Transforming and Refining Abstract Constraint Specifications

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

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

Abstract. Experts at modelling constraint satisfaction problems carefully choose model transformations 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 transformations. A problem may be viewed and transformed at various levels of abstraction. It is often easier to transform an abstract specification rather than a concrete model. We illustrate this point by means of the SONET problem, a realistic combinatorial optimisation problem, which is transformed and subsequently refined into a number of constraint models.

1 Introduction

Constraint satisfaction is a successful technology for tackling a wide variety of search problems including resource allocation, transportation and scheduling. To use constraint technology to solve a problem, the solutions to a problem must first be characterised, or *modelled*, by a set of constraints that they must satisfy. 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 transformations, which are sometimes complex, that can dramatically reduce the effort needed to solve the problem by systematic search (see, for example, [24]). Such transformations 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, [8]), there has been little research on how to perform such transformations automatically.

This paper presents our vision of how such transformations can be performed and is part of our ongoing work towards formalising and automating the modelling process. In particular, we present our vision of transformation through a case study of transforming the Simplified SONET problem. We present the transformations in a somewhat systematic way; in our previous work systematic manual modelling has proved to be an important first step towards the automation of modelling.

Let us outline our vision of transformation. A fundamental observation is that some transformations operate on a particular model of a problem whereas others are model independent. We refer to these two kinds of transformations as *model transformations* and *problem transformations*. Though a problem transformation could also be performed on a particular model, the advantage of transforming the problem is that the benefits provided by the problem transformation will be inherited by all models.

To formalise problem transformations, problems themselves must be presented in a formal language. Such a language must allow problems to be specified at a level of abstraction above that at which modelling decisions are made. For the Simplified SONET example studied in this paper, we use a language that is like an ordinary constraint language except that in addition to the usual atomic variables (variables whose domains comprise atomic elements) and set variables, it allows relation variables (variables whose domains comprise finite relations). The idea of developing constraint languages that support non-atomic variables is gaining momentum. Set variable have been around for several years and are incorporated into constraint toolkits such as ILOG Solver and Eclipse. ESRA [11] supports relation variables and $\mathcal{F}$ supports function variables [16]. ESSENCE [13] takes this this approach to its ultimate conclusion, supporting a wide-variety of variable types—such as sets, multisets, relations, functions, partitions and tuples—as well as arbitrarily-nested variable types, such as set of sets and set of set of tuples. Frisch *et al.* [13] propose ESSENCE as an appropriate language for expressing abstract problem specifications.

After presenting the abstract problem specification of the Simplified SONET problem, our case study proceeds by transforming it by adding implied and other constraints. Then, from this augmented specification, we generate models of the kind that are supported by existing constraint toolkits such as ILOG Solver and Eclipse. We call these *concrete models* and call the process of generating them *refinement*. We reserve the term *transformation* to refer to operations that change a model or specification but, unlike refinement, do not alter the level of abstraction. Since transformations can and should take place at various levels of abstraction, a study of transformation inevitably involves refinement.

Refining the Simplified SONET specification involves replacing a relation variable—which is not supported by existing toolkits—with a structured collection of atomic variables and set variables. We present three general methods for modelling a relation variable and then use these three methods to systematically generate five concrete models. If the target language into which we are refining does not support set variables, these could be refined into atomic variables; doing so is not addressed in this paper.

Before turning to the case study, we examine, and learn from, a previous constraint-transformation system.

2 Why Transform *Abstract* Specifications?

In addition to the reasons discussed in the previous section, our architecture for transformation and refinement has been motivated by experience with the CGrass (Constraint Generation And Symmetry-breaking [14]) system. CGrass transforms constraint models of problem *instances* in order to make them easier to solve. This section describes CGrass and then explains why, on its own, it is insufficient to perform all desired transformations. It is these deficiencies that motivate the approach to transformation advocated by this paper.

