each other are a blocking pair, and a matching without blocking pairs is stable. The problem of finding a stable matching is not NP-complete but an extended version allowing preference ties is, and can be modelled in a similar way; here we discuss only the simpler problem.

A direct encoding of Gent et al.'s first constraint model would contain a Boolean variable for each man-woman pair, which is true when they are matched. However, they also present a second model that is more compact and already in SAT form. They define a variable $x_{i,p}$ to be true if and only if man i is either unmatched or matched to the woman in position p or later in his preference

list, where $1 \leq p \leq L_{m,i}$ and $L_{m,i}$ is the length of his list. They also define $x_{i,L_{m,i}+1}$ which is true if and only if man i is unmatched. Similarly they define variables $y_{j,q}$ for each woman j. Note that each SAT variable corresponds to a set of possibilities, unlike those in our previous examples; also that marriages are modelled indirectly via the preference lists.

The clauses are as follows. Each man or woman is either matched with someone in their preference list or is unmatched, which is enforced by the unit clauses

$$x_{i,1} \wedge y_{j,1}$$

If a man gets his p^{th} choice then he certainly gets his $p + 1^{th}$ choice:

$$\overline{x_{i,p}} \vee x_{i,p+1}$$

Similarly for women:

$$\overline{y_{j,q}} \vee y_{j,q+1}$$

The x and y variables must be linked so that if man i is matched to woman j then woman j is also matched to man i. We can express the fact that man i is matched to the woman in position p in his preference list indirectly by the assignments $[x_{i,p}, \overline{x_{i,p+1}}]$ so the clauses are:

$$(\overline{x_{i,p}} \vee x_{i,p+1} \vee y_{j,q}) \wedge (\overline{x_{i,p}} \vee x_{i,p+1} \vee \overline{y_{j,q+1}})$$

where p is the rank of woman j in the preference list of man i, and q is the rank of man i in the preference list of woman j. Similarly:

$$(\overline{y_{j,q}} \vee y_{j,q+1} \vee x_{i,p}) \wedge (\overline{y_{j,q}} \vee y_{j,q+1} \vee \overline{x_{i,p+1}})$$

Stability is enforced by clauses stating that if man i is matched to a woman he ranks below woman j then woman j must be matched to a man she ranks no lower than man i:

$$\overline{x_{i,p}} \vee \overline{y_{j,q+1}}$$

where the man finds the woman acceptable (mentioned in the preference list). Similarly:

$$\overline{y_{j,q}} \vee \overline{x_{i,p+1}}$$

This problem provides a nice illustration of the use of non-obvious variable definitions.

Here it is worth pointing out another common source of error in SAT encoding: omitting common-sense axioms. The meaning of variable $x_{i,p}$ being true is that man i is matched to the woman in position p or later in his list. This must of course be explicitly stated in the SAT encoding via clauses $\overline{x_{i,p}} \vee x_{i,p+1}$. In the same way, if we are trying to construct a tree, a DAG or some other data structure then we must include axioms for those structures. However, in some cases these axioms are implied by other clauses. For example, if we are trying to construct a permutation of n numbers with some property then we would normally use at-least-one and at-most-one clauses, because the permutation is a list of length n with each position containing exactly one number. But if we add clauses stating that no two positions contain the same number then the at-most-one clauses become redundant.

2.3.4. Modelling for local search

So far we have largely ignored an important aspect of SAT modelling: the search algorithm that will be applied to the model. It is not immediately clear that the choice of algorithm should affect the choice of model, but it has been found that local search sometimes requires different models than DPPL if it is to be used effectively. This may partly explain the relatively poor performance of local search algorithms on industrial problems in the regular SAT solver competitions: they were perhaps modelled with DPLL in mind. [Pre02] showed that two particularly hard problems for local search can be solved by remodelling. Here we briefly summarise the SAT modeling techniques that were used.

A problem that makes heavy use of parity constraints is *minimal disagreement parity learning* described in [CKS94]. DPLL (and extended versions) on standard encodings is much more successful than local search on this problem. The explanation seems to be that standard encodings use a similar structure to the linear encoding of the parity constraint described in Section 2.2.5, which is very compact but creates a long chain of variable dependencies. These are known to slow down local search [KMS97], and when they form long chains they may cause local search to take polynomial or even exponential time to propagate truth assignments [Pre02, WS02]. A key to solving large parity learning instances by local search turns out to be a different encoding of the parity constraint [Pre02]. Decompose a large parity constraint $\bigoplus_{i=1}^{n} v_i = p$ into $\beta + 1$ smaller ones:

$$\bigoplus_{i=1}^{\alpha} v_i \equiv p_1 \qquad \bigoplus_{i=\alpha+1}^{2\alpha} v_i \equiv p_2 \qquad \ldots \qquad \bigoplus_{i=n-\alpha+1}^{n} v_i \equiv p_\beta \qquad \text{and} \qquad \bigoplus_{i=1}^{\beta} p_i \equiv p$$

where $p_1, \ldots, p_\beta$ are new variables and $n = \alpha \times \beta$ (in experiments $\beta \approx 10$ gave the best results). Now expand these small parity constraints into clauses. This exponentially increases the number of clauses in the model yet greatly boosts local search performance, allowing a standard local search algorithm to solve the notorious 32-bit instances of [CKS94]. This was previously achieved only by using an enhanced local search algorithm that exploited an analysis of the problem [PTS07]. Note that a similar result could be obtained by eliminating some of the variables in the linear encoding, thus a promising modelling technique for local search is to collapse chains of dependency by selective variable elimination.

Another hard problem for local search is the well-known Towers of Hanoi expressed as a STRIPS planning problem (see elsewhere in this book for background on modelling planning problems as SAT). Towers of Hanoi consists of P pegs (or towers) and D disks of different sizes; usually $P = 3$. All D disks are initially on the first peg. A solution is a sequence of moves that transfers all disks to the third peg with the help of the second peg so that (i) only one disk can be moved at a time; (ii) only the disk on top can be moved; (iii) no disk can be put onto a smaller disk. There always exists a plan with $2^D - 1$ steps. No SAT local search algorithm previously solved the problems with 5 and 6 disks in a reasonable time and 4 disks required a specially-designed algorithm, while DPLL algorithms can solve the 6-disk problem. However, 6 disks can be solved quite quickly by local search after remodelling the problem [Pre02]. Firstly, a more compact STRIPS

model was designed, which allowed a standard local search algorithm to solve 4 disks (it also improved DPLL performance and is not relevant here). Secondly, parallel plans were allowed, but not enforced, so that instead of a single serial plan there are a choice of many plans with varying degrees of parallelism. This increases the solution density of the SAT problem and allowed local search to solve 5 disks. Thirdly, implied clauses were added to logically connect variables that were otherwise linked only by chains of dependency. This is another way of reducing the effects of such chains on local search, and was first described in [WS02] for an artificial SAT problem. In planning-as-SAT the state at time t is directly related to the state at time $t+1$ but not to states at more distant times, via (for example) frame axioms which in *explanatory* form are:

$$\tau_{p,t} \wedge \overline{\tau_{p,t+1}} \rightarrow \left(\bigvee_a \alpha_{a,t} \right)$$

where a ranges over the set of actions with delete effect p. This axiom states that if property p is true at time t but false at time $t+1$ then at least one action must have occurred at time t to delete property p. (Similar axioms are added for properties that are true at time t but false at time $t+1$.) These axioms create chains of dependency between the τ variables, which can be "short-circuited" by a generalisation of the frame axioms that connect states at arbitrary times:

$$\tau_{pt} \wedge \overline{\tau_{p,t'}} \rightarrow \left[\bigvee_{t''=t}^{t'-1} \left(\bigvee_a \alpha_{a,t''} \right) \right]$$

where $t' > t$. Interestingly, and somewhat mysteriously, in both applications optimal performance was obtained by adding only a small, randomly-chosen subset of these *long-range dependencies*.

In summary, two encoding properties that seem to make a problem hard for local search are: (i) low solution density, which can sometimes be artificially increased by remodelling the problem; and (ii) the presence of chains of dependent variables, which can either be (a) collapsed by variable elimination or (b) short-circuited by adding long-range dependencies. However, these techniques have an erratic and sometimes very bad effect on DPLL performance [Pre02].

2.4. Desirable properties of CNF encodings

What features of a CNF encoding make it better than another? In this section we discuss features that have been proposed as desirable.

2.4.1. Encoding size

By the *size* of an encoding we may mean the number of its clauses, literals or variables. Firstly, consider the number of clauses. As noted above, this can be reduced by introducing new variables as in the Tseitin encodings , possibly transforming an intractably large encoding into a manageable one. However,

the resulting encoding is not always the best in practice. For example [BM00] describes an encoding of a round robin tournament scheduling problem with n teams that has $O(n^6)$ clauses, which gave much better results than a considerably more compact encoding with auxiliary variables and only $O(n^4)$ clauses. Moreover, adding implied clauses [ABC+02, CI96, KD95], symmetry breaking clauses [CGLR96] and blocked clauses [Kul99] can greatly improve performance.

Secondly, the total number of literals in an encoding, computed by summing the clause lengths, also has a strong effect on runtime overheads (though this is reduced by the implementation technique of watched literals). As a measure of the memory used by an encoding it is more accurate than the number of clauses. However, the same examples show that minimising this measure of size does not necessarily give the best results. Moreover, the CSP support encoding has given superior results to the direct encoding, despite typically containing more literals.

Thirdly, consider the number of variables in an encoding. Several papers have analysed the worst- or average-case runtime of SAT algorithms in terms of the number of variables in the formula (for example [Sch99]), so it seems worthwhile to minimise the number of variables. However, a counter-example is the log encoding and its variants, which generally give poor performance both with DPLL [FP01, VG07] and local search algorithms [EMW97, Hoo98]. There may also be modelling reasons for not minimising the number of variables: Gent & Lynce [GL05] describe a SAT-encoding of the Social Golfer problem that contains several times more variables than necessary, which was found to be more convenient for expressing the problem constraints.

In practice, reducing the size of an encoding is no guarantee of performance, no matter how it is measured. Nevertheless, small encodings are worth aiming for because computational results can be very unpredictable, and all else being equal a smaller encoding is preferable.

2.4.2. Consistency properties

DPLL uses unit propagation extensively, and it is interesting to ask what type of consistency reasoning is achieved by unit propagation on a given encoding. The support encoding [Gen02, Kas90] for binary CSPs has the property that unit propagation maintains arc consistency, whereas the direct encoding maintains a weaker form of consistency called forward checking [Wal00], and the log encoding maintains a weaker form still [Wal00]. The encodings of [BHW03] generalise the support encoding to higher forms of consistency, and those of [Bac07] achieve global arc consistency for arbitrary constraints. The cardinality constraint SAT-encoding of [BB03b] has the property that unit propagation maintains generalised arc consistency. Implementing consistency in this way is much easier than directly implementing filtering algorithms, especially if we want to interleave them or combine them with techniques such as backjumping and learning [Bac07].

The strength of consistency achieved by unit propagation is reflected in DPLL performance on many problems. Despite the indirect nature of this implementation technique, in some cases it even outperforms specialised constraint solvers [QW07]. However, it is well-known in Constraint Programming that stronger consistency is not always worth achieving because of its additional overheads, and

[BHW03] report similar findings with some of their encodings. The same is true of implied clauses, symmetry breaking clauses and blocked clauses: adding them can greatly boost the power of unit propagation, but can also have unpredictable effects on performance (see [Pre03b] and the example discussed in Section 2.3.2).

2.4.3. Solution density

Another interesting property of an encoding is its solution density, which can be defined as the number of solutions divided by 2^n where n is the number of SAT variables. The multivalued CSP encoding mentioned in Section 2.2.4 increases solution density and is often used with local search. The idea that higher solution density makes a problem easier to solve seems natural, and is usually assumed to have at least some effect on search performance [CFG+96, CB94, HHI03, Par97, SGS00, Yok97]. It was pointed out by Clark et al. [CFG+96] that one might expect a search algorithm to solve a (satisfiable) problem more quickly if it has higher solution density. In detailed experiments on random problems across solvability phase transitions they did find a correlation between local and backtrack search performance and the number of solutions, though not a simple one. Problem features other than solution density also seem to affect search cost, for example Yokoo [Yok97] showed that adding more constraints to take a random problem beyond the phase transition removes local minima, making them easier for local search algorithms to solve despite removing solutions.

An obvious counter-example to the notion that higher density problems are easier is the CSP log encoding [IM94]. This has only a logarithmic number of variables in the CSP domain sizes, therefore a much higher SAT solution density than the direct encoding for any given CSP. Yet it generally performs poorly with both DPLL and local search algorithms. However, the binary transform of [FP01] is of similar form to the log encoding but with even higher solution density, and gives better results with local search (at least in terms of local search moves) [Pre03a]. It is possible to modify the direct encoding to increase its solution density, but this gives very erratic results [Pre03a].

2.4.4. Summary

It seems that no single feature of an encoding can be used to characterise how good it is. To further confuse the issue, it turns out that a good encoding for one algorithm might be a poor encoding for another algorithm. This was found with the modified models for parity learning and Towers of Hanoi in Section 2.3.4: standard encodings of these problems are hard for local search but can be solved by DPLL, while modified encodings boost local search but can have drastic effects on DPLL performance. Similarly, in Section 2.2.5 different encodings of the at-most-one constraint were found to be better for DPLL and local search. There is also evidence that adding clauses to break symmetry makes a problem harder to solve by local search but easier by DPLL [Pre03b]. Finally, new variables introduced by Tseitin encoding are *dependent* on others, and as mentioned in Section 2.3.4 these are known to slow down local search [KMS97, Pre02, WS02]. Thus the standard Tseitin approach might be inappropriate when combined with local search. However, it does not necessarily harm DPLL performance, because

the selection of these variables for branching can be delayed until unit propagation eliminates them — though this assumes a measure of control over the branching heuristic.

2.5. Conclusion

There are usually many ways to model a given problem in CNF, and few guidelines are known for choosing among them. There is often a choice of problem features to model as variables, and some might take considerable thought to discover. Tseitin encodings are compact and mechanisable but in practice do not always lead to the best model, and some subformulae might be better expanded. Some clauses may be omitted by polarity considerations, and implied, symmetry breaking or blocked clauses may be added. Different encodings may have different advantages and disadvantages such as size or solution density, and what is an advantage for one SAT solver might be a disadvantage for another. In short, CNF modelling is an art and we must often proceed by intuition and experimentation. But enumerating known techniques and results, as we have done in this chapter, helps to clarify the alternatives.

References

[ABC+02] T. Alsinet, R. Béjar, A. Cabiscol, C. Fernàndez, and F. Manyà. Minimal and redundant sat encodings for the all-interval-series problem. In *Fifth Catalonian Conference on Artificial Intelligence*, pages 139–144. Springer, 2002. Lecture Notes in Computer Science vol. 2504.

[AG93] A. Armando and E. Giunchiglia. Embedding complex decision procedures inside an interactive theorem prover. *Ann. Math. Artif. Intell.*, 8(3-4):475–502, 1993.

