Weighted Constraint Satisfaction with Set Variables

J.H.M. Lee and **C.F.K. Siu**
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
{jlee,fksiu}@cse.cuhk.edu.hk

Abstract

Set variables are ubiquitous in modeling (soft) constraint problems, but efforts on practical consistency algorithms for Weighted Constraint Satisfaction Problems (WCSPs) have only been on integer variables. We adapt the classical notion of set bounds consistency for WCSPs, and propose efficient representation schemes for set variables and common unary, binary, and ternary set constraints, as well as cardinality constraints. Instead of reasoning consistency on an entire set variable directly, we propose local consistency check at the set element level, and demonstrate that this apparent "micro"-management of consistency does imply set bounds consistency at the variable level. In addition, we prove that our framework captures classical CSPs with set variables, and degenerates to the classical case when the weights in the problem contain only 0 and $\top$. Last but not least, we verify the feasibility and efficiency of our proposal with a prototype implementation, the efficiency of which is competitive against ILOG Solver on classical problems and orders of magnitude better than WCSP models using 0-1 variables to simulate set variables on soft problems.

Introduction

Many constraint problems can be modeled *naturally* using set variables. This is no exception with weighted constraint satisfaction problems (WCSPs). A set variable with n possible set elements has a domain of size 2^n. Domain consistency techniques (Larrosa 2002) developed for integer variables cannot be practically adapted for set variables since these techniques require all elements of a variable domain to be represented explicitly. Following Gervet (1997), we propose efficient set bounds consistency techniques for set variables which reason only on the bounds of the variables.

Constraints in WCSPs are cost functions, mapping tuples to costs. Instead of specifying the cost functions at the tuples (of set values) level, we devise a general scheme for representing tuple costs according to costs associated with the existence and inexistence of elements in the set values. This scheme allows us to specify cost functions to all common set operations and constraints, and degenerates to classical CSPs with set variables when all costs are either 0 or $\top$. Node, arc, hyper-arc, and cardinality consistency notions and the associated enforcement algorithms are defined for unary, binary, ternary, and cardinality constraints at the set element level respectively. We show that these element consistencies imply set bounds consistency (Gervet 1997; Müller & Müller 1997; Hawkins, Lagoon, & Stuckey 2005) generalized for WCSPs. We construct a prototype implementation of our algorithms by modifying the ToolBar WCSP solver (Bouveret *et al.* 2004). Experiments are conducted to compare our implementation against ILOG Solver (ILOG 2003) on classical set CSPs, and against 0-1 variable emulation of set variables in ToolBar on softened versions of the same classical benchmarks. Results confirm that our implementation is more efficient than ILOG Solver on classical problems and two orders of magnitude better than WCSP models using 0-1 variable to simulate set variables on soft problems.

Background

A classical constraint satisfaction problem (CSP) is a tuple $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ where $\mathcal{X}$ is a finite set of *variables*, $\mathcal{D}$ is a finite set of *variable domains*, and $\mathcal{C}$ is a finite set of *constraints*. A variable $x_i \in \mathcal{X}$ can only be assigned with a value from its variable domain $D(x_i) \in \mathcal{D}$. Each constraint $C_{i_1,\ldots,i_n} \in \mathcal{C}$ restricts the values taken by the variables $x_{i_1}, \ldots, x_{i_n}$ simultaneously.

Gervet (1997) proposed a framework to handle set variables in classical CSP. The framework allows the specification of common set constraints involving set operations: union ($\cup$), intersection ($\cap$), difference ($\setminus$), and set relations: subset ($\subseteq$) and equality ($=$).

The domain of a set variable S is approximated by a set interval. The set interval is specified by its upper bound and lower bound. The upper bound contains all the set elements that may be in the set value and the lower bound contains all the set elements that must exist in the set value. The upper bound and the lower bound are also called the *possible set* $PS(S)$ and the *required set* $RS(S)$ respectively, which are ordered by set inclusion: $RS(S) \subseteq PS(S)$. The set interval $[RS(S), PS(S)]$ represents the set $\{A | RS(S) \subseteq A \subseteq PS(S)\}$. For example, the set interval $[\{a, b\}, \{a, b, c, d\}]$ represents the set: $\{\{a, b\}, \{a, b, c\}, \{a, b, d\}, \{a, b, c, d\}\}$. Constraint propagation is done on the set intervals by enforcing *set bounds consistency* (Gervet 1997; Müller & Müller

1997; Hawkins, Lagoon, & Stuckey 2005).

Definition 1. *A set variable S with set interval domain $[RS(S), PS(S)]$ is* set bounds consistent *w.r.t. a constraint C if and only if $RS(S) = \bigcap\{v\} \wedge PS(S) = \bigcup\{v\}, \forall v \in [RS(S), PS(S)] \wedge v$ satisfies C.*

A weighted constraint satisfaction problem (WCSP) extends the CSP framework by associating costs to the tuples of variable assignments. Each cost indicates the degree of preference for the tuple. Costs are specified by the *valuation structure* $S = (E, \oplus, \succeq)$. E is the set of costs which is totally ordered by $\succeq$. The costs are combined by the operator $\oplus$. The maximum and minimum costs are denoted as $\top$ and $\bot$ respectively.

A *Weighted Constraint Satisfaction Problem (WCSP)* is a tuple $\mathcal{P} = (k, \mathcal{X}, \mathcal{D}, \mathcal{C})$. $\mathcal{X}$ and $\mathcal{D}$ are the set of variables and set of domains respectively. $\mathcal{C}$ is a set of cost functions. The valuation structure is $S(k)$ which denotes a triple $([0, 1, \ldots, k], \oplus, \geq)$, where $k \in [1, \ldots, \infty]$. The operator $\oplus$ is defined as $a \oplus b = min\{k, a + b\}$. The set is totally ordered by the standard order $\geq$. There is a zero-arity constraint which implies the global lower bound of the problem.

The cost $\mathcal{V}(t)$ of a tuple of variable assignment t can be obtained by combining the costs for all constraints.

$$\mathcal{V}(t) = \sum_{\substack{C_{i_1 \ldots i_n} \in \mathcal{C}, \\ \{x_{i_1}, \ldots, x_{i_n}\} \subseteq var(t)}} C_{i_1 \ldots i_n}(t \downarrow_{\{x_{i_1}, \ldots, x_{i_n}\}}) \oplus C_\emptyset$$

The projection notation $t \downarrow_{\{x_{i_1}, \ldots, x_{i_n}\}}$ means projecting t on $\{x_{i_1}, \ldots, x_{i_n}\}$. If $\mathcal{V}(t) < \top$, the tuple t is said to be *consistent*. There are some consistency notions and consistency enforcing algorithms defined in WCSPs to reduce the search space by local information. These includes the star node consistency and arc consistency defined by Larrosa (2002). In solving a WCSP, we are finding a complete consistent assignment with the minimum cost.

WCSPs with Set Variables

In WCSPs, if we model a set variable with the domain containing all the possible set values, the time and space complexity makes solution searching impractical. We observe that common set constraints only consider the existence state of each particular set element individually. For example, the subset constraint $S_i \subseteq S_j$ restricts each possible element a among the set variables S_i and S_j, such that $a \in S_i \rightarrow a \in S_j$. As a result, we can refine set constraints to consider the existence state of each set element for the satisfiability in classical CSPs and also for the cost in WCSPs.

Soft Set Variables

A set variable S is associated with a domain of integer sets and a universal set $\mathcal{U}$ where $\mathcal{U} = \bigcup D_0(S)$ and $D_0(S)$ is the initial domain of variable S. The domain of S is presented by the cost of its existence and inexistence of each set element. When we assign $\top$ to the existence (resp. inexistence) of a set element, we prohibit the set variable to contain (resp. remove) the set element. When the cost is less than $\top$, we allow the existence (resp. inexistence) of the set element. The domain of a set variable is also its unary constraint.

Soft Set Constraints

Set constraints defined here consider the existence state of each set element. This nature allows us to express the common soft set constraints which include element membership ($a \in S_i$, $a \notin S_i$), equality ($S_i = S_j$), subset ($S_i \subseteq S_j$), union ($S_i \cup S_j = S_k$), intersection ($S_i \cap S_j = S_k$), difference[1] ($S_i \backslash S_j = S_k$), and cardinality ($|S_i| = n$, $|S_i| \leq n$, $|S_i| \geq n$) where n is a constant.

In this paper, we focus on unary, binary, ternary, and cardinality constraints. These constraints enable us to express the common set constraints listed above with set variables. Since the performance of constraint propagation will degrade when the arity of a constraint is high, such kind of constraints is usually decomposed to some primitive low arity constraints by introducing auxiliary variables (Cleary 1987; Gervet 1997). For example, the constraint $S_1 \cap S_2 \subseteq S_3 \cup S_4$ can be decomposed to $S_1 \cap S_2 = A_1$, $S_3 \cup S_4 = A_2$ and $A_1 \subseteq A_2$ with the introduction of two auxiliary set variables A_1 and A_2. However, our definitions and algorithms do not restrict the arity of the constraints theoretically.

Definition 2. *An* existence state *of a set element a w.r.t. a set value u is the boolean value of $a \in u$ with possible values true (t) and false (f).*

Since the cost of a constraint is determined by the existence states of the set elements, we decompose the cost of the constraint by the corresponding *element cost functions* to reflect this relation. An element cost function maps the existence states $\{t, f\}$ of a set element to an *element cost*. The cost for the constraint is the sum of all the element costs given from the set of refined element cost functions for that constraint. This approach gives compact representation of constraints. Figure 1 shows an example of $2 \in S_1 \wedge 3 \notin S_2$ and $S_1 \subseteq S_2$. A dotted rectangle represent a set variable. Each oval in the rectangle is associated with a set element. The two circles in the oval represent the existence states of the set element and contains the corresponding unary costs. The binary costs between two set variables are indicated on the lines representing constraints.

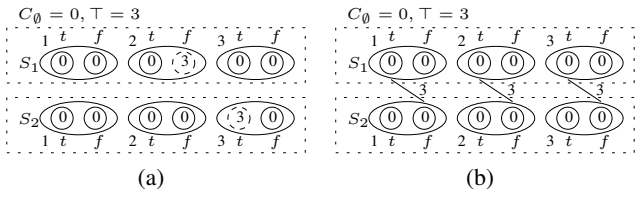

Figure 1: (a) $2 \in S_1 \wedge 3 \notin S_2$ (b) $S_1 \subseteq S_2$

The Zero-arity Constraint As in WCSPs, there is a zero-arity constraint $C_\emptyset$ in the problem. The cost of the zero-arity constraint can be interpreted as the global lower bound of the problem. The problem contains no solutions when $C_\emptyset = \top$.

Unary Constraints A unary constraint C_i assigns costs to assignments to variable S_i ($C_i : D(S_i) \rightarrow [0, \ldots, k]$).

[1]Complementation can be implemented using difference.

The corresponding *unary element cost function*, which assigns costs for the existence (t) and inexistence (f) for each set element $a \in \mathcal{U}_i$ w.r.t. set variable S_i, is $\varphi_{(i)/a} : \{t, f\} \rightarrow [0, \ldots, k]$. The unary cost is decomposed as $C_i(u) = \sum_{a \in \mathcal{U}_i} \varphi_{(i)/a}(a \in u)$.

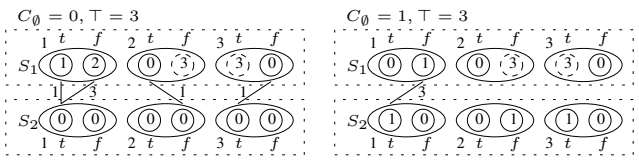

Figure 2: (a) An example WCSP (b) ENC and EAC

In the WCSP in Figure 2(a), for example, the cost of $2 \in S_1$ is 0 while that of $2 \notin S_1$ is 3:

$$\varphi_{(1)/2}(\alpha) = \begin{cases} 0 & \text{if } \alpha = t; \\ 3 & \text{if } \alpha = f. \end{cases}$$

The cost $C_1(\{1, 2\})$ for the unary constraint C_1 on $\{1, 2\}$ equals the sum of all the unary element costs:

$$\begin{aligned} C_1(\{1, 2\}) &= \varphi_{(1)/1}(t) \oplus \varphi_{(1)/2}(t) \oplus \varphi_{(1)/3}(f) \\ &= 1 \oplus 0 \oplus 0 = 1 \end{aligned}$$

As in integer WCSPs, we assume there is a unary constraint for each set variable. The domain and unary constraint of a set variable is interchangeable. When $C_i(u) = \top$, the unary constraint prohibits the variable S_i taking the set value u; otherwise it allows such assignment with cost $C_i(u)$. As reasoning on each domain value for a set variable is impractical, we focus on the bounds of a domain.

Definition 3. *The* domain bounds *of a set variable S_i is $[RS(S_i), PS(S_i)]$ such that $\forall a \in RS(S_i), \varphi_{(i)/a}(f) \oplus C_\emptyset = \top$ and $\forall b \in PS(S_i), \varphi_{(i)/b}(t) \oplus C_\emptyset < \top$.*

When $\varphi_{(i)/a}(t) \oplus C_\emptyset < \top$, the unary element cost function allows the existence of the set element a in the set S_i with the corresponding cost. Therefore the set element a can exist in the set. When $\varphi_{(i)/a}(f) \oplus C_\emptyset = \top$, the unary element cost function forbids the inexistence of the set element a, and the set element a must exist in the set. According to the unary constraint C_i in Figure 2(a), we have $PS(S_i) = \{1, 2\}$ and $RS(S_i) = \{2\}$ since $C_\emptyset = 0$ and $\top = 3$.

Binary Constraints A binary constraint C_{ij} assigns costs to assignments to variables S_i and S_j ($C_{ij} : D(S_i) \times D(S_j) \rightarrow [0, \ldots, k]$). The corresponding *binary element cost function*, which assigns costs for the existence states of a set element $a \in \mathcal{U}_i \cup \mathcal{U}_j$ for set variables S_i and S_j, is $\varphi_{(i,j)/a} : \{t, f\} \times \{t, f\} \rightarrow [0, \ldots, k]$. Since $\mathcal{U}_i$ may not be equal to $\mathcal{U}_j$, $\forall \alpha, \beta \in \{t, f\}$, the binary element cost function $\varphi_{(i,j)/a}(t, \alpha) = \top, \forall a \notin \mathcal{U}_i$ and $\varphi_{(i,j)/a}(\beta, t) = \top, \forall a \notin \mathcal{U}_j$. The binary constraint of the set variables S_i and S_j can be refined as: $C_{ij}(u, v) = \sum_{a \in \mathcal{U}_i \cup \mathcal{U}_j} \varphi_{(i,j)/a}(a \in u, a \in v)$.

Figure 2(a) shows the binary element costs among the set elements of S_1 and S_2. The costs are indicated on the lines linking the existence states of the elements in the two sets. No line is drawn if the cost is 0. According to the figure, the element cost for 1 in S_1 and S_2 is:

$$\varphi_{(1,2)/1}(\alpha, \beta) = \begin{cases} 1 & \text{if } \alpha = t \wedge \beta = t; \\ 3 & \text{if } \alpha = f \wedge \beta = t; \\ 0 & \text{otherwise} \end{cases}$$

The binary cost for $S_1 = \{1, 2, 3\}$ and $S_2 = \{1, 3\}$ is the sum of all the binary element costs:

$$\begin{aligned} & C_{12}(\{1, 2, 3\}, \{1, 3\}) \\ &= \varphi_{(1,2)/1}(t, t) \oplus \varphi_{(1,2)/2}(t, f) \oplus \varphi_{(1,2)/3}(t, t) \\ &= 1 \oplus 1 \oplus 0 = 2 \end{aligned}$$

Ternary Constraints A ternary constraint C_{ijk} assigns costs to assignments to variables S_i, S_j and S_k ($C_{ijk} : D(S_i) \times D(S_j) \times D(S_k) \rightarrow [0, \ldots, k]$). The corresponding *ternary element cost function*, which assigns costs to the existence states of a set element $a \in \mathcal{U}_i \cup \mathcal{U}_j \cup \mathcal{U}_k$ for variables S_i, S_j and S_k, is $\varphi_{(i,j,k)/a} : \{t, f\} \times \{t, f\} \times \{t, f\} \rightarrow [0, \ldots, k]$.

Similar to the case for the binary constraint, there may be an element $a \in \mathcal{U}_i \cup \mathcal{U}_j \cup \mathcal{U}_k$ where $a \notin \mathcal{U}_i$. In this case, all the ternary element cost functions taking $a \in S_i$ return $\top$ as the cost. This is also the same for variables S_j and S_k. The ternary constraint of the set variables S_i, S_j and S_k can be refined as: $C_{ijk}(u, v, w) = \sum_{a \in \mathcal{U}_i \cup \mathcal{U}_j \cup \mathcal{U}_k} \varphi_{(i,j,k)/a}(a \in u, a \in v, a \in w)$.

Cardinality Constraints A cardinality constraint $C_{|i|}$ assigns costs to assignments to a set variable S_i according to the cardinality of S_i. It is decomposed as $C_{|i|} = (Cost_{|i|} \circ Card)$ where $Card : D(S_i) \rightarrow \mathbb{N} \cup \{0\}$ and $Cost_{|i|} : \mathbb{N} \cup \{0\} \rightarrow [0, \ldots, k]$. This constraint first maps the assignment of the variable S_i to its cardinality $|S_i|$ by using $Card$. It then assigns costs to $|S_i|$ by $Cost_{|i|}$.

Property 1. *All constraints in our framework can be modeled as hard constraints in WCSP with costs 0 and $\top$.*

Proof. Some examples are listed in Table 1 while the construction is similar for the rest of set constraints. □

Set Constraint	Equivalent Cost Function
$a \in S_i$	$\varphi_{(i)/a}(\alpha) = \begin{cases} \top & \text{if } \alpha = f; \\ \bot & \text{otherwise.} \end{cases}$
$S_i \subseteq S_j$	$\forall a \in \mathcal{U}_i \cup \mathcal{U}_j, \varphi_{(i,j)/a}(\alpha, \beta) = \begin{cases} \top & \text{if } \alpha = t \wedge \beta = f; \\ \bot & \text{otherwise.} \end{cases}$
$S_i \cup S_j = S_k$	$\forall a \in \mathcal{U}_i \cup \mathcal{U}_j \cup \mathcal{U}_k, \varphi_{(i,j,k)/a}(\alpha, \beta, \gamma) = \begin{cases} \top & \text{if } (\alpha = t \vee \beta = t) \wedge \gamma = f; \\ \top & \text{if } \alpha = f \wedge \beta = f \wedge \gamma = t; \\ \bot & \text{otherwise.} \end{cases}$
$S_i \cap S_j = S_k$	$\forall a \in \mathcal{U}_i \cup \mathcal{U}_j \cup \mathcal{U}_k, \varphi_{(i,j,k)/a}(\alpha, \beta, \gamma) = \begin{cases} \top & \text{if } (\alpha = f \vee \beta = f) \wedge \gamma = t; \\ \top & \text{if } \alpha = t \wedge \beta = t \wedge \gamma = f; \\ \bot & \text{otherwise.} \end{cases}$
$S_i \backslash S_j = S_k$	$\forall a \in \mathcal{U}_i \cup \mathcal{U}_j \cup \mathcal{U}_k, \varphi_{(i,j,k)/a}(\alpha, \beta, \gamma) = \begin{cases} \top & \text{if } \beta = t \wedge \gamma = t; \\ \top & \text{if } \alpha = t \wedge \beta = f \wedge \gamma = f; \\ \top & \text{if } \alpha = f \wedge \beta = f \wedge \gamma = t; \\ \bot & \text{otherwise.} \end{cases}$
$\lvert S_i \rvert = m$	$C_{\lvert i \rvert} = \begin{cases} \top & \text{if } \lvert S_i \rvert \neq m; \\ \bot & \text{otherwise.} \end{cases}$

Table 1: Soft versions of some classical set constraints

Property 2. *When a WCSP with set variables involves only 0 and $\top$ in the element costs, the WCSP can be transformed into a problem with classical set constraints only.*

Proof. Since constraints in a WCSP can be refined to corresponding element cost functions, this can be shown by transforming every element cost function to a classical membership constraint. For each unary element cost function, $\varphi_{(i)/a}(t) = \top$ becomes $a \notin S_i$ and $\varphi_{(i)/a}(f) = \top$ becomes $a \in S_i$. The transformations for binary element cost functions are listed below.

Cost Function	Classical Constraint
$\varphi_{(i,j)/a}(t,t) = \top$	$a \notin (S_i \cap S_j)$
$\varphi_{(i,j)/a}(t,f) = \top$	$a \notin (S_i \backslash S_j)$
$\varphi_{(i,j)/a}(f,t) = \top$	$a \notin (S_j \backslash S_i)$
$\varphi_{(i,j)/a}(f,f) = \top$	$a \in (S_i \cup S_j)$

The transformations for ternary element cost functions are similar. For the cardinality constraints, we transform each n such that $Cost_{|i|}(n) = \top$ to $|S_i| \neq n$ as a classical constraint. □

Theorem 1. *WCSP with set variables subsumes CSP with set variables.*

Proof. This follows directly from Properties 1 and 2. □

Consistency Notions

As we are dealing with element cost functions, local consistency notions are defined at the element level. Element node consistency, element arc consistency, and element hyper-arc consistency are defined for the corresponding element cost functions.

Definition 4. *An existence state α of set element a is* element node consistent (ENC) *w.r.t. unary constraint C_i if $C_\emptyset \oplus \varphi_{(i)/a}(\alpha) < \top$. A set element is ENC if (1) all its possible existence states are ENC w.r.t. unary constraint C_i and (2) $\exists \alpha \in \{t, f\}$ such that $\varphi_{(i)/a}(\alpha) = \bot$. The existence state α is a support for the set element a. The set variable is ENC w.r.t. unary constraint C_i if every set element is ENC. The problem is ENC if every set variable is ENC.*

Definition 5. *An existence state α of set element a is* element arc consistent (EAC) *w.r.t. binary constraint C_{ij} if $\exists \beta \in \{t, f\}$ such that $\varphi_{(i,j)/a}(\alpha, \beta) = \bot$. An existence state β is a support of the existence state α. The set element is EAC if all its possible existence states is EAC w.r.t. the binary constraint C_{ij}. A set variable is EAC if every set element is EAC w.r.t. binary constraint C_{ij}. The problem is EAC if every set variable is EAC and ENC.*

Definition 6. *An existence state α of set element a is* element hyper-arc consistent (EHAC) *w.r.t. ternary constraint C_{ijk} if $\exists \beta, \gamma \in \{t, f\}$ such that $\varphi_{(i,j,k)/a}(\alpha, \beta, \gamma) = \bot$. Existence states β and γ are supports of the existence state α. The set element is EHAC if all its possible existence states is EHAC w.r.t. ternary constraint C_{ijk}. A set variable is EHAC if every set element is EHAC w.r.t. ternary constraint C_{ijk}. The problem is EHAC if every set variable is EHAC and ENC.*

For the cardinality constraint, we adopt a weighted cardinality consistency which maintains the maximum cardinality interval inside the range $[|RS(S_i)|, |PS(S_i)|]$ for the corresponding set variable S_i while removing the inconsistent cardinality from the bