Transforming and Refining Abstract Constraint Specifications

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

[1] Dept. of Computer Science, University of York, UK
[2] Cork Constraint Computation Centre, University College Cork, Ireland
[3] School of Computer Science, University of St Andrews, UK
[4] National ICT Australia and Dept. of CS & E, UNSW, Australia.

Abstract. To use constraint technology to solve a problem, the solutions to the problem must first be characterised, or *modelled*, by a set of constraints that they must satisfy. A significant part of the modelling process can be characterised as *refinement*, the process of generating a concrete model from an abstract specification of the problem. Expert modellers also identify and perform *transformations* that can dramatically reduce the effort needed to solve the problem by systematic search. Through a case study of modelling a simplified version of the SONET fibre-optic communication problem, this paper examines the processes of refinement and transformation, and especially the interaction between the two.

1 Introduction

Constraint programming is a successful technology for tackling a wide variety of combinatorial problems. To use constraint technology to solve a problem, the solutions to the problem must first be characterised, or *modelled*, by a set of constraints on a set of decision variables that they must satisfy. A significant part of the modelling process can be characterised as *refinement*, the process of generating a concrete model from an *abstract* problem specification. Following [16], and the convention in formal methods, by an abstract specification of a constraint problem, we mean simply a representation in which the details (the modelling decisions) have been abstracted away. Refinement adds these details to produce the concrete model (the modelling decisions are made). There are usually many possible refinements of an abstract problem specification; identifying the effective ones often requires considerable expertise.

Expert modellers also identify and perform *transformations*, which are sometimes complex, that can dramatically reduce the effort needed to solve the problem by systematic search (see, for example, [25]). We use the term *transformation* to refer particularly to operations that change a model or specification but, unlike refinement, do not alter the level of abstraction. 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 (i.e., those that yield no extra pruning but add overhead) and replacing constraints with their logical equivalents.

Through a case study of modelling a simplified version of the SONET fibre-optic communication problem [23], this paper examines the processes of refinement and transformation, and especially the interaction between the two. Starting with an abstract specification of the problem, we perform refinements and transformations to produce seven alternative models. These models are *concrete* in that they are similar to the kind that are supported by existing constraint toolkits. We generate the models in an explicit and somewhat systematic way; a sytematic manual exploration of the possible models has proved to be an important first step in our ongoing work towards formalising and automating the modelling process [12].

This case study illustrates the fundamental observation that some transformations operate on a particular (concrete) 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 corresponds to some transformation on a particular model, an advantage of transforming a problem specification is that the benefits of the transformation are inherited by all models. Some transformations can also be performed more easily at the more abstract level of the problem specification.

The case study also shows how a refinement operation can trigger a useful transformation, thus saving the work of searching for it. In particular, we will see a case where a refinement operation that introduces a matrix into a model can easily recognise that the matrix has column symmetry.

Given the ability to generate alternative models, heuristics are needed to guide refinement and transformation towards good models. Towards this goal, we perform an empirical analysis of the generated models to begin to form generalisations about the expected utility of alternative modelling decisions.

2 Specifying the SONET Problem

For illustration, consider the SONET fibre-optic communications problem [22].

> A communications network has client nodes and known levels of demand between pairs of nodes. Traffic can only be routed between two nodes if they are installed, via an add-drop multiplexer (ADM), on the same SONET ring. Each node may be installed on multiple rings and demand between two 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. Objective: minimise the number of ADMs.

It suffices here to consider a simplified version of the SONET problem, previously considered by Smith [23], in which it is known which node pairs must communicate, but demand *levels* are ignored. Consider an instance of the SONET problem with 5 nodes and 2 rings, where each ring is able to accommodate 4

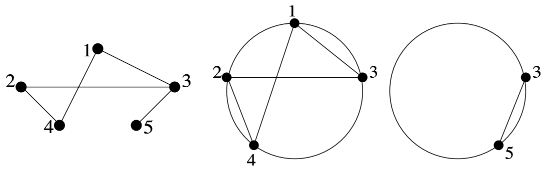

Fig. 1. Demand pairs for and optimal solution of example Simplified SONET instance.

nodes. Figure 1 depicts both the demands between nodes and an optimal solution using only 6 ADMs.

2.1 S_A: An Abstract Problem Specification

This section presents an abstract specification of the Simplified SONET problem, which, in subsequent sections, is refined and transformed to produce concrete models. The problem must be specified at a level of abstraction above that at which modelling decisions are made. We use ESSENCE [12], an abstract constraint language whose key feature is that, in addition to the usual atomic variables (variables whose domains comprise atomic elements), it allows non-atomic variables. In doing so, it builds on the facilities available in constraint toolkits such as Ilog Solver and Eclipse, which have supported set variables for several years, ESRA [10], which supports relation variables, $\mathcal{F}$ [17], which supports function variables, and NP-SPEC [7], which supports a variety of variable types, including partitions. However, ESSENCE is unique in that it supports *arbitrarily-nested* variable types, such as set of sets and set of set of tuples.

An instance of Simplified SONET is identified by four parameters: *nnodes*, *nrings* the number of nodes and rings; c, the uniform node capacity per ring; and D, the demand. D is a set of unordered node pairs, $\{n, n'\}$ where n and n' are nodes that must communicate. The decision variable must represent an assignment of nodes to rings; since a node can be assigned to multiple rings, we treat this as a relation, which we call *rings-nodes*, between rings and nodes.