[AM04] C. Ansótegui and F. Manyà. Mapping problems with finite-domain variables into problems with boolean variables. In *Seventh International Conference on Theory and Applications of Satisfiability Testing*, pages 1–15. Springer, 2004. Lecture Notes in Computer Science vol. 3542.

[ANORC09] R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Cardinality networks and their applications. In *Twelfth International Conference on Theory and Applications of Satisfiability Testing*, pages 167–180. Springer, 2009. Lecture Notes in Computer Science vol. 5584.

[ANORC11a] I. Abío, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Bdds for pseudo-boolean constraints - revisited. In *Fourteenth International Conference on Theory and Applications of Satisfiability Testing*, pages 61–75. Springer, 2011. Lecture Notes in Computer Science vol. 6695.

[ANORC11b] R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Cardinality networks: a theoretical and empirical study. *Constraints*, 16(2):195–221, 2011.

[AS14] I. Abío and P. J. Stuckey. Encoding linear constraints into sat. In *Twentieth International Conference on Principles and Practice of Constraint Programming*, pages 75–91. Springer, 2014. Lecture Notes in Computer Science vol. 8656.

[Bac07] F. Bacchus. Gac via unit propagation. In *Thirteenth International Conference on Principles and Practice of Constraint Programming*, pages 133–147. Springer, 2007. Lecture Notes in Computer Science vol. 4741.

[BB03a] O. Bailleux and Y. Boufkhad. Efficient cnf encoding of boolean cardinality constraints. In *Ninth International Conference on Principles and Practice of Constraint Programming*, pages 108–122. Springer, 2003. Lecture Notes in Computer Science vol. 2833.

[BB03b] O. Bailleux and Y. Boufkhad. Efficient cnf encoding of boolean cardinality constraints. In *Ninth International Conference on Principles and Practice of Constraint Programming*, pages 108–122. Springer, 2003. Lecture Notes in Computer Science vol. 2833.

[BHN14] P. Barahona, S. Hölldobler, and V.-H. Nguyen. Representative encodings to translate finite csps into sat. In *Eleventh International Conference on Integration of AI and OR Techniques in Constraint Programming*, pages 251–267. Springer, 2014. Lecture Notes in Computer Science vol. 8451.

[BHW03] C. Bessière, E. Hebrard, and T. Walsh. Local consistencies in sat. In *Sixth International Conference on Theory and Applications of Satisfiability Testing*, pages 299–314, 2003. Lecture Notes in Computer Science vol. 2919.

[BM00] R. Béjar and F. Manyà. Solving the round robin problem using propositional logic. In *Seventeenth National Conference on Artificial Intelligence*, pages 262–266, 2000. Austin, Texas.

[CB94] J. M. Crawford and A. B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Twelfth National Conference on Artificial Intelligence*, pages 1092–1097. AAAI Press, 1994. vol. 2.

[CFG+96] D. Clark, J. Frank, I. Gent, E. MacIntyre, N. Tomov, and T. Walsh. Local search and the number of solutions. In *Second International Conference on Principles and Practice of Constraint Programming*, pages 119–133, 1996.

[CGLR96] J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *International Conference on Principles of Knowledge Representation and Reasoning*, pages 148–159, 1996.

[Che10] J.-C. Chen. A new sat encoding of the at-most-one constraint. In *Tenth International Workshop of Constraint Modelling and Reformulation*, 2010.

[CI96] B. Cha and K. Iwama. Adding new clauses for faster local search. In *Fourteenth National Conference on Artificial Intelligence*, pages 332–337, 1996. American Association for Artificial Intelligence.

[CKS94] J. M. Crawford, M. J. Kearns, and R. E. Shapire. The minimal

disagreement parity problem as a hard satisfiability problem. Technical report, Computational Intelligence Research Laboratory and AT&T Bell Labs, 1994.

[CZI10] M. Codish and M. Zazon-Ivry. Pairwise cardinality networks. In *Sixteenth international conference on Logic for programming, artificial intelligence, and reasoning*, pages 154–172. Springer, 2010. Lecture Notes in Computer Science vol. 6355.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association of Computing Machinery*, 7(3), 1960.