The input to CGrass is a problem instance expressed in a language that does not allow *schematic* constraints. To see what this means consider the straightforward model of the Golomb Ruler problem where the the goal is to find values for n integer variables, $x_1, \ldots, x_n$, so that the "distance" between any two distinct pairs of them is distinct. We can specify this by a constraint schema as

$$\begin{aligned}&\forall i, j, k, l \in \{1, ..., n\} \\ &\quad (i \neq j) \wedge (k \neq l) \wedge (i \neq k \vee j \neq l) \rightarrow |x_i - x_j| \neq |x_k - x_l|\end{aligned} \tag{1}$$

For a problem instance, n would be replaced by a constant specified by the instance data. A schema is a linguistic shorthand for generating a set of constraints. For the instance where n is 3, this schema generates 3 constraints:

$$\begin{aligned}&|x_1 - x_2| \neq |x_1 - x_3| \\ &|x_1 - x_2| \neq |x_2 - x_3| \\ &|x_1 - x_3| \neq |x_2 - x_3|.\end{aligned}$$

CGrass works in a forward-chaining manner, selectively applying the heuristically most-promising rule at each iteration to improve a model gradually. A history mechanism prevents cycles of transformations. After each transformation the model is put into a normal form based on that used by the HartMath (`www.hartmath.org`) computer algebra system; this makes it easier to test which transformations can be applied next. CGrass can perform a range of transformations, including adding implied constraints, introducing auxiliary variables and introducing symmetry-breaking constraints. The last of these is performed by testing whether the model is unchanged when two candidate expressions are exchanged throughout and normalisation is performed. This is an expensive method of symmetry detection, and therefore influenced our decision to focus on refinement-based modelling, where much symmetry can be detected easily as it is introduced. Nonetheless, when applied to a naive formulation of the Golomb ruler problem, CGrass is able to produce a far more efficient model that is close to that produced by human experts.

Though CGrass has successfully transformed some problem instances into much more efficient ones, it has some drawbacks. Since it transforms individual problem instances, much effort is repeated as each instance of a problem class is transformed. Therefore, it is desirable to perform transformations on problems, where possible, rather than instances.

Though the job that CGrass performs is simplified because it does not have to reason with quantifiers, some transformations are difficult to perform because they involve recognising a pattern within the non-schematic specification. For example, one useful transformation is to replace a clique of not-equals constraints with an all-different constraint.

Furthermore, instance specifications that do not use schemas can grow extremely large. For example, in the Golomb Ruler Problem the number of inequalities generated by schema (1) is $\Omega(n^4)$. Thus some types of transformations will need to be performed $\Omega(n^4)$ times. The performance improvement realised by the transformed instance specification can be greatly diminished by the time it takes to perform all the transformations.

Our conclusion from the CGrass work is that, where possible, it is better to transform schematic specifications than non-schematic specifications, and, where possible, it is better to transform problem specifications than instance specifications. There may be cases where it is necessary to transform instances—schematic or non-schematic—such as when a transformation is only sanctioned by the particular instance data. We therefore do not see the problem transformations advocated by this paper as fully replacing the need or desirability of all CGrass-style transformations.

3 Specifying the SONET Problem

To illustrate our approach, we consider the SONET fibre-optic communications problem [21].

> 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 1. An optimal solution using only 6 ADMs (in this solution only two of the three rings are used) is presented in Figure 2. This instance will be used to illustrate the transformed models throughout.

For clarity, this paper focuses on a simplified version of the SONET problem, ignoring actual demand levels, following [22]. The full problem is handled similarly, but all major points are covered by transforming the simplified problem.