Figure 2 gives S_A, an ESSENCE specification of the Simplified SONET problem[5]. Here the nodes are represented by N, a range of natural numbers. The rings are represented by R, a set comprising *nrings* unnamed elements. The provision of sets of unnamed objects is a unique and important feature of ESSENCE. It facilitates abstraction in specifications by not forcing the elements of a set to be given arbitrary names that are never used. There is no need to name the rings, since individual rings are not mentioned in the specification.

The objective, (1), is to minimise the number of ADMs, represented by the cardinality of *rings-nodes*. The capacity constraint is imposed by (2) and the communication constraint is imposed by (3). To clarify, the expression $\{n, n'\} \in$

[5] Space precludes a full description of ESSENCE. See [12] for details. The simple specification given should be clear from the description.

D means that two distinct elements are drawn from D and, without loss of generality, one is called n and the other is called n'. Note also 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*: `integer`, *nnodes*: `integer`, c: `integer`	
`where`	$nrings \geq 0$, $nnodes \geq 0$, $c \geq 0$	
`letting`	N `be integer` $(1..nnodes)$, R `be new type` (`size` *nrings*)	
`given`	D: `set of set` (`size` 2) `of` N	
`find`	*rings-nodes*: $R \times N$	
`minimising`	$\lvert \textit{rings-nodes} \rvert$	(1)
`such that`	$\forall\, r \in R.\ \lvert \textit{rings-nodes}(r, \_)\} \rvert \leq c$	(2)
	$\forall \{n, n'\} \in D.\ \textit{rings-nodes}(\_, n) \cap \textit{rings-nodes}(\_, n') \neq \emptyset$	(3)

Fig. 2. Specification of the simplified SONET problem.

3 Transforming the Abstract Problem Specification

This section presents transformations on the abstract specification of the Simplified SONET problem, S_A. We begin by deriving and adding two implied constraints to the specification.

The first imposes a lower bound on the number of ADMs required for each node. We define the *partner* set of a node n to be $\{n' | \{n, n'\} \in D\}$, each element of which must be related to at least one common ring with n by *rings-nodes*. Once n is installed, the remaining capacity of a ring is $c-1$. Hence, the minimum number of installations of n required to satisfy the communication demand between it and each member of its partner set is $\left\lceil \frac{|\{n' | \{n,n'\} \in D\}|}{c-1} \right\rceil$, to which we refer henceforth as $\textit{ADMMin}_c(n)$. Observe that all the terms contained in $\textit{ADMMin}_c(n)$ are parameters; hence, for any given problem instance, it is constant. The implied constraint follows:

$$\forall n \in N. \textit{ADMMin}_c(n) \leq |\textit{rings-nodes}(\_, n)| \tag{4}$$

The second implied constraint imposes a lower bound on the number of 'open' rings, i.e. those rings with at least one node installed. From (4), it is simple to derive a minimum total number of ADMs. Division by the ring capacity c gives the bound $\left\lceil \frac{\sum_{n \in N} \textit{ADMMin}_c(n)}{c} \right\rceil$, to which we refer henceforth as $\textit{RingMin}_c$. This is also a constant for any given instance. The implied constraint is:

$$\textit{RingMin}_c \leq |\{r \in R | \textit{rings-nodes}(r, \_) \neq \emptyset\}| \tag{5}$$

We now exploit *dominances*. Given an optimisation problem, a partial assignment a dominates another a' if the utility of the best extension of a is at least as good as the best extension of a'. We exploit dominances by adding constraints to preclude dominated partial assignments.

First, an assignment where a node n has more installations than the cardinality k of its partner set is dominated by an assignment where n has at most k installations:

$$\forall n \in N. |\{n' | \{n, n'\} \in D\}| \geq |\textit{rings-nodes}(\_, n)| \tag{6}$$

Henceforth, we refer to $|\{n'|\{n, n'\} \in D\}|$ as $\textit{ADMMax}(n)$. It is also constant for any given instance.

Second, an assignment where a node n is installed on a ring that contains no elements of its partner set is dominated by an assignment where n is only installed on rings containing at least one element of its partner set:

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

Finally, an assignment where the sum of the installations on two non-empty rings is less than or equal to c is dominated by an assignment where the contents of the two rings are merged:

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

4 Refining the Simplified SONET Problem

Refining the transformed Simplified SONET specification principally involves replacing the *rings-nodes* relation variable with a structured collection of atomic variables and set variables. 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. We consider three possibilities for refining an arbitrary relation variable, $R : A \times B$:

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

In the following subsections, we use combinations of these three representations to refine S_A to seven different CSP models, as summarised in Table 1. S_B, S_C and S_D each use one of the three, above-listed representations of a relation variable; the other models each use multiple representations channelled together. Two models, S_B and S_H, closely resemble basic models created by experts in Operations Research [22] and Constraint Programming [23].

Model	Characteristics
S_A	Sets and Relations
S_B	Matrix
S_C	Ring Set Variables
S_D	Node Set Variables
S_E	Matrix + Ring Set Variables
S_F	Matrix + Node Set Variables
S_G	Ring Set + Node Set Variables
S_H	Matrix + Ring Set Variables + Node Set Variables

Table 1. Simplified SONET: Specification and models.