[EMW97] M. Ernst, T. Millstein, and D. Weld. Automatic sat-compilation of planning problems. In *Fifteenth International Joint Conference on Artificial Intelligence*, pages 1169–1176, 1997. Nagoya, Japan.

[ES06] N. Eén and N. Sörensson. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[FP01] A. Frisch and T. Peugniez. Solving non-boolean satisfiability problems with stochastic local search. In *Seventeenth International Joint Conference on Artificial Intelligence*, 2001. Seattle, Washington.

[FPDN05] A. M. Frisch, T. J. Peugniez, A. J. Doggett, and P. W. Nightingale. Solving non-boolean satisfiability problems with stochastic local search: A comparison of encodings. *J. Autom. Reasoning*, 35:143–179, 2005.

[Gav07] M. Gavanelli. The log-support encoding of csp into sat. In *Thirteenth International Conference on Principles and Practice of Constraint Programming*, pages 815–822. Springer, 2007. Lecture Notes in Computer Science vol. 4741.

[Gen02] I. P. Gent. Arc consistency in sat. In *Fifteenth European Conference on Artificial Intelligence*, pages 121–125. IOS Press, 2002.

[GIM+01] I. P. Gent, R. W. Irving, D. Manlove, P. Prosser, and B. M. Smith. A constraint programming approach to the stable marriage problem. In *Seventh International Conference on Principles and Practice of Constraint Programming*, pages 225–239. Springer-Verlag, 2001. Lecture Notes In Computer Science vol. 2239.

[GL05] I. P. Gent and I. Lynce. A sat encoding for the social golfer problem. In *Workshop on Modelling and Solving Problems with Constraints*, 2005. IJCAI'05.

[GMS03] I. P. Gent, I. McDonald, and B. M. Smith. Conditional symmetry in the all-interval series problem. In *Third International Workshop on Symmetry in Constraint Satisfaction Problems*, 2003.

[GN04] I. P. Gent and P. Nightingale. A new encoding of alldifferent into sat. In *Third International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, 2004. CP'04.

[HHI03] Y. Hanatani, T. Horiyama, and K. Iwama. Density condensation of

boolean formulas. In *Sixth International Conference on the Theory and Applications of Satisfiability Testing*, pages 126–133. Springer, 2003. Lecture Notes In Computer Science vol. 2919.

[Hoo98] H. H. Hoos. *Stochastic Local Search — Methods, Models, Applications*. PhD thesis, Darmstadt University of Technology, 1998.

[IM94] K. Iwama and S. Miyazaki. Sat-variable complexity of hard combinatorial problems. In *IFIP World Computer Congress*, pages 253–258. Elsevier Science B. V., North-Holland, 1994.

[JT96] D. S. Johnson and M. A. Trick, editors. *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge*, volume 26. American Mathematical Society, 1996.

[Kas90] S. Kasif. On the parallel complexity of discrete relaxation in constraint satisfaction networks. *Artificial Intelligence*, 45:275–286, 1990.

[KD95] K. Kask and R. Dechter. Gsat and local consistency. In *Fourteenth International Joint Conference on Artificial Intelligence*, pages 616–622. Morgan Kaufmann, 1995.

[KK07] W. Klieber and G. Kwon. Efficient cnf encoding for selecting 1 from n objects. In *Fourth Workshop on Constraint in Formal Verification*, 2007.

[KMS97] H. Kautz, D. McAllester, and B. Selman. Exploiting variable dependency in local search. In *Poster Sessions of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

[Kul99] O. Kullmann. New methods for 3-sat decision and worst-case analysis. *Theoretical Computer Science*, 223(1-2):1–72, 1999.

[Li00] C. M. Li. Integrating equivalence reasoning into davis-putnam procedure. In *Seventeenth National Conference on Artificial Intelligence*, pages 291–296, 2000.

[LM98] J. Lang and P. Marquis. Complexity results for independence and definability in propositional logic. In *International Conference on Principles of Knowledge Representation and Reasoning*, pages 356–367, 1998.

[MS06] R. Muhammad and P. J. Stuckey. A stochastic non-cnf sat solver. In *Trends in Artificial Intelligence, Ninth Pacific Rim International Conference on Artificial Intelligence*, pages 120–129. Springer, 2006. Lecture Notes in Computer Science vol. 4099.

[MSL07] J. Marques-Silva and I. Lynce. Towards robust cnf encodings of cardinality constraints. In *Thirteenth International Conference on Principles and Practice of Constraint Programming*, pages 483–497. Springer, 2007. Lecture Notes in Computer Science vol. 4741.

[Nad90] B. A. Nadel. Representation selection for constraint satisfaction: a case study using n-queens. *IEEE Expert: Intelligent Systems and Their Applications*, 5(3):16–23, 1990.

[Ngu17] V.-H. Nguyen. Sat encodings of finite-csp domains: A survey. In *Eighth International Symposium on Information and Communication Technology*, pages 84–91. ACM, 2017.

[NM15] V.-H. Nguyen and S. T. Mai. A new method to encode the at-

most-one constraint into sat. In *Sixth International Symposium on Information and Communication Technology*, pages 46–53. ACM, 2015.

[OGMS02] R. Ostrowski, É. Grégoire, B. Mazure, and L. Saïs. Recovering and exploiting structural knowledge from cnf formulas. In *Eighth International Conference on Principles and Practice of Constraint Programming*, pages 185–199. Springer-Verlag, 2002. Lecture Notes in Computer Science vol. 2470.

[OLH+13] T. Ogawa, Y. Liu, R. Hasegawa, M. Koshimura, and H. Fujita. Modulo based cnf encoding of cardinality constraints and its application to maxsat solvers. In *International Conference on Tools with Artificial Intelligence*, pages 9–17. IEEE, 2013.

[Ott97] J. Otten. On the advantage of a non-clausal davis-putnam procedure. Technical report, Technische Hochschule Darmstadt, Germany, 1997.

[Par97] A. Parkes. Clustering at the phase transition. In *Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference*, pages 340–345. AAAI Press / MIT Press, 1997.

[PG86] D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, 1986.

[PJ11] J. Petke and P. Jeavons. The order encoding: From tractable csp to tractable sat. In *Fourteenth International Conference on Theory and Applications of Satisfiability Testing*, pages 371–372. Springer, 2011. Lecture Notes in Computer Science vol. 6695.

[Pre02] S. D. Prestwich. Sat problems with chains of dependent variables. *Discrete Applied Mathematics*, 3037:1–22, 2002.

[Pre03a] S. D. Prestwich. Local search on sat-encoded colouring problems. In *Sixth International Conference on the Theory and Applications of Satisfiability Testing*, pages 105–119. Springer, 2003. Lecture Notes in Computer Science vol. 2919.

[Pre03b] S. D. Prestwich. Negative effects of modeling techniques on search performance. *Annals of Operations Research*, 118:137–150, 2003.

[Pre04] S. D. Prestwich. Full dynamic substitutability by sat encoding. In *Tenth International Conference on Principles and Practice of Constraint Programming*, pages 512–526. Springer, 2004. Lecture Notes in Computer Science vol. 3258.

[Pre07a] S. D. Prestwich. Finding large cliques using sat local search. In F. Benhamou, N. Jussien, and B. O'Sullivan, editors, *Trends in Constraint Programming*, pages 269–274. ISTE, 2007. Chapter 15.

[Pre07b] S. D. Prestwich. Variable dependency in local search: Prevention is better than cure. In *Tenth International Conference on Theory and Applications of Satisfiability Testing*, pages 107–120. Springer, 2007. Lecture Notes in Computer Science vol. 4501.

[PTS07] D. N. Pham, J. R. Thornton, and A. Sattar. Building structure into local search for sat. In *Twentieth International Joint Conference on*

Artificial Intelligence, pages 2359–2364, 2007. Hyderabad, India.

[QW07] C.-G. Quimper and T. Walsh. Decomposing global grammar constraints. In *Thirteenth International Conference on Principles and Practice of Constraint Programming*, pages 590–604. Springer, 2007. Lecture Notes in Computer Science vol. 4741.

[Sch99] U. Schöning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *Fortieth Annual Symposium on Foundations of Computer Science*, pages 410–414. IEEE Computer Society, 1999.

[Seb94] R. Sebastiani. Applying gsat to non-clausal formulas. *Journal of Artificial Intelligence Research*, 1:309–314, 1994. research note.

[SGS00] J. Singer, I. P. Gent, and A. Smaill. Backbone fragility and the local search cost peak. *Journal of Artificial Intelligence Research*, 12:235–270, 2000.

[Sin05] C. Sinz. Towards an optimal cnf encoding of boolean cardinality constraints. In *Eleventh International Conference on Principles and Practice of Constraint Programming*, pages 827–831. Springer, 2005. Lecture Notes in Computer Science vol. 3709.

[Sta01] Z. Stachniak. Non-clausal reasoning with definite theories. *Fundam. Inform.*, 48(4):363–388, 2001.

[TBW04] C. Thiffault, F. Bacchus, and T. Walsh. Solving non-clausal formulas with dpll search. In *Seventh International Conference on Theory and Applications of Satisfiability Testing*, pages 147–156. Springer, 2004. Lecture Notes in Computer Science vol. 3542.

[Tse83] G. Tseitin. On the complexity of derivation in propositional calculus. *Automation of Reasoning: Classical Papers in Computational Logic*, 2:466–483, 1983. Springer-Verlag.

[TTB11] T. Tanjo, N. Tamura, and M. Banbara. A compact and efficient sat-encoding of finite domain csp. In *Fourteenth International Conference on Theory and Applications of Satisfiability Testing*, pages 375–376. Springer, 2011. Lecture Notes in Computer Science vol. 6695.

[TTKB09] N. Tamura, A. Taga, S. Kitagawa, and M. Banbara. Compiling finite linear csp into sat. *Constraints*, 14(2):254–272, 2009.

[Vel05] M. N. Velev. Comparison of schemes for encoding unobservability in translation to sat. In *Asia and South Pacific Design Automation Conference*, pages 1056–1059, 2005.

[Vel07] M. N. Velev. Exploiting hierarchy and structure to efficiently solve graph coloring as sat. In *International Conference on Computer-Aided Design*, pages 135–142. IEEE/ACM, 2007.

[VG88] A. Van Gelder. A satisfiability tester for non-clausal propositional calculus. *Information and Computation*, 79:1–21, 1988.

[VG07] A. Van Gelder. Another look at graph coloring via propositional satisfiability. *Discrete Applied Mathematics*, 156(2):230–243, 2007.

[Wal00] T. Walsh. Sat v csp. In *Sixth International Conference on Principles and Practice of Constraint Programming*, pages 441–456. Springer-Verlag, 2000. Lecture Notes in Computer Science vol.

1894.

[War98] J. P. Warners. A linear-time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters*, 68(2):63–69, 1998.

[WS02] W. Wei and B. Selman. Accelerating random walks. In *Eighth International Conference on Principles and Practice of Constraint Programming*, pages 216–232. Springer, 2002. Lecture Notes in Computer Science vol. 2470.

[Yok97] M. Yokoo. Why adding more constraints makes a problem easier for hill-climbing algorithms: Analyzing landscapes of csps. In *Third International Conference on Principles and Practice of Constraint Programming*, pages 356–370. Springer-Verlag, 1997. Lecture Notes in Computer Science vol. 1330.