Towards CSP Model Reformulation at Multiple Levels of Abstraction

Alan M. Frisch[1], Brahim Hnich[2], Ian Miguel[1], Barbara M. Smith[3], and Toby Walsh[4]

[1] Dept. Computer Science, University of York, UK
[2] Dept. Information Science, Uppsala University, Sweden
[3] School of Computing and Engineering, University of Huddersfield, UK
[4] Cork Constraint Computation Centre, University College Cork, Ireland

Abstract. Experts at modelling constraint satisfaction problems (CSPs) carefully choose model reformulations to reduce greatly the amount of effort that is required to solve a problem by systematic search. It is a considerable challenge to automate such reformulations. A problem may be viewed and reformulated at various levels of abstraction. Equivalent reformulations may be simple at one such level, yet very difficult at another. Therefore we argue that it is essential for a system for the automatic reformulation of CSPs to reason at multiple levels of abstraction. Reformulations can then be made at the level of abstraction at which they are most straightforward. We describe how the CGRASS system, which currently reformulates individual CSP instances, could be augmented to reformulate models at various levels of abstraction and to refine models from one level to a less abstract level. The SONET problem, a realistic combinatorial optimisation problem, is used to illustrate our approach.

1 Introduction

Constraint satisfaction is a successful technology for tackling a wide variety of search problems including resource allocation, transportation and scheduling. Constructing an effective model of a constraint satisfaction problem (CSP) is, however, a challenging task as new users typically lack specialised expertise. One difficulty is in identifying reformulations, which are sometimes complex, that can dramatically reduce the effort needed to solve the problem by systematic search (see, for example, [13]). Such reformulations include adding constraints that are implied by other constraints in the problem, adding constraints that eliminate symmetrical solutions to the problem, removing redundant constraints (where redundant constraints are those which result in no extra pruning, but just add overhead) and replacing constraints with their logical equivalents. Unfortunately, outside a highly focused domain like planning (see, for example, [2]), there has been little research on how to perform such reformulations automatically.

If we assume that our aim is to improve automatically an initial model proposed by the user, it is natural to consider the reformulation of one CSP instance (the user's model) into another. We have investigated this approach [6], and in so

4+	Higher levels
	$\downarrow$`refine` $\uparrow$`abstract`
3	Sets/Multisets/Relations/Functions
	$\downarrow$`refine` $\uparrow$`abstract`
2	A CSP Class
	$\downarrow$`refine` $\uparrow$`abstract`
1	A CSP instance

Fig. 1. Levels of Abstraction of a Problem Involving Constraints

doing have discovered the drawback that some reformulations are very difficult to make automatically to individual instances, involving complex manipulations or long sequences of primitive steps. We will show that this drawback can be overcome by reasoning at higher levels of abstraction, *in addition* to the instance level. We will also gain the additional benefit of being able to reason about an entire *class* of problems. Hence, any inferences made will hold for all instances of that class, where previously a large proportion of inferences were unnecessarily repeated between instances.

2 Reasoning at Higher Levels of Abstraction

A CSP instance is defined in terms of a finite set of variables, each with a finite domain of possible values, and a finite set of constraints specifying the allowed combinations of assignments of values to variables [10]. These problems naturally occur at higher levels of abstraction. Figure 1 presents a number of levels of abstraction at which we might consider an input problem involving constraints. All levels above level 1 can represent a parameterised class of problems. All reformulations made at these levels of abstraction therefore hold for all instances of the class. The horizontal line separates the levels of abstraction at which problems are and are not represented solely in terms of CSP variables. Clearly, however, some set-based models at level 3 should be fairly directly translatable using set variables. We will use `refine`ment operations to reformulate all or part of a model from a higher to a lower level of abstraction. It is possible to perform `abstraction` to make the opposite reformulation, but we do not discuss this operation here. We suggest that the natural language description of a problem from which human modelling often begins is at the top of this hierarchy.

As a short example, consider the well known Golomb ruler problem (006 at `www.csplib.org`). We first discuss some of the possible refinements that can be made. Model $G_{\mathcal{N}}$ is at the natural language level of description. It is a compact model useful for communication between humans.

Model $G_{\mathcal{N}}$ (level $\top$):
- Given n. Put n ticks at integer points on a ruler of size m such that all the inter-tick distances are unique.
- Minimise m.

Model G_A is formalised with mathematical objects, such as sets and functions. Notice the close correspondence between the natural language description and the objects in this model.

Model G_A (level 3):
- Given n, Find $T \subseteq \{0, ..., m\}, |T| = n$
- Minimise m
- distance($\{x, y\}$) $= |x - y|$
- $\forall x, y, x', y' \in T : \{x, y\} \neq \{x', y'\} \rightarrow$ distance($\{x, y\}$) $\neq$ distance($\{x', y'\}$)

The last component is a set of constraints.

Further down the abstraction hierarchy we create model G_B based on quantified constraints. At this point we use CSP variables, enabling the production of a CSP instance once parameters are instantiated and instance data is given.