4.1 S_B: A Matrix Model

Using rule (1), the *rings-nodes* relation is refined into a two-dimensional matrix of 0/1 variables, $\textit{rings-nodes}_m$, where $\textit{rings-nodes}_m[r, n]$ denotes the element in column r and row n. The matrix needs to be indexed by N and R. Since N is the set $\{1, \ldots, nnodes\}$ it can serve as an index. However, R is an unnamed set, so it cannot serve as an index. We therefore refine R to the set $\{1, \ldots, nrings\}$.

In the ESSENCE statement of the problem there is no way to refer to particular rings, from which it follows that the rings are constrained identically. By naming the rings in S_B we introduced into the model symmetry among rings. In particular, if an assignment is a solution to a Simplified SONET instance, then it is still a solution after we exchange all the nodes installed on any two rings. For example, if installing nodes $\{1, 2\}$ and $\{3, 4\}$ on rings 1 and 2, respectively, is a solution, then so is installing nodes $\{3, 4\}$ and $\{1, 2\}$ on nodes 1 and 2, respectively. Intuitively, the rings are interchangeable. In S_B the rings are the values of the column index of $\textit{rings-nodes}_m$, so the columns of an assignment can be interchanged without affecting whether the assignment is a solution. This is called column symmetry and, in the general case, index symmetry [9].

This discussion illustrates an important observation: refinement can (and often does) introduce symmetry into the model it generates—and it does so in a *systematic* way that can be characterised formally. Indeed, the formal refinement rules presented by Frisch et. al. [12] identify the symmetries that they introduce. The significance of this is that we can avoid the (potentially expensive) process of trying to detect these symmetries in *each* generated model.

Once symmetries are identified, there are several alternative methods that can be used to break them, and thus reduce solution time. One class of methods, called dynamic symmetry breaking (e.g. SBDS [15]), are the symmetry-aware search methods. These search methods take a description of the symmetries and use it to dynamically prune symmetric parts of the search space. Alternatively, the model can be transformed by adding symmetry-breaking constraints that prune some symmetrical assignments from the search space. This is the approach we take here, the advantage of which will become apparent later.

Column symmetry can be dealt with effectively by treating each column as a vector and constraining the columns to be in non-increasing lexicographic order as the column index increases [9]. Thus, to S_B we add the symmetry breaking constraint:

$$\forall\ 1 \le r < nrings.\ rings\text{-}nodes_m[r, \_] \ge_{\text{lex}} rings\text{-}nodes_m[r+1, \_] \tag{9}$$

where $rings\text{-}nodes_m[r, \_]$ is column r of $rings\text{-}nodes_m$, and $\ge_{\text{lex}}$ denotes lexicographically greater than or equal to, enforceable by the GACLex algorithm [11].

The nodes N in the Simplified SONET problem are not, in general, interchangeable because they have different demands, as specified by D. However, in certain instances some, or all, of the nodes have identical demands. If a set of nodes has identical demands then the corresponding rows are interchangeable. In Figure 1 n_1 and n_2 have identical demands, so the first two rows of an assignment to $rings\text{-}nodes_m$ can be interchanged without affecting whether the assignment is a solution. Given such a set of interchangeable rows, the symmetry can be broken by constraining them to be in non-increasing lexicographic order as the row index increases. This has been shown to be consistent with the lexicographic ordering constraints that we imposed on the columns [9]. If there are multiple sets of interchangeable rows, each such set can be handled in this way.

The previous paragraph shows how certain symmetries in a model of a particular instance can be handled by adding symmetry-breaking constraints to the model. Our main focus is on building models of problems not instances, so we do not discuss this in detail. However, to handle instance-specific symmetries in a model of a problem, preconditions must be placed on the methods used to break symmetry. As a problem is instantiated into an instance, the preconditions are tested and symmetry is broken among the objects that are symmetrical in the instance.

Now that we have discussed the refinement of *rings-nodes* to a matrix, and the symmetries involved, we continue by refining the constraints and objective function. This requires replacing constraints on *rings-nodes* with constraints on $rings\text{-}nodes_m$. Each column r of $rings\text{-}nodes_m$ corresponds to the characteristic function for the set of nodes installed on ring r (i.e. $rings\text{-}nodes(r, \_)$), and similarly for each row n, so refining (1) and (2) is straightforward:

$$\text{Minimise}(\sum_{r \in R} \sum_{n \in N} rings\text{-}nodes_m[r, n]) \tag{10}$$

$$\forall r \in R.\ \sum_{n \in N} rings\text{-}nodes_m[r, n] \le c \tag{11}$$

The demand constraint requires 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:

$$\forall \{n, n'\} \in D.\text{scalar-product}(rings\text{-}nodes_m[\_, n], rings\text{-}nodes_m[\_, n']) \ne 0 \tag{12}$$

where $rings\text{-}nodes_m[\_, n]$ denotes the nth row of the $rings\text{-}nodes_m$ matrix. S_B is a basic version of that used in [22]. Indeed, matrix models in general are a common pattern in constraint programming [8].

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

$$\forall n \in N. ADMMin_c(n) \leq \sum_{r \in R} \textit{rings-nodes}_m[r, n] \tag{13}$$

$$\forall\, n \in N, r \in R. \textit{rings-nodes}_m[r, n] = 1 \rightarrow \sum_{n' | \{n, n'\} \in D} \textit{rings-nodes}_m[r, n'] > 0 \tag{14}$$

Constraint (5) places a lower bound on the number of open rings. A ring, r is open if it has at least one ADM installed on it, which in this model means that column r of *rings-nodes*$_m$ has a non-zero sum. So, constraint (5) could be implemented by introducing a 0/1 variable for each ring to indicate if it is open. This is cumbersome to impose and it is a weak constraint because it does not force any particular ring to be open.

A much better way of dealing with constraint (5) is obtained by noticing that symmetry-breaking constraint (9) implies that all the open rings are less than[6] the unopen rings. Thus we can impose the constraint that each of the first $RingMin_c$ columns of $ringnodes_m$ has a non-zero sum:

$$\forall\, 1 \leq r \leq RingMin_c. \sum_{n \in N} ringnodes_m[r, n] \neq 0 \tag{15}$$

Observe that this constraint, which is much stronger than merely saying that at least $RingMin_c$ rings are open, can be imposed only because of the symmetry-breaking constraint. In general, the choice between alternative symmetry-breaking constraints should consider the inferred constraints they enable [13]. Also note that this is often a significant advantage to using symmetry-breaking constraints over dynamic symmetry-breaking methods.

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

$$\begin{aligned} &\forall \{r, r'\} \subseteq R. \\ &\quad \textstyle\sum_{n \in N} \textit{rings-nodes}_m[r, n] > 0 \wedge \sum_{n \in N} \textit{rings-nodes}_m[r', n] > 0 \\ &\quad \rightarrow \textstyle\sum_{n \in N} (\textit{rings-nodes}_m[r, n] + \textit{rings-nodes}_m[r', n]) > c \end{aligned} \tag{16}$$

4.2 S_C: A Set Variable (Rings) Model

Using rule (2), the *rings-nodes* relation is refined into a one-dimensional matrix of set variables, *nodesOnRing*$_{ms}$, indexed by R such that *nodesOnRing*$_{ms}[r]$ contains the set of nodes installed on r. As in the previous sub-section, to serve as an index R is refined to the set $\{1, .., nrings\}$.

The objective and ring capacity constraint are easily stated:

$$\text{Minimise}\Big(\sum_{r \in R} |nodesOnRing_{ms}[r]|\Big) \tag{17}$$

$$\forall r \in R. |nodesOnRing_{ms}[r]| \leq c \tag{18}$$

[6] Since each ring is identified by an integer, some rings are "less than" others

The demand constraint is more difficult to specify. It constrains at least one of the set variables to contain particular pairs of nodes:

$$\forall \{n, n'\} \in D. \sum_{r \in R} (n \in nodesOnRing_{ms}[r] \wedge n' \in nodesOnRing_{ms}[r]) > 0 \quad (19)$$

In the above we have *reified* each conjunction to a 0/1 value and used summation to express the disjunction.

The symmetry among the indices of $nodesOnRing_{ms}$ can be broken cheaply (but only partially) by ordering the cardinalities of the sets:

$$\forall 1 \leq r < nrings.\ |nodesOnRing_{ms}[r]| \geq |nodesOnRing_{ms}[r+1]| \quad (20)$$

Having broken the symmetry in this way, the implied constraint on the minimum number of open rings can be refined simply by disallowing the first $RingMin_c$ elements of $nodesOnRing_{ms}$ from being empty:

$$\forall 1 \leq r \leq RingMin_c. |nodesOnRing[r]| \neq 0 \quad (21)$$

The remaining implied constraint (4) on the minimum number of installations for any node is more awkward since it requires that we check each of the individual rings. Again we use a summation of reified element constraints:

$$\forall n \in N. \sum_{r \in R} n \in nodesOnRing_{ms}[r] \geq ADMMin_c(n) \quad (22)$$

The *ADMMax* constraint to exploit dominance (6) can be stated similarly.

Of the remaining constraints to exploit dominance, $nodesOnRing_{ms}$ facilitates the expression of the content merging constraint (8) most easily:

$$\forall \{r, r'\} \subseteq R. |nodesOnRing_{ms}[r]| > 0 \wedge |nodesOnRing_{ms}[r']| > 0 \rightarrow$$
$$|nodesOnRing_{ms}[r]| + |nodesOnRing_{ms}[r']| > c \quad (23)$$

Finally, we refine the constraint that specifies a node should only be installed on a ring that contains at least one element of its partner set (7):

$$\forall n \in N, r \in R. n \in nodesOnRing[r] \rightarrow \sum_{n' | \{n, n'\} \in D} (n' \in nodesOnRing[r]) > 0 \quad (24)$$

4.3 S_D: A Set Variable (Nodes) Model

Using rule (3), the *rings-nodes* relation is refined into a one-dimensional matrix of set variables $ringsWithNodes_{ms}$, indexed by N such that $ringsWithNodes_{ms}[n]$ contains the set of rings on which n is installed. Since R is an unnamed set it cannot provide the domain elements for the set variables. Once again, therefore, it is refined to the set $\{1, .., nrings\}$.