3.1 S_A: An Abstract Problem Specification

An instance of the SONET problem is identified by four parameters: *nnodes*, the number of nodes; *nrings*, the number of rings; *c*, the uniform node capacity

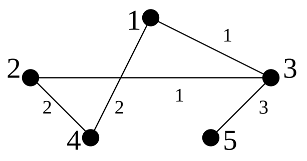

Fig. 1. Demand pairs in an example instance of the SONET problem

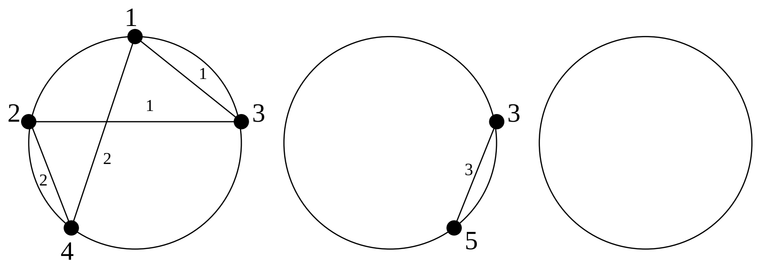

Fig. 2. An optimal solution to the example instance of the SONET problem

per ring; and D, the demand. In particular, D is a set of unordered node pairs, $\{n, n'\}$ where n and n' are nodes that must that must communicate. The decision variable must represent an assignment of nodes to rings; since a node can be assigning to multiple rings, we treat this as a relation, which we call *rings-nodes*, between rings and nodes.

Figure 3 gives S_A, an ESSENCE specification of the simplified SONET problem. Here the nodes and rings are represented by ranges of natural numbers. The objective is to minimise the number of ADMs, which is the number of node attachments and is represented by the cardinality of the *rings-nodes* relation. The capacity constraint is imposed by (3) and the communication constraint is imposed by (4). Note that *rings-nodes*$(\_, n)$ is the projection of the *rings-nodes* relation onto $n \in N$, that is $\{r | \textit{rings-nodes}(r, n)\}$.

`given` *nrings*: `nat`, *nnodes*: `nat`, *capacity*: `nat`
`letting` N `be` $1..\textit{nnodes}$, R `be` $1..\textit{nrings}$
`given` D: `set of set (size` 2`) of` N
`find` *rings-nodes*: $R \times N$
`minimising` $|\textit{rings-nodes}|$ (2)
`such that` $\forall\, r \in R \;\; |\textit{rings-nodes}(r, \_)\}| \leq c$ (3)

$$\forall \{n, n'\} \in D \;\; \textit{rings-nodes}(\_, n) \cap \textit{rings-nodes}(\_, n') \neq \emptyset \quad (4)$$

Fig. 3. Specification of the simplfied SONET problem.

4 Transforming the Abstract Problem Specification

This section presents transformations on the abstract specification of the SONET problem, S_A. First, three implied constraints are derived and added to the specification. Then a fourth constraint, which is not implied, is added.

The first implied constraint imposes a lower bound on the number of ADMs required for each node.

i) From (4), 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 $\{n, n'\} \in D$, $|\{n' | \{n, n'\} \in D\}|$, 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) (3) 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:

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

Henceforth, we refer to $\left\lceil \frac{|\{n' | \{n,n'\} \in D\}|}{c-1} \right\rceil$ as $\mathit{ADMMin}_c(n)$. Observe that all the terms contained in $\mathit{ADMMin}_c(n)$ are parameters; hence, for any given problem instance, this is a constant.

The second implied constraint imposes a lower bound on the number of 'open' rings, i.e. those rings with at least one node installed.

i) From (5), it is simple to derive a minimum total number of ADMs.
ii) Each ADM is installed on a ring via *rings-nodes*.
iii) (3) gives the capacity, c, of each ring.
iv) Simple division gives the bound:

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

Henceforth, we refer to $\left\lceil \frac{\sum_{n \in N} \mathit{ADMMin}_{c}(n)}{c} \right\rceil$ as $\mathit{RingMin}_c$. Again, all the terms in this expression are parameters; hence for any given problem instance, this is a constant.

The third implied constraint imposes the constraint that a node should be installed on a ring only if it needs to communicate with one of the other nodes on the ring.

i) To satisfy (4), 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 (4).
iii) From (2) we are minimising the size of *rings-nodes*. Hence:

$$\begin{aligned}&\forall n{\in}N, r{\in}R \\ &\quad \textit{rings-nodes}(r,n) {\rightarrow} |\{n'{\in}N|\{n,n'\} \in D \wedge \textit{rings-nodes}(r,n')\}| > 0\end{aligned} \tag{7}$$

We now consider another constraint, which is obtained by observing that if a solution exists, then there is a solution in which for any two non-empty rings the sum of the number of nodes installed on them exceeds c. This is the case since if two non-empty rings contain no more than c nodes then their nodes can be merged onto a single ring. Thus, as with symmetry-breaking constraints, we can impose a constraint that restricts search to those solutions in which no two rings can be merged.

$$\begin{aligned}\forall \{r, r'\} \subseteq R : (\textit{rings-nodes}(r, \_) \neq \emptyset) \wedge (\textit{rings-nodes}(r', \_) \neq \emptyset) \rightarrow \\ |\textit{rings-nodes}(r, \_)| + |\textit{rings-nodes}(r', \_)| > c\end{aligned} \tag{8}$$

5 Refining the SONET Problem

Given abstract model S_A, based upon the *rings-nodes* relation variable, a process of refinement is necessary to generate models suitable for input to a constraint toolkit. Hnich [16] shows how to refine specifications in an extended constraint language ($\mathcal{F}$) supporting function variables, producing a set of alternative models. At present, there is no mechanism for choosing among the refined models, however. Alternatively, ESRA [11] is an extended constraint language supporting relation variables. Specifications in ESRA are refined to a single constraint model. As noted in the introduction, the ESSENCE language [13] supports a number of abstract variable types. Unlike ($\mathcal{F}$) and ESRA, ESSENCE supports arbitrary nesting of operators and types (e.g. sets of sets of partitions). ESSENCE specifications are refined by the CONJURE system into a set of alternative constraint models.

We focus now on the refinement of S_A in particular. The decision variable in S_A is of type relation. There are three possibilities for refining an arbitrary relation variable $R : A \times B$:

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

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

5.1 S_B: A Matrix Model

The *rings-nodes* relation can be refined into a two dimensional matrix of 0/1 variables, $\textit{rings-nodes}_m$, where $\textit{rings-nodes}_m[r, n]$ 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 $\textit{rings-nodes}_m$. Node capacity constraints are on the cardinality of subsets of N and the objective minimises the cardinality of *rings-nodes*. Each column r of $\textit{rings-nodes}_m$ corresponds to the characteristic function for the set of nodes installed on ring r (i.e. $\textit{rings-nodes}(r, \_)$), and similarly for each row n, so refining (2) and (3) is straightforward:

$$\text{Minimise}(\sum_{r \in R} \sum_{n \in N} \textit{rings-nodes}_m[r, n]) \tag{9}$$

$$\forall r \in R : \sum_{n \in N} \textit{rings-nodes}_m[r, n] \leq c \tag{10}$$

The demand constraint requires the intersection of subsets of N to be non-empty. When using characteristic functions, (4) is easily represented via scalar products, which are the cardinality of the intersection:

$$\forall \{n, n'\} \in D : \text{scalar-prod}(\textit{rings-nodes}_m[\_, n], \textit{rings-nodes}_m[\_, n']) \neq 0 \tag{11}$$

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 [21]. Indeed, matrix models in general are a common pattern in constraint programming [9].

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 (5) and (7) are refined to:

$$\forall n \in N : \textit{ADMMin}_c(n) \leq \sum_{r \in R} \textit{rings-nodes}_m[r, n] \tag{12}$$

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

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:

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

$$RingMin_c \leq |open\text{-}rings_s| \tag{15}$$

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

$$\forall r \in R : \sum_{n \in N} rings\text{-}nodes_m[r, n] \neq 0 \leftrightarrow open\text{-}rings_m[r] = 1 \tag{16}$$

$$RingMin_c \leq \sum_{r \in R} open\text{-}rings_m[r] \tag{17}$$

Finally, (8) is refined into model S_B straightforwardly, as follows:

$$\begin{aligned} &\forall \{r, r'\} \subseteq R \\ &\quad \textstyle\sum_{n \in N} rings\text{-}nodes_m[r, n] > 0 \wedge \sum_{n \in N} rings\text{-}nodes_m[r', n] > 0 \\ &\quad \rightarrow \ \textstyle\sum_{n \in N} rings\text{-}nodes_m[r, n] + rings\text{-}nodes_m[r', n] > c \end{aligned} \tag{18}$$

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 (or 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 into the refinement methods the means to detect and break the symmetry. 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 transformations. When symmetry depends on instance-specific information, preconditions must be placed on the methods used to break symmetry. As a problem is instantiated 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) [15]. 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, the model can be transformed by adding symmetry-breaking constraints that prune some symmetrical assignments from the search space. The symmetry created by interchangeable rows and or columns in a matrix can be dealt with effectively by imposing lexicographic ordering constraints [10]. To do this, we treat a whole row (column) as a binary number and constrain interchangeable rows (columns) to be in lexicographic order. With respect to our example, since the rings are indistinguishable, we impose the constraint that all columns are lexicographically ordered as follows:

$$\forall r \prec r' \in R : \textit{rings-nodes}_m[r, \_] \geq_{\text{lex}} \textit{rings-nodes}_m[r', \_] \tag{19}$$

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 [12]. The refinement of (6) can now be simplified. Given the decreasing order imposed in (19) and a lower bound of $\textit{RingMin}_c$, 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 [10].

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 $\textit{rings-nodes}_m$.

5.2 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:

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

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

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 5.1). Channelling constraints keep the two representations consistent:

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

Implied constraints (5–7) are also most easily stated on $\textit{rings-nodes}_m$. Ring set variables allow us to simplify the content merging constraint (8):

$$\forall \{r, r'\} \subseteq R : |\textit{nodesOnRing}_{ms}[r]| > 0 \wedge |\textit{nodesOnRing}_{ms}[r']| > 0 \rightarrow$$
$$|\textit{nodesOnRing}_{ms}[r]| + |\textit{nodesOnRing}_{ms}[r']| > c \tag{23}$$

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.

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

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

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

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:

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

Implied constraint (5) is easily stated on the node set variables:

$$\sum_{n \in N} \mathit{ADMMin}_c(n) \leq \sum_{n \in N} |\mathit{ringsWithNode}_{ms}[n]| \tag{26}$$

Implied constraints (6) and (7) and the content merging constraint (8) 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.

5.4 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:

$$\forall n \in N, r \in R : n \in \mathit{nodesOnRing}_{ms}[r] \leftrightarrow r \in \mathit{ringsWithNode}_{ms}[n] \tag{27}$$

Implied constraint (5) is stated as per S_D (26). Implied constraint (6) can be stated using either *open-rings*$_s$ (14) or *open-rings*$_m$ (16) as follows:

$$\forall r \in R : |\mathit{nodesOnRing}_{ms}[r]| \neq 0 \leftrightarrow r \in \text{open-rings}_s \tag{28}$$

$$\forall r \in R : |\mathit{nodesOnRing}_{ms}[r]| \neq 0 \leftrightarrow \text{open-rings}_m[r] = 1 \tag{29}$$

However, if we break symmetry between the rings, it is sufficient to specify that the first $\mathit{RingMin}_c$ ring set variables are non-empty, as per section 5.1. The content merging constraint (8) is stated as per S_C (23).

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

$$\begin{aligned} \forall n \in N : \quad & n \in \mathit{nodesOnRing}_{ms}[r] \rightarrow \\ & \exists n' : n' \in \mathit{nodesOnRing}_{ms}[r] \wedge \{n, n'\} \in D \end{aligned} \tag{30}$$

5.5 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:

$$\forall n \in N, r \in R : \textit{rings-nodes}_m[r, n] = 1 \leftrightarrow n \in \textit{nodesOnRing}_{ms}[r] \quad (31)$$

$$\forall n \in N, r \in R : n \in \textit{nodesOnRing}_{ms}[r] \leftrightarrow r \in \textit{ringsWithNode}_{ms}[n] \quad (32)$$

The problem constraints, implied constraints (5, 6) and the content merging constraint (8) 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 [22].

5.6 Simplified SONET: Summary of Transformation

Table 1 summarises our models of the simplified SONET problem. Encouragingly, two models, S_B and S_F, closely resemble models created by experts in Operations Research [21] and Constraint Programming [22].

Model	Characteristics
S_A	Sets and Relations
S_B	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 Variables + Node Set Variables

Table 1. Transformed Simplified SONET models.

6 Related Work in Modelling and Transformation

In addition to the ESRA, $\mathcal{F}$, and ESSENCE languages mentioned throughout, there are several other methods to aid in constraint modelling. This section briefly surveys them.

As early as 1976, Laurière introduced a modelling language called ALICE to formally state a problem [18]. The ALICE language is characterised by the use of sets, set operators, Cartesian product of sets, vectors, matrices, graphs and paths, constants, and functions. Constraints are stated on these mathematical objects using the classical logical connectives ($\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, and $\neg$), the logic quantifiers ($\exists$ and $\forall$), the set operators ($\in$, $\cap$, $\cup$), and the arithmetic operators ($+$, $-$, $*$, $/$, $=$, $\geq$, $<$, and $\sum$).

Tsang *et al.* had two projects related to ours:

- The adaptive constraint satisfaction project [1–3] aimed at systematically mapping problems, in a dynamic manner, to algorithms and heuristics so as to achieve an adaptive constraint satisfaction strategy, thus helping researchers to bring effective algorithms to applications.

– The computer-aided constraint programming [4, 20] project aimed at building a system that encapsulates the entire process of applying CP technology to problems. And by allowing non-expert users to declaratively state their problems, the users may easily experiment with different problem formulations. The system would then assist them to choose an existing solver, as well as in understanding the underlying technology.

The language NP-SPEC is a logic-based executable specification language [7, 6], which allows the user to specify problems that belong to the NP complexity class. In NP-SPEC, metapredicates, called *tailoring* predicates, are used to restrict the extensions of a given predicate. These metapredicates are *subset*, *partition*, *permutation*, and *intfunc*, which can be used to capture certain classes of output. The *subset* predicate has as extensions all subsets of a given set, the *partition* predicate has as extensions all n subsets of a given set such that these subsets form a partition, the *permutation* predicate has as extensions all permutations over a given set, and the *intfunc* predicate has as extensions all functions from a given set into an integer interval.

In [23], a functional specification language called REFINE is used to specify global search problems for a program synthesizer. The REFINE language augments a functional programming language with three type constructors, namely *set*, *sequence*, and *map*, as well as their operations. The *set* type constructor allows the declaration of variables of set type, the *sequence* type constructor allows the declaration of variable of sequence type, and the *map* type constructor allows the declaration of variable of partial function type.

The ALICE, NP-SPEC, and REFINE languages allow variables of types other than integer and integer sets, which allows the statement of problems at a different level of abstraction when compared to current CP languages such as Eclipse.

7 Conclusions and Future Work

We have considered the transformation of constraint satisfaction problems and have seen that transformations can and should be performed at various levels of abstraction. 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.

The Simplified SONET problem [21] was used to illustrate how such a system integrating transformation and refinement could work. We started with an abstract problem specification in the ESSENCE language and transformed the specification by adding implied and other constraints. Then we refined the resulting specification into five alternative models and showed how the refinement process could break the symmetries that it introduces into the model.

Our immediate goal is to formalise fully the transformations we use. Having done so, it will be possible to produce many models of varying quality. The next step is then to develop heuristics to guide refinement and transformation towards good models. To do this, we need to identify patterns in effective models, and in the processes that generate them.

Part of the pattern elicitation process is experimental, for instance comparing the performance of the models of the SONET problem developed here. This comparison is, however, not straightforward. Usually, model utility can only be judged accurately in the context of the intended solution procedure, such as the variable/value heuristics employed. A thorough empirical comparison, therefore, must consider solving each model in a variety of ways. We will perform such a comparison on alternative models of a number of problems from the literature, including SONET.

In some cases, it is possible to show that one model is stronger than another, irrespective of certain aspects of the solution procedure. Recently, for example, alternative models of permutation and injection problems have been studied in the context of a range of constraint propagation algorithms [17]. Combining empirical and theoretical analysis will allow us to develop the means to evaluate models statically, and therefore be more selective about the models produced during refinement itself.

Acknowledgements Brahim Hnich and Toby Walsh are supported by Science Foundation Ireland. Ian Miguel is supported by a UK-Royal Academy of Engineering/EPSRC Post-doctoral Fellowship.

References

1. A. Abbas and E.P.K. Tsang. Toward a general language for the specification of constraint satisfaction problems. In *Proc of the Constraint Programming, Artificial Intelligence and Operations Research (CP-AI-OR) Workshop*, 2001.
2. J.E. Borrett. *Formulation selection for constraint satisfaction problem: a heuristic approach.* PhD Thesis, Department of Computer Science, University of Essex, Colchester, UK, 1998.
3. J.E. Borrett and E.P.K. Tsang. A context for constraint satisfaction problem formulation selection. *Constraints*, 6(4):299-327, 2001.
4. R. Bradwell, J. Ford, P. Mills, E.P.K. Tsang, and R. Williams. An overview of the CACP project: modelling and solving constraint satisfaction/optimisation problems with minimal expert intervention. InProc of the Workshop on Analysis and Visualization of Constraint Programs and Solvers, Constraint Programming 2000, 2000.
5. 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.
6. M. Cadoli and A. Schaerf. Compiling program specification into SAT. *Proc. of ESOP'01.* LNCS 2028. Springer-Verlag, 2001.
7. M. Cadoli, L. Palopoli, A. Schaerf, and D. Vasile. NP-SPEC: An executable specification language for solving all problems in NP. In: G. Gupta (ed), *Proc. of PADL'99*, pp. 16–30. LNCS 1551. Springer-Verlag, 1999.
8. 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.
9. 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.

10. 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.
11. P. Flener, J. Pearson, and M. Agren. Introducing ESRA, a relational language for modelling combinatorial problems. In, M. Bruynooghe (ed), *LOPSTR'03: Revised Selected Papers*, LNCS 3018, 2004.
12. 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.
13. A.M. Frisch, C. Jefferson, B. Martínez-Hernández, I. Miguel. *The Rules of Modelling: Automatic Generation of Constraint Programs.* APES Technical Report 85, 2004.
14. 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*, LNCS 2627, pages 23–26, 2002.
15. I. P. Gent and B. M. Smith. Symmetry Breaking During Search in Constraint Programming. *Proc. European Conference on AI*, pages 599–603, 2000.
16. B. Hnich. *Function Variables for Constraint Programming.* PhD Thesis, University of Uppsala, 2003.
17. B. Hnich, B.M. Smith and T. Walsh. Models of Permutation and Injection Problems. *Journal of Artificial Intelligence Research*, 21, 2004.
18. J-L. Lauriere. A language and a program for stating and solving combinatorial problem. *Artificial Intelligence*, 10(1):29-127, 1978.
19. A.K. Mackworth. Constraint Satisfaction Problems. *Encyclopedia of AI*, pages 285–293, 1992.
20. P. Mills, E.P.K. Tsang, R. Williams, J. Ford, and J. Borrett. EaCL 1.5: An Easy abstract Constraint optimisation Programming Language, Technical Report CSM-324, University of Essex, Colchester, UK, 1999.
21. H.D. Sherali and J.C. Smith. Improving Discrete Model Representations via Symmetry Considerations. *Management Science* 47, pages 1396–1407, 2001.
22. B. M. Smith Search Strategies for Optimization: Modelling the SONET Problem. *Proc. 2nd International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, pages 143–157, 2003.
23. D.R. Smith. The structure and design of global search algorithms. *TR KES.U.87.12*, Kestrel Institute, 1988.
24. 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.