Model G_B (level 2):
- Given n variables, $\{x_1, x_2, ..., x_n\}$, each with domain $\{0, ..., n^2\}$
- All-different($\{x_1, ..., x_n\}$)
- $\forall i, j, k, l \in \{1, ..., n\} : (i \neq j) \wedge (k \neq l) \wedge (i \neq k \vee j \neq l) \rightarrow |x_i - x_j| \neq |x_k - x_l|$
- Minimise($max_i(x_i)$)

Lastly, model G_C is a 3-tick instance of the Golomb ruler.

Model G_C (level 1):
- Variables: $x_1, x_2, x_3 \in \{0, ..., 9\}$
- All-different($\{x_1, ..., x_n\}$)
- Minimise($max_i(x_i)$)
- $|x_1 - x_2| \neq |x_1 - x_3|$
- $|x_1 - x_2| \neq |x_2 - x_3|$
- $|x_1 - x_3| \neq |x_2 - x_3|$

One could formalise rules to refine G_A to G_B to G_C. We now consider the reformulations possible at each level to produce more effective models. Successful reformulation of a naive model of the Golomb ruler has already been performed by hand [13]. At some levels of abstraction these reformulations are very difficult, at others very straightforward. Useful Golomb Ruler reformulations include:

- Symmetry breaking. In naive models, such as G_B and G_C, the tick variables, $x_1, x_2, ..., x_n$ are symmetrical: in any (non-)solution we can permute their assignments to obtain another (non-)solution. This symmetry is broken by introducing inequalities between ticks: $x_1 \leq x_2 \leq ... \leq x_n$, which can be reformulated to strong inequalities since the x_i are all-different. Note that, in models $G_{\mathcal{N}}$ and G_A, this flaw does not exist: we speak only of a *set* of ticks. Symmetry is introduced only when we decide to represent the set of n ticks by a set of n indexed CSP variables. As we shall see, through careful construction of refinement operations we can systematically break this symmetry as it is introduced in the refinement from G_A to G_B. In model G_C, detecting this symmetry is possible by comparing normalised models before and after two variables are exchanged. The cost of this process increases with n, however. In model G_B this symmetry is difficult to detect.
- Variable introduction/substitution. Quaternary constraints such as those in models G_B and G_C are slow to propagate because of their arity. The introduction of distance variables, e.g. $d_{12} = x_2 - x_1$, reduces the arity of

the inter-tick distance constraints to two, e.g. $d_{12} \neq d_{13}$. It also allows the introduction of an all-different constraint amongst the distance variables, promoting strong propagation. This reformulation is natural in G_A, G_B, G_C, because of the recurrence of $|x_i - x_j|$ and distance($\{x, y\}$) in the set of distance constraints. In model G_C, however, the number of variable introduction/substitution operations grows rapidly with n [6].

- Implied constraint. The assignment of each tick variable is equivalent to a summation of distance variables, e.g. $x_3 = d_{12} + d_{23}$. Since all distance variables are different, this gives stronger propagation than bounds consistency alone. This reformulation relies on the ordering of the tick variables introduced by symmetry breaking, and is therefore best suited to the quantified constraint or instance or level, i.e. models G_B and G_C.

From this short example, it should be clear that no one level of abstraction is the best. Indeed, it may seem that the instance level is no use at all. However, some problems will have instance level features (such as input data) that will mean some reformulations are *only* possible at this level.

3 Automation

We intend to automate reformulations of the type described in the previous section by extending the prototype CGRASS system [6] and combining it with the system for model refinement developed by Hnich [9].

3.1 CGRASS

CGRASS is based on, and extends, Bundy's *proof planning* technology [1]. Proof planning is a technique used to guide the search for proof in automated theorem proving. Common patterns in proofs are identified and encapsulated in *methods*. A proof planner takes a goal to prove, and selects from a database of methods one which matches this goal. The proof planner checks that the pre-conditions of the method hold. If so, it executes the post-conditions. This constructs the output goal or goals. This system can readily be adapted to the reformulation of CSPs. Methods in CGRASS capture patterns in hand-reformulation rather than constructing sub-goals to prove. Hence, CGRASS is forward chaining, reformulating a problem from one model into another.

Proof planning offers several potential advantages over other theorem proving techniques for the task of reformulating CSPs automatically. First, strong method preconditions limit transformations to those that are likely to produce a simplified problem. Second, methods act at a high level, performing complex transformations that might require complex proofs to justify at the individual inference rule level. Finally, search control is cleanly separated from the inference steps. We can therefore try out a variety of search strategies.

Currently, CGRASS operates at only the CSP instance level. The set of methods we have developed thus far at this level is not complete in any sense, but the following are three of the most important types:

Symmetry Often the most useful constraints can only be derived when some or all symmetry has been broken. Hence, CGRASS attempts to detect symmetry as a first step. Symmetrical variables have identical domains and are such that, if all occurrences of the pair in the constraint set are exchanged and the constraint set is re-normalised it returns to its original state. A similar process is used to check for symmetrical sub-terms. Symmetry is broken by partitioning the variables (sub-terms) into equivalence classes. The elements of each class are formed into a list and ordered lexicographically. We then add weak inequality constraints between adjacent variables (sub-terms) in each list, e.g. $x_1 \leq x_2 \leq x_3, \ldots$.

Introduce This method binds a new variable to a non-atomic term that recurs in the constraint set. Variable introduction is a powerful tool which, in combination with `eliminate` (below), can tighten the constraint graph and reduce constraint arity. Tightening the constraint graph means that propagation on the introduced variable's domain has a wider reaching effect. Reducing a constraint's arity means that fewer variables need to be instantiated before it can be used to prune the search space.

Eliminate This method performs Gaussian-like variable elimination. The preconditions identify variables or terms which can be eliminated from a more complex constraint, insisting that the resulting constraint has a smaller size (in terms of number of constituent terms) than the original.

CGRASS has automatically reformulated naive models of small instances of the Golomb ruler [6]. However, the naive model produces such a large volume of input as larger instances are considered that CGRASS is overwhelmed. This highlights the need for reasoning about a class of problems, if possible beginning from a high-level description.

3.2 Automated Refinement

Hnich [9] presents an extended constraint language to support function and relation variables, i.e. reasoning at level 3 of the hierarchy given in Figure 1. An n-ary relation variable takes a value from the set of all possible n-ary relations with arguments from given finite sets, $S_1, S_2, \ldots, S_n$. For instance, the domain of relation variable $R : S_1 \times S_2$ is the power set of $S_1 \times S_2$.

A function variable takes a value from the set of all possible functions from a given finite source set into a given finite target set. For instance, the domain of total function variable $F : S \rightarrow T$, when $S = \{1, 2\}$ and $T = \{1, 2\}$ is:

$$\{\{\langle 1,1\rangle, \langle 2,1\rangle\}, \{\langle 1,2\rangle, \langle 2,2\rangle\}, \{\langle 1,2\rangle, \langle 2,1\rangle\}, \{\langle 1,1\rangle, \langle 2,2\rangle\}\}$$

Function variables may be total or partial, injective, bijective or surjective.

Refinement operations, as used in the previous section, allow the reformulation of function and relation variables into CSP variables and constraints. There are many ways, for example, to refine a function variable. Given a total function variable $F : S \rightarrow T$, three immediate possibilities are:

1. A one-dimensional matrix of variables, $TtoS_m$, indexed by S. Each $TtoS_m[s]$ has T as its domain, where $s \in S$.
2. A two-dimensional 0/1 matrix, F_m, indexed by $S \times T$. $F_m[s,t]{=}1$ indicates that $F(s) = t$, where $s \in S, t \in T$. To preserve the functional property we impose the constraint:

$$\forall s \in S : \sum_{t \in T} F_m[s,t] \leq 1$$

3. A one-dimensional matrix of set variables, $StoT_{ms}$, indexed by T. For each $t \in T$, $StoT_{ms}[t]$ is $\{s \in S | F(s) = t\}$.

Similarly, three possibilities for the refinement of relation variable $R : A \times B$ are:

1. A one-dimensional matrix of set variables, $BtoA_{ms}$ indexed by A. For each $a \in A$, $BtoA_{ms}[a]$ is $\{b \in B | R(a,b)\}$.
2. A two-dimensional 0/1 matrix, R_m, indexed by $A \times B$, where $R_m[a,b]{=}1$ indicates $R(a,b)$, when $a \in A, b \in B$.
3. A one-dimensional matrix of set variables, $AtoB_{ms}$ indexed by B. For each $b \in B$, $AtoB_{ms}[b]$ is $\{a \in A | R(a,b)\}$.

3.3 Automatic Reformulation at Multiple Levels of Abstraction

Automated reformulation via CGRASS requires the present system to be augmented in two ways. First, it must be able to represent and reason with higher level constructs such as sets and functions. This is in addition to its current ability to reason at the CSP instance level, allowing reformulation to take place at the level at which it is most straightforward.

Second, refinement operations, such as those proposed by Hnich [9] and illustrated in section 2, must be incorporated into CGRASS. Preconditions of a class of specialised refinement methods would consider the current problem state: if no useful reformulations can be made at the current level of abstraction, some or all of the problem should be refined to the level below. The refinement methods would be crafted to avoid common modelling pitfalls, such as the introduction of symmetry (see section 5), as we move to a more concrete level of abstraction.

Some of the reformulation operations will introduce branch points. For instance, in the previous section we described three possible refinements of a function variable. The augmented system will therefore produce multiple models by following each alternative branch. Currently, we aim to make the system as comprehensive as possible, i.e. that a subset of the reformulated models are those that would be produced by an expert modeller by hand. In future we will introduce mechanisms to be more selective about the models produced.

4 The SONET Problem

To illustrate our approach, we consider the SONET fibre-optic communications problem [11]. Model $S_{\mathcal{N}}$ comprises the natural language description:

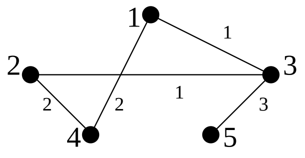

Fig. 2. Demand pairs in an example instance of the SONET problem

– In a communications network, there are client nodes and known levels of demand between pairs of nodes. However, traffic can only be routed between pairs of distinct nodes if they are installed on the same SONET ring. A node is installed on a SONET ring via a dedicated add-drop multiplexer (ADM). Each node may be installed on multiple rings and demand between a pair of nodes may be split over several rings. The maximum number of rings available is known. Each ring has a capacity in terms of the volume of traffic and the number of nodes that can be installed on it. The objective is to minimise the number of ADMs used.

Consider an instance of the SONET problem with 5 nodes and 3 rings, where each ring is able to accommodate 4 nodes and 6 units of traffic. The demands between nodes are presented in Figure 2. An optimal solution using only 6 ADMs (in this solution only two of the three rings are used) is presented in Figure 3. This instance will be used to illustrate the reformulated models throughout.

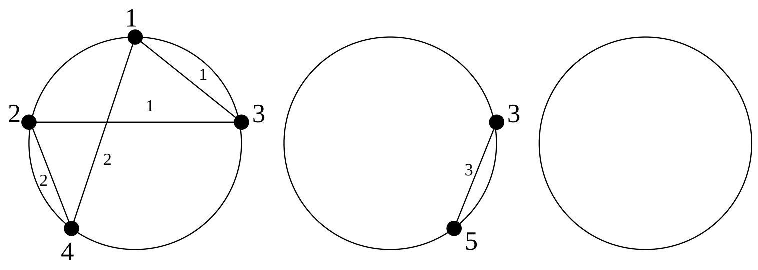

Fig. 3. An optimal solution to the example instance of the SONET problem

5 Reformulating the Simplified SONET Problem

For clarity, this paper focuses on a simplified version of the SONET problem, ignoring actual demand levels, following [12]. The full problem is handled similarly, but all major points are covered by reformulating the simplified problem.

5.1 S_A: A Model Based on Sets, Relations and Functions

Directly from the natural language description, the following sets can be defined:

- R of rings.
- N of nodes.
- D of unordered node pairs, $\{n, n'\}$ where $n, n' \in N$, that must communicate.

We represent the assignment of nodes to rings as a relation, since a node can be installed on multiple rings.

- *rings-nodes* : $R \times N$

Given D and $c \geq 2$, the uniform node capacity per ring, the problem constraints can be expressed:

$$\text{Minimise}(|\textit{rings-nodes}|) \tag{1}$$

$$\forall r \in R : |\textit{rings-nodes}(r, \_)| \leq c \tag{2}$$

$$\forall \{n, n'\} \in D : \textit{rings-nodes}(\_, n) \cap \textit{rings-nodes}(\_, n') \neq \emptyset \tag{3}$$

where $\textit{rings-nodes}(\_, n)$ is the projection of the *rings-nodes* relation onto $n \in N$, that is $\{r \in R | \textit{rings-nodes}(r, n)\}$. The reformulation to produce S_A is little work for a user, but provides a foundation for further useful reformulation.

Implied Constraints Several implied constraints can be derived straightforwardly from S_A. The SONET problem broadly conforms to the pattern of a bin-packing problem. Hence, we focus on the capacity of the rings (bins) and the number of ADMs required per node (items). It would be useful if an automated system were able to detect such common patterns to guide reformulation.

We begin with a lower bound on the number of ADMs required for each node.

i) From (3), the (separate) projection of *rings-nodes* onto certain pairs of elements, $n, n' \in N$, must result in subsets of R with a non-empty intersection. That is, *rings-nodes* relates $\{n, n'\} \in D$ to at least one common ring.
ii) The number of occurrences of n in a pair $\{n, n'\} \in D$, $|\{n' \in N | \{n, n'\} \in D\}|$, is the cardinality of the *partner* set of n, each element of which must be related to at least one common ring with n by *rings-nodes*.
iii) (2) is a capacity constraint. It specifies that the maximum number of nodes that can be related with a given ring, r, by *rings-nodes* is at most c.
iv) Since n must be related to at least one common ring with each element of its partner set, this reduces the effective capacity of each ring to $c - 1$.
v) Simple division now provides a lower bound on the number of ADMs per node n, equivalently the number of rings on which n is installed:

$$\forall n \in N : \left\lceil \frac{|\{n' \in N | \{n, n'\} \in D\}|}{c - 1} \right\rceil \leq |\textit{rings-nodes}(\_, n)| \tag{4}$$

The total number of ADMs is the cardinality of *rings-nodes*, which is to be minimised from (1). Hence, it is natural to try and establish a lower bound.

i) From above we have a minimum number of ADMs required per node.
ii) From (4), the minimum for each node n depends only on the composition of D, which is static. Hence the lower bound is a simple summation:

$$\sum_{n \in N} \left\lceil \frac{|\{n' \in N | \{n, n'\} \in D\}|}{c-1} \right\rceil \leq \sum_{n \in N} |\mathit{rings\text{-}nodes}(\_, n)| \tag{5}$$

We use (5) to establish a lower bound on the number of 'open' rings, i.e. those rings with at least one node installed.

i) From above we have a minimum total number of ADMs.
ii) Each ADM is installed on a ring via *rings-nodes*.
iii) (2) gives the capacity, c, of each ring.
iv) Simple division gives the bound:

$$\left\lceil \frac{\sum_{n \in N} \left\lceil \frac{|\{n' \in N | \{n, n'\} \in D\}|}{c-1} \right\rceil}{c} \right\rceil \leq |\{r \in R | \mathit{rings\text{-}nodes}(r, \_) \neq \emptyset\}| \tag{6}$$

Finally, we restrict the assignment of nodes onto rings.

i) To satisfy (3), the intersection of the projection of *rings-nodes* onto each of a *pair* of nodes, $\{n, n'\} \in D$, must be non-empty.
ii) Consider a tuple, $\langle r \in R, n \in N \rangle$, of *rings-nodes*. If *rings-nodes* does not also relate some $n' \in N$ with r, where $\{n, n'\} \in D$, then this tuple does not satisfy any instance of (3).
iii) From (1) we are minimising the size of *rings-nodes*. Hence:

$$\forall n \in N, r \in R : \mathit{rings\text{-}nodes}(r, n) \rightarrow |\{n' \in N | \{n, n'\} \in D \wedge \mathit{rings\text{-}nodes}(r, n')\}| > 0 \tag{7}$$

In the following subsections, we refine S_A to five different CSP models.

5.2 S_B: A Matrix Model

The *rings-nodes* relation can be refined into a two dimensional matrix of 0/1 variables, $\mathit{rings\text{-}nodes}_m$, where $\mathit{rings\text{-}nodes}_m[r, n]$ denotes the element at the intersection of the rth column and nth row. To perform this refinement, all references to *rings-nodes* must be replaced with $\mathit{rings\text{-}nodes}_m$. Node capacity constraints are on the cardinality of subsets of N and the objective minimises the cardinality of *rings-nodes*. Each column r of $\mathit{rings\text{-}nodes}_m$ corresponds to the characteristic function for the set of nodes installed on ring r (i.e. $\mathit{rings\text{-}nodes}(r, \_)$), and similarly for each row n, so reformulating (1) and (2) is straightforward:

$$\text{Minimise}(\sum_{r \in R} \sum_{n \in N} \mathit{rings\text{-}nodes}_m[r, n]) \tag{8}$$

$$\forall r \in R : \sum_{n \in N} \mathit{rings\text{-}nodes}_m[r, n] \leq c \tag{9}$$

The demand constraints require the intersection of subsets of N to be non-empty. When using characteristic functions, (3) is easily represented via scalar products, which are the cardinality of the intersection:

$$\forall \{n, n'\} \in D : \text{scalar-product}(\textit{rings-nodes}_m[\_, n], \textit{rings-nodes}_m[\_, n']) \neq 0 \quad (10)$$

where $\textit{rings-nodes}_m[\_, n]$ denotes the nth row of the $\textit{rings-nodes}_m$ matrix. S_B is a basic version of that used in [11]. Indeed, matrix models in general are a common pattern in constraint programming [3].

In our example instance, $\textit{rings-nodes}_m$ is instantiated to a 3×5 matrix of 0/1 variables. The total number of '1' entries is minimised and the rows have a maximum sum of 4 (node capacity). Scalar product constraints ensure that all node (row) pairs that must communicate have at least one '1' entry in common.

As noted above, each row (column) of $\textit{rings-nodes}_m$ is equivalent to the characteristic function for the projection of *rings-nodes* onto an element of N (R). A bound on the cardinality of such a projection is easily enforced using a summation on a row or column. Hence (4) and (5) and (7) are refined to:

$$\forall n \in N : \left\lceil \frac{|\{n' \in N | \{n, n'\} \in D\}|}{c-1} \right\rceil \leq \sum_{r \in R} \textit{rings-nodes}_m[r, n] \quad (11)$$

$$\sum_{n \in N} \left\lceil \frac{|\{n' \in N | \{n, n'\} \in D\}|}{c-1} \right\rceil \leq \sum_{n \in N} \sum_{r \in R} \textit{rings-nodes}_m[r, n] \quad (12)$$

$$\forall n \in N, r \in R : \textit{rings-nodes}_m[r, n] = 1 \rightarrow \sum_{n' \in N | \{n, n'\} \in D} \textit{rings-nodes}_m[r, n'] > 0 \quad (13)$$

Constraint (6) is on the cardinality of a subset of R. For each $r \in R$, membership of this subset is predicated on a non-empty projection of *rings-nodes* onto r. This translates to a non-zero sum of a column of $\textit{rings-nodes}_m$. Hence, we must constrain the number of columns of $\textit{rings-nodes}_m$ that meet this criterion. One way to specify (6) is then to introduce a set variable, $\textit{open-rings}_s$, as follows:

$$\forall r \in R : \sum_{n \in N} \textit{rings-nodes}_m[r, n] \neq 0 \leftrightarrow r \in \textit{open-rings}_s \quad (14)$$

$$\left\lceil \frac{\sum_{n \in N} \left\lceil \frac{|\{n' \in N | \{n, n'\} \in D\}|}{c-1} \right\rceil}{c} \right\rceil \leq |\textit{open-rings}_s| \quad (15)$$

Alternatively, a one-dimensional matrix, $\textit{open-rings}_m$ (essentially the characteristic function for the set variable) indexed by R, could be used:

$$\forall r \in R : \sum_{n \in N} \textit{rings-nodes}_m[r, n] \neq 0 \leftrightarrow \textit{open-rings}_m[r] = 1 \quad (16)$$

$$\left\lceil \frac{\sum_{n \in N} \left\lceil \frac{|\{n' \in N | \{n, n'\} \in D\}|}{c-1} \right\rceil}{c} \right\rceil \leq \sum_{r \in R} \textit{open-rings}_m[r] \quad (17)$$

Breaking Symmetry A potential pitfall of refinement to a matrix model is symmetry on the rows and/or columns of the matrix. If the elements of R (N) are indistinguishable, the columns (rows) of any (non-)solution may be permuted to generate another (non-)solution. To determine whether elements of R and N are indistinguishable, we examine S_A prior to refinement. The elements of R are not distinguished by any of the constraints or relations. However, the elements of N could be distinguished by D, which is instance-specific.

If we can identify similar cases where the target objects of refinements may contain symmetry, we can build the means to detect and break the symmetry into the refinement methods. This avoids the (potentially expensive) problem of symmetry detection at a lower level following refinement. Several alternative symmetry breaking methods exist, providing further branching points in the space of reformulations. When symmetry depends on instance-specific information, preconditions must be placed on the methods used to break symmetry. As a higher level model is refined into a CSP instance, the preconditions are tested and symmetry is broken among the objects that are symmetrical in this instance.

For example, the symmetry among the rings can be eliminated using Symmetry Breaking During Search (SBDS) [8]. SBDS takes in a description of the action of each symmetry on decisions made during search (i.e. to assign or not to assign a value to a variable). Once a decision has been explored, if the search backtracks to explore its alternative, SBDS ensures that further decisions symmetric to those already explored will not be explored in future. In our example, we must describe the effect on the assignment of a value to a variable of a transposition of a pair of rings. To automate this process, our refinement methods must generate the descriptions of each symmetry (e.g. interchangeable columns).

Alternatively [4], a lot of row/column matrix symmetry can be broken effectively using lexicographic ordering constraints. We treat a whole row (column) as a binary number and ensure that indistinguishable rows (columns) are ordered. With respect to our example, we impose the constraint that all columns are lexicographically ordered as follows, since the rings are indistinguishable:

$$\forall r \prec r' \in R : \textit{rings-nodes}_m[r, \_] \geq_{\text{lex}} \textit{rings-nodes}_m[r', \_] \tag{18}$$

where $\textit{rings-nodes}_m[r, \_]$ is the rth column of $\textit{rings-nodes}_m$, $r \prec r'$ denotes that r is less than and adjacent to r' in a fixed arbitrary total ordering of R, and $\geq_{\text{lex}}$ denotes lexicographically greater than or equal to, enforceable by the algorithm of [5]. The refinement of (6) can now be simplified. Given the decreasing order imposed in (18) and a lower bound of α, by the transitivity of $\geq_{\text{lex}}$ it is sufficient to insist that the first α columns of $\textit{rings-nodes}_m$ have non-zero sums.

When using lexicographic ordering constraints it is important to choose variable and value ordering heuristics which complement rather than compete with the constraints. It is also important, when lexicographically ordering both rows and columns of a matrix, to ensure that the orderings do not conflict [4].

Generally, disjoint subsets of a set of objects may be symmetrical (*partial* symmetry). If so, we create equivalence classes of indistinguishable objects and break the symmetry among their elements. We might use SBDS, or order each class arbitrarily and lexicographically order adjacent elements. In our example, $\{n_1, n_2\}$ is an equivalence class, since D does not distinguish the two nodes. Hence, symmetry can be broken between the corresponding rows of $\textit{rings-nodes}_m$.

A similar type of constraint, in that it restricts the set of solutions to those of a certain form, may also be added. If a solution exists, then there is a solution in which the contents of pairs of rings, the combined sum of whose installed nodes is less than c, are merged. Generally, rings can be merged in several ways. However, the following is simple and restricts search:

$$\begin{aligned}\forall \{r, r'\} \subseteq R : \sum_{n \in N} \textit{rings-nodes}_m[r, n] > 0 \wedge \sum_{n \in N} \textit{rings-nodes}_m[r', n] > 0 \rightarrow \\ \sum_{n \in N} \textit{rings-nodes}_m[r, n] + \textit{rings-nodes}_m[r', n] > c \quad (19)\end{aligned}$$

5.3 S_C: A Matrix and Set Variable (Rings) Model

We can also refine S_A using set variables to represent the *rings-nodes* relation. This uses either a one-dimensional matrix of set variables $\textit{nodesOnRing}_{ms}$ indexed by R, or $\textit{ringsWithNode}_{ms}$ indexed by N. The matrix $\textit{nodesOnRing}_{ms}[r]$ contains the set of nodes installed on r and $\textit{ringsWithNode}_{ms}[n]$ contains the set of rings on which n is installed. This section discusses the model based on $\textit{nodesOnRing}_{ms}$ and the next discusses the model based on $\textit{ringsWithNode}_{ms}$.

The objective and node capacity constraints on each ring are easily stated:

$$\text{Minimise}(\sum_{r \in R} |\textit{nodesOnRing}_{ms}[r]|) \quad (20)$$

$$\forall r \in R : |\textit{nodesOnRing}_{ms}[r]| \leq c \quad (21)$$

The demand constraints are more difficult to specify, since each constrains the set variable representing an unspecified ring to contain a particular pair of nodes. Therefore, we state the demand constraints on $\textit{rings-nodes}_m$ (section 5.2). Channelling constraints keep the two representations consistent:

$$\forall r \in R : \; n \in \textit{nodesOnRing}_{ms}[r] \leftrightarrow \textit{rings-nodes}_m[r, n] = 1 \quad (22)$$

Implied constraints (4-7) are also most easily stated on $\textit{rings-nodes}_m$. Ring set variables allow us to simplify the content merging constraint (19):

$$\begin{aligned}\forall \{r, r'\} \subseteq R : |\textit{nodesOnRing}_{ms}[r]| > 0 \wedge |\textit{nodesOnRing}_{ms}[r']| > 0 \rightarrow \\ |\textit{nodesOnRing}_{ms}[r]| + |\textit{nodesOnRing}_{ms}[r']| > c \quad (23)\end{aligned}$$

For our example, in addition to the 3×5 matrix, three set variables (one per ring) are created. Each has a maximum of c elements drawn from N.

5.4 S_D: A Matrix and Set Variable (Nodes) Model

The dual model to S_C comprises a one-dimensional matrix of set variables, *ringsWithNode*$_{ms}$, where *ringsWithNode*$_{ms}[n]$ contains the set of rings to which n is assigned. The demand constraints are easily stated:

$$\forall \{n, n'\} \in D : |\mathit{ringsWithNode}_{ms}[n] \cap \mathit{ringsWithNode}_{ms}[n']| \geq 1 \quad (24)$$

However, the node capacity constraints and objective are difficult to state. Once again, it is easiest to use the *rings-nodes*$_m$ matrix and channel:

$$\forall n \in N : \; r \in \mathit{ringsWithNode}_{ms}[n] \leftrightarrow \mathit{rings\text{-}nodes}_m[r, n] = 1 \quad (25)$$

Implied constraints (4) and (5) are easily stated on the node set variables:

$$\forall n \in N : \left\lceil \frac{|\{n' \in N | \{n, n'\} \in D\}|}{c-1} \right\rceil \leq |\mathit{ringsWithNode}_{ms}[n]| \quad (26)$$

$$\sum_{n \in N} \left\lceil \frac{|\{n' \in N | \{n, n'\} \in D\}|}{c-1} \right\rceil \leq \sum_{n \in N} |\mathit{ringsWithNode}_{ms}[n]| \quad (27)$$

Implied constraints (6) and (7) and the content merging constraint (19) are easier to state on *rings-nodes*$_m$ as per S_B.

With respect to our example, in addition to the 3×5 matrix, five set variables (one per node) are created, each drawing its elements from R.

5.5 S_E: A Dual Set Variable Model

S_E comprises a dual model containing both node and ring set variables. The problem constraints combine S_C (20 and 21) and S_D (24). We must channel between the two matrices of set variables to maintain consistency:

$$\forall n \in N, r \in R : n \in \mathit{nodesOnRing}_{ms}[r] \leftrightarrow r \in \mathit{ringsWithNode}_{ms}[n] \quad (28)$$

Implied constraints (4) and (5) are stated as per S_D (26, 27). Implied constraint (6) can be stated using *open-rings*$_s$ (15) or *open-rings*$_m$ (17) as follows:

$$\forall r \in R : |\mathit{nodesOnRing}_{ms}[r]| \neq 0 \leftrightarrow r \in \text{open-rings}_s \quad (29)$$

$$\forall r \in R : |\mathit{nodesOnRing}_{ms}[r]| \neq 0 \leftrightarrow \text{open-rings}_m[r] = 1 \quad (30)$$

However, if we break symmetry between the rings, it is sufficient to specify that the first α ring set variables are non-empty, as per section 5.2. The content merging constraint (19) is stated as per S_C (23).

Without *rings-nodes*$_m$, it is more difficult to refine implied constraint (7):

$$\forall n \in N : n \in \mathit{nodesOnRing}_{ms}[r] \rightarrow \exists n' \in \mathit{nodesOnRing}_{ms}[r] \wedge \{n, n'\} \in D \quad (31)$$

5.6 S_F: A Matrix and Dual Set Variable Model

Finally, S_F comprises a dual model containing *rings-nodes*$_m$ and both node and ring set variables. We must channel to maintain consistency:

$$\forall n \in N, r \in R : \textit{rings-nodes}_m[r, n] = 1 \leftrightarrow n \in \textit{nodesOnRing}_{ms}[r] \qquad (32)$$

$$\forall n \in N, r \in R : n \in \textit{nodesOnRing}_{ms}[r] \leftrightarrow r \in \textit{ringsWithNode}_{ms}[n] \qquad (33)$$

The problem constraints, implied constraints (4-6) and the content merging constraint (19) are stated as per S_E. Implied constraint (7) is easier to state on *rings-nodes*$_m$, as per S_B (13). This model is close to that used in [12].

5.7 Simplified SONET: Summary of Reformulation

Table 1 summarises our models of the simplified SONET problem. Encouragingly, two CSP class level models, S_B and S_F, closely resemble models created by experts in Operations Research [11] and Constraint Programming [12].

Model	Level	Characteristics
$S_{\mathcal{N}}$	$\top$	Natural Language
S_A	3	Sets, Relations and Functions
S_B	2	Matrix
S_C		Matrix + Ring Set Variables
S_D		Matrix + Node Set Variables
S_E		Ring Set + Node Set Variables
S_F		Matrix + Ring Set + Node Set Variables

Table 1. Reformulated Simplified SONET models.

6 Conclusion

We have considered the reformulation of constraint satisfaction problems. We argued that, for effective reformulation, it is crucial to view a problem at multiple levels of abstraction, some more abstract than a CSP. We identified refinement as the process of progressively moving to more concrete levels of abstraction and showed that mechanisms for dealing with some common modelling problems, such as symmetry, can be embedded into refinement rules. Furthermore, we examined the addition of implied constraints, and their reformulation among alternative models, to reduce search. Proposed improvements to the CGRASS system [6] should allow reformulations outlined here to be made automatically.

The SONET problem [11] was used to illustrate how such a system would work. We produced multiple models of the simplified version of this problem by hand, refining a natural language description into a high level model and working through the alternative reformulations. The next step is to formalise fully the reformulations we use and automate their selection and application. The models produced should then be evaluated experimentally to determine which parts of each are the most beneficial/detrimental to search. Through this process, and similar analyses of other problems, we can start to establish patterns in both good and bad models. This is an important step since a more complex input problem might lead to many candidate models by this method, some of which are far better than others. By developing means to evaluate models statically, we can be more selective about the models produced during reformulation itself.

Acknowledgements This work is partially supported by grants 221-99-369 of VR (Sweden) and GR/N16129 of UK EPSRC. The last author is supported by Science Foundation Ireland. We thank our anonymous referees.

References

1. A. Bundy. A Science of Reasoning. J-L. Lassez and G, Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 178–198, 1991.
2. M.D. Ernst, T.D. Millstein, and D.S. Weld. Automatic SAT-compilation of planning problems. *Proc. 15th International Joint Conference on AI*, pages 1169–1176, 1997.
3. P. Flener, A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, T. Walsh. Matrix Modelling: Exploiting Common Patterns in Constraint Programming *Proc. International Workshop on Reformulating Constraint Satisfaction Problems*, 2002.
4. P. Flener, A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking Row and Column Symmetries in Matrix Models. *Proc. 8th International Conference on Principles and Practice of Constraint Programming* LNCS 2470, 2002.
5. A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, T. Walsh. Global Constraints for Lexicographic Orderings. *Proc. 8th International Conference on Principles and Practice of Constraint Programming* LNCS 2470, 2002.
6. A.M. Frisch, I. Miguel, and T. Walsh. CGRASS: A System for Transforming Constraint Satisfaction Problems. *Proc. Joint Workshop of the ERCIM/CologNet area on Constraint Solving and Constraint Logic Programming*, pages 23–26, 2002.
7. I. P. Gent. Heuristic Solution of Open Bin Packing Problems. *Journal of Heuristics*, 3-4, pages 299–304, 1998.
8. I. P. Gent and B. M. Smith. Symmetry Breaking During Search in Constraint Programming. *Proc. European Conference on AI*, pages 599–603, 2000.
9. B. Hnich. *Function Variables for Constraint Programming.* PhD Thesis, University of Uppsala (to appear) 2002.
10. A.K. Mackworth. Constraint Satisfaction Problems. *Encyclopedia of AI*, pages 285–293, 1992.
11. H.D. Sherali and J.C. Smith. Improving Discrete Model Representations via Symmetry Considerations. *Proc. International Symposium on Mathematical Programming*, 2000.
12. B. M. Smith Solve your Problem Faster by Changing the Model. Invited Talk at *ERCIM/CologNet Workshop on Constraint Solving and Constraint Logic Programming*, 2002.
13. B. M. Smith, K. Stergiou, and T. Walsh. Modelling the Golomb Ruler Problem. *Proc. IJCAI-99 Workshop on Non-Binary Constraints.* International Joint Conference on AI, 1999.