Given $ringsWithNode_{ms}$, the objective is refined as follows:

$$\text{Minimise}(\sum_{n \in N} |ringsWithNode_{ms}[n]|) \quad (25)$$

The demand constraints are also easily stated:

$$\forall \{n, n'\} \in D. |\mathit{ringsWithNode}_{ms}[n] \cap \mathit{ringsWithNode}_{ms}[n']| \geq 1 \tag{26}$$

However, since the ring capacity constraints involve all the node constraints we again make use of reification:

$$\forall r \in R. \sum_{n \in N} r \in \mathit{ringsWithNode}_{ms}[n] > 0 \tag{27}$$

The symmetry among the values of $\mathit{ringsWithNode}_{ms}$ can be broken partially by insisting that the first node, n, with a non-empty partner set is installed on the first ring:

$$1 \in \mathit{ringsWithNode}_{ms}[n] \tag{28}$$

Hence, the implied constraint on the minimum number of open rings (5) is refined by ensuring that the first $\mathit{RingMin}_c$ rings appear in at least one of the sets in $\mathit{ringsWithNode}_{ms}$:

$$\forall 1 \leq r \leq RingMin_c. \sum_{n \in N} r \in \mathit{ringsWithNode}[n] \neq 0 \tag{29}$$

The implied constraint on the minimum number of installations per node (4) is easily stated on the node set variables:

$$\forall n \in N. \mathit{ADMMin}_c(n) \leq \sum_{n \in N} |\mathit{ringsWithNode}_{ms}[n]| \tag{30}$$

Again, the ADMMax constraint (6) is specified similarly.

The remaining constraints to exploit dominances, i.e. that a node should only be installed on a ring that contains at least one element of its partner set (7) and the content merging constraint (8), are refined as follows:

$$\forall n \in N, r \in R. r \in \mathit{ringsWithNode}[n] \rightarrow \sum_{n' | \{n,n'\} \in D} (r \in \mathit{ringsWithNode}[n']) > 0 \tag{31}$$

$$\forall \{r, r'\} \subseteq R. (\sum_{n \in N} r \in \mathit{ringsWithNode}[n] > 0) \wedge (\sum_{n \in N} r' \in \mathit{ringsWithNode}[n] > 0)$$
$$\rightarrow \sum_{n \in N} (r \in \mathit{ringsWithNode}[n] + r' \in \mathit{ringsWithNode}[n]) > c \tag{32}$$

4.4 S_E: A Matrix and Set Variable (Rings) Model

To maintain consistency between $\mathit{rings\text{-}nodes}_m$ and $\mathit{nodesOnRing}_{ms}$, the following channelling constraint is used:

$$\forall r \in R.\ n \in \mathit{nodesOnRing}_{ms}[r] \leftrightarrow \mathit{rings\text{-}nodes}_m[r, n] = 1 \tag{33}$$

The objective can be stated on either $\mathit{rings\text{-}nodes}_m$ (10) or $\mathit{nodesOnRing}_{ms}$ (17). We will explore both alternatives. Symmetry breaking is performed on *rings-nodes*$_m$ as in model S_B (9), since this scheme breaks all symmetry whereas the ordering on the cardinalities used in model S_C (20) does not. Also from model S_B we take the demand constraint (12), the *ADMMin* (13) and *ADMMax* constraints, and the constraint that specifies that a node should only be installed on a ring with at least one of its partners (14). From model S_C we take the ring capacity constraints (18), the constraint on the minimum number of open rings (21), and the content merging constraint (23).

4.5 S_F: A Matrix and Set Variable (Nodes) Model

To maintain consistency between *rings-nodes*$_m$ and *ringsWithNode*$_{ms}$, the following channelling constraint is used:

$$\forall n \in N.\ r \in \mathit{ringsWithNode}_{ms}[n] \leftrightarrow \mathit{rings\text{-}nodes}_m[r,n] = 1 \tag{34}$$

The objective can be stated on either *rings-nodes*$_m$ (10) or *ringsWithNode*$_{ms}$ (25). Again, we will explore both alternatives. As in models, S_B and S_E, we perform complete symmetry breaking via *rings-nodes*$_m$ (9). From model S_B we take the ring capacity constraint (11), the constraint on the minimum number of open rings (15), the constraint that a node should only be installed on a ring containing at least one of its partners (14), and the content merging constraint (16). From model S_D we take the demand constraint (26), and the *ADMMin* (30) and *ADMMax* constraints.

4.6 S_G: A Dual Set Variable Model

To maintain consistency between *nodesOnRing*$_{ms}$ and *ringsWithNode*$_{ms}$, the following channelling constraint is used:

$$\forall n \in N, r \in R. n \in \mathit{nodesOnRing}_{ms}[r] \leftrightarrow r \in \mathit{ringsWithNode}_{ms}[n] \tag{35}$$

The objective can be stated on either *nodesOnRing*$_{ms}$ (17) or *ringsWithNode*$_{ms}$ (25). Again, we will explore both alternatives. Symmetry breaking is performed on *nodesOnRing*$_{ms}$, as in model S_C (20). Also from model S_C we take the ring capacity constraint (18), the constraint on the minimum number of open rings (21), the constraint that specifies that a node should only be installed on a ring with at least one of its partners (24), and the content merging constraint (23). From model S_D we take the demand constraint (26), and the *ADMMin* (30) and *ADMMax* constraints.

4.7 S_H: A Matrix and Set Variable (both Rings and Nodes) Model

Although only two channelling constraints are sufficient to maintain consistency among the 0/1 matrix and the two matrices of set variables, we use the three channelling constraints from models S_E (33), S_F (34) and S_G (35). The objective can be stated easily on any of the three models. We will explore all three alternatives. Symmetry is broken completely on *rings-nodes*$_{ms}$ as described in model

S_B (9). Also from model S_B we take the constraint that a node should only be installed on a ring containing at least one of its partners (24). From model S_C we take the ring capacity constraint (18), the minimum number of open rings (21), and the content merging constraint (23). From model S_D we take the demand constraint (26), and the *ADMMin* (30) and *ADMMax* constraints.

5 Model Selection

As we have shown in introducing each of the models S_B to S_H, constraints may be more or less difficult to express, depending on the variables included in the model. However, which model is best in solving time, given some standard constraint solver, is hard to determine. 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 [18]. In many cases, however, empirical tests are needed to develop guidelines for making informed model choices. In this section, we contribute to this goal of pattern elicitation by performing an empirical analysis of our models of the Simplified SONET problem. Despite the small scale of this study, the trends are strong and immediately apparent.

There are a number of issues to consider in designing our experiment. First, introducing new variables can introduce a choice of search variables. For instance, in model S_E, we can search either on the matrix variables *rings-nodes*$_m$ or on the ring set variables *nodesOnRing*$_{ms}$. Second, having chosen the search variables, we need to decide the order in which to assign the variables (either statically or dynamically). It is well known that the choice of variable ordering can dramatically affect the search effort required to solve a CSP. However, we can only compare the performance of the models presented to a limited extent. For instance, we could compare models S_B, S_E and S_F using the matrix variables as search variables and the same variable ordering in each case. This would show whether being able to express some of the constraints more easily using set variables has any effect on performance. The results would not, however, necessarily reflect the best known performance for these models, still less what the best performance for each model might be with the ideal ordering heuristic.

Each of our models is described by a triple $\langle BasicModel, BranchingStrategy,$ $ObjectiveExpression\rangle$. $BasicModel \in \{B, C, D, E, F, G, H\}$ corresponds to the models S_B - S_H. We considered $BranchingStrategy \in \{M, N, R\}$ where M stands for using the matrix variables as search variables, N for using the node set variables, and R for the using the ring set variables. Finally, we considered $ObjectiveExpression \in \{M, N, R\}$, where M stands for expressing the objective function using the matrix variables (10), N stands for using the node set variables (17), and R for the ring set variables (25). We tested the 24 consistent combinations of these three choices on 10 instances (from [23]) using a 750Mhz 128Mb Pentium III and Ilog Solver 5.3 (Windows version).

Model	s2ring1	s2ring2	s2ring3	s2ring4	s2ring5	s2ring6	s2ring7	s2ring8	s2ring9	s2ring10
$\langle B, M, M\rangle$	25411	27063	20032	7938	24097	9625	8460	10001	41849	9428
$\langle C, R, R\rangle$	>514K	>514K	>468K	>492K	>518K	>495K	441K	>512K	>506K	>513K
$\langle D, N, N\rangle$	12744	374983	63892	16771	48955	400641	25311	78181	239662	15680
$\langle E, M, R\rangle$	7971	5421	4765	1583	8601	2491	1394	2597	15912	3761
$\langle E, R, R\rangle$	68395	199656	87765	36343	134385	140771	15301	103778	225852	41991
$\langle F, M, N\rangle$	**112**	**165**	**73**	**21**	**356**	**193**	**39**	**188**	**1136**	**70**
$\langle F, N, N\rangle$	1407	17303	3804	2525	2758	27920	2702	6218	4329	1387
$\langle G, R, N\rangle$	378K	>675K	>636K	>634K	>663K	>614K	256K	>688K	>668K	>461K
$\langle G, N, N\rangle$	20629	>546K	67994	56995	98936	>642K	51673	78982	229565	41031
$\langle H, M, N\rangle$	**112**	**165**	**73**	**21**	**356**	**193**	**39**	**188**	**1136**	**70**
$\langle H, R, N\rangle$	12876	217440	23569	20472	84782	87275	10559	145915	52622	7572
$\langle H, N, N\rangle$	1407	17303	3804	2525	2758	27920	2702	6218	4329	1387

Table 2. Experimental results on 10 instances of Simplified SONET (choices).

Model	s2ring1	s2ring2	s2ring3	s2ring4	s2ring5	s2ring6	s2ring7	s2ring8	s2ring9	s2ring10
$\langle B, M, M\rangle$	2.49	2.78	2.29	0.86	2.41	1.03	0.97	1.08	4.2	1.05
$\langle C, R, R\rangle$	>160	>160	>160	>160	>160	>160	132.44	>160	>160	>160
$\langle D, N, N\rangle$	6.7	158.37	32.44	7.64	24.32	160.02	13.5	33.26	115.96	8
$\langle E, M, R\rangle$	1.22	0.88	0.79	0.26	1.44	0.42	0.25	0.47	2.7	0.65
$\langle E, R, R\rangle$	14.76	39.43	18.74	7.47	27.05	29.11	3.22	18.95	44.76	9.47
$\langle F, M, N\rangle$	**0.03**	**0.04**	**0.03**	**0.02**	**0.08**	**0.04**	**0.02**	**0.04**	**0.19**	**0.03**
$\langle F, N, N\rangle$	0.25	2.29	0.6	0.39	0.43	3.45	0.44	0.82	0.71	0.24
$\langle G, R, N\rangle$	92.13	>160	>160	>160	>160	>160	60.95	>160	>160	>160
$\langle G, N, N\rangle$	6.47	>160	22.37	15.86	24.34	>160	15.3	18.03	79.66	11.75
$\langle H, M, N\rangle$	0.05	0.07	0.05	0.03	0.13	0.07	0.03	0.07	0.38	0.04
$\langle H, R, N\rangle$	3.06	46.86	5.81	4.29	16.43	18.6	2.33	27.54	11.87	1.94
$\langle H, N, N\rangle$	0.47	4.32	1.23	0.75	0.73	6.2	0.87	1.45	1.37	0.41

Table 3. Experimental results on 10 instances of Simplified SONET (time).

Tables 2 and 3 present the number of choice points and time taken within a 160 seconds time limit. For brevity, given a subset of models that are identical apart from the objective function, we only show the results for the model with the most effective expression of the objective. For instance, we do not show the results for model $\langle E, M, M\rangle$, since $\langle E, M, R\rangle$ is consistently more effective.

This filter immediately reveals a general observation: in these tests it was always more effective to express the objective via the cardinality of the set variables. This is because of an interaction with the *RingMin* constraint (5) in the case of the ring set variables, and with the *ADMMin* constraint (4) in the case of the node set variables. Consider *rings-nodes*$_m$ for a small instance:

$$\begin{array}{c} \\ node_1 \\ node_2 \\ node_3 \end{array} \begin{array}{c} \begin{array}{ccc} ring_1 & ring_2 & ring_3 \end{array} \\ \begin{pmatrix} 0/1 & 0/1 & 0/1 \\ 0/1 & 0/1 & 0/1 \\ 0/1 & 0/1 & 0/1 \end{pmatrix} \end{array}$$

Assume that all nodes need to be installed at least once, and that all three rings must be open. These constraints can be imposed as sums on the rows and columns of *rings-nodes*$_m$. Propagating these constraints results in no domain pruning initially. Consider the search for a solution with less than 3 installations.

Since all elements of $rings\text{-}nodes_m$ can still be set to 0, search is necessary to determine that this is not possible.

Consider now expressing the *RingMin* constraint and the objective on the $nodesOnRing_{ms}$ matrix. The lower bound on the cardinality of each ring is one. Hence, the sum of the cardinalities is at least 3 and the search fails immediately. Expressing the *ADMMin* constraint and the objective on $ringsWithNode_{ms}$ gives a similar result. The key observation is that the bounds directly tighten the domain of a variable (the hidden cardinality variable associated with each set variable). Since these same variables are used to express the objective, any tightening of the bounds has a direct effect on the bound on the objective. This is not the case for the sum constraints on $rings\text{-}nodes_m$. Since the *ADMMin* constraint gives a tighter bound on the cardinality variables than *RingMin*, this also explains why expressing the objective on $ringsWithNode_{ms}$ is the most effective choice in these experiments.

A second observation is that it is most effective to branch on $rings\text{-}nodes_m$. This is probably due to the fact that assigning a single 0/1 variable is less of a commitment than assigning a whole set at once. Hence, the culprit at a dead end is more readily apparent. These two observations together explain the performance of $\langle F, M, N\rangle$ as the best model. Model $\langle H, M, N\rangle$ explores the same search tree, but incurs an overhead for maintaining $nodesOnRing_{ms}$.

6 Related Work in Modelling and Transformation

Several recent efforts focus on automating refinement. Hnich [17] shows how to automatically refine specifications in $\mathcal{F}$ and Frisch et. al. [12] show how to refine specifications in ESSENCE. Both of these refinement systems generate a set of alternative models, including models with multiple, channelled representations, but neither provides a mechanism for choosing among the alternatives. The initial implementation of Relational ESRA, which is under development, will refine specifications to a single constraint model in which relation variables are always refined to 0/1 matrices (as in S_B) [10].

Our work is also motivated by experience with the CGRASS (Constraint Generation And Symmetry-breaking [14]) system. CGRASS automatically transforms constraint models of problem *instances* in order to make them easier to solve. However, since CGRASS transforms individual instances, much effort is repeated if one wants to solve multiple instances of a problem. Furthermore, the instance specifications that CGRASS transforms are non-schematic; instead of using universal quantifiers to implicitly state a set of constraints, the set is explicitly stated. For some problem instances this results in very large specifications, which, in turn, require many applications of the transforation rules. This is why we focus on schematic problem specifications in this paper.

There are several other methods to aid in constraint modelling, which we briefly survey. Laurière [19] introduced a modelling language called ALICE to formally state a problem. The language is characterised by the use of sets, set operators, Cartesian product of sets, vectors, matrices, graphs and paths, constants, and functions. The language NP-SPEC is a logic-based executable specification language [7, 6], which allows the user to specify problems by using metapredicates (*subset*, *partition*, *permutation*, and *intfunc*). REFINE is a functional language for specifying global search problems for a program synthesizer [24]. The REFINE language augments a functional programming language with three type constructors, namely *set*, *sequence*, and *map*, as well as their operations.

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. The computer-aided constraint programming project [4, 21] aimed at building a system that encapsulates the entire process of applying CP technology to problems.

7 Conclusions

We have considered the transformation of constraint satisfaction problems and shown that we can and should transform problems at various levels of abstraction. Refinement is a process of progressively moving to more concrete models; mechanisms for dealing with some common modelling problems, such as symmetry, can be embedded into refinement rules.

The Simplified SONET problem illustrates how integrating transformation and refinement could work in general: an abstract problem specification in the ESSENCE language was transformed by adding implied and other constraints. The result was refined into seven alternative models. In addition, we showed how the refinement process could trigger further useful transformations: in this case, breaking the symmetries that it introduces into the model.

Our immediate goal is to formalise fully the transformations we use. Furthermore, we wish to combine theoretical analysis with the lessons learnt from empirical analyses, like the one performed on the Simplified SONET problem herein, to evaluate models statically, and therefore be more selective about the models produced during refinement.

Acknowledgements We thank Sherali and Smith for providing sample instances. Brahim Hnich is supported by Science Foundation Ireland. Ian Miguel is supported by a UK-Royal Academy of Engineering/EPSRC Research Fellowship. This work was done while the fourth author was employed at the University of Huddersfield.

References

1. A. Abbas and E.P.K. Tsang. Toward a general language for the specification of constraint satisfaction problems. *Proc. Constraint Programming, Artificial Intelligence and Operations Research (CP-AI-OR) Wshop*, 2001.
2. J.E. Borrett. *Formulation selection for constraint satisfaction problem: a heuristic approach*. PhD Thesis, Dept. of Computer Science, University of Essex, 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. *Proc. Wshop on Analysis and Visualization of Constraint Programs & Solvers*, 2000.

5. A. Bundy. A Science of Reasoning. J-L. Lassez and G, Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, 178–198, 1991.
6. M. Cadoli and A. Schaerf. Compiling program specifications into SAT. *Proc. 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. *Proc. PADL'99*, pp. 16–30. LNCS 1551. Springer-Verlag, 1999.
8. P. Flener, A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, T. Walsh. Matrix Modelling: Exploiting Common Patterns in Constraint Programming *Proc. Int. Wshop on Reformulating CSPs*, 2002.
9. 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 Int. Conf. on Principles & Practice of CP* LNCS 2470, 2002.
10. P. Flener, J. Pearson, and M. Agren. Introducing ESRA, a relational language for modelling combinatorial problems. *LOPSTR'03: Revised Selected Papers*, LNCS 3018, 2004.
11. A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, T. Walsh. Global Constraints for Lexicographic Orderings. *Proc. 8th Int. Conf. on Principles & Practice of CP* LNCS 2470, 2002.
12. A.M. Frisch, C. Jefferson, B. Martínez-Hernández, I. Miguel. *The Rules of Constraint Modelling*. *Proc. 19th Int. Joint Conf. on AI*, 2005.
13. A.M. Frisch, C. Jefferson, I. Miguel. *Symmetry Breaking as a Prelude to Implied Constraints: A Constraint Modelling Pattern*. *Proc. 16th Euro. Conf. on AI*, 171-175, 2004.
14. A.M. Frisch, I. Miguel, and T. Walsh. CGRASS: A System for Transforming Constraint Satisfaction Problems. *Proc. Joint Wshop of the ERCIM/CologNet Area on CP & CLP*, LNCS 2627, 23–26, 2002.
15. I. P. Gent and B. M. Smith. Symmetry Breaking During Search in Constraint Programming. *Proc. European Conf. on AI*, 599–603, 2000.
16. F. Giunchiglia, T. Walsh. A Theory of Abstraction. *Artificial Intelligence* 56(2–3), 323-390, 1992
17. B. Hnich. *Function Variables for Constraint Programming*. PhD Thesis, University of Uppsala, 2003.
18. B. Hnich, B.M. Smith and T. Walsh. Models of Permutation and Injection Problems. *Journal of Artificial Intelligence Research*, 21, 2004.
19. J-L. Lauriere. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10(1):29-127, 1978.
20. A.K. Mackworth. Constraint Satisfaction Problems. *Encyclopedia of AI*, 285–293, 1992.
21. P. Mills, E.P.K. Tsang, R. Williams, J. Ford, and J. Borrett. EaCL 1.5: An Easy abstract Constraint optimisation Programming Language, TR CSM-324, University of Essex, 1999.
22. H.D. Sherali and J.C. Smith. Improving Discrete Model Representations via Symmetry Considerations. *Management Science* 47, 1396–1407, 2001.
23. B. M. Smith Symmetry and Search in a Network Design Problem. *Proc. 2nd Int. Conf. on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, LNCS 3524, 336-350, 2005.
24. D.R. Smith. The structure and design of global search algorithms. *TR KES.U.87.12*, Kestrel Institute, 1988.
25. B. M. Smith, K. Stergiou, and T. Walsh. Modelling the Golomb Ruler Problem. *Proc. IJCAI-99 Wshop on Non-Binary Constraints*. Int. Joint Conf. on AI, 1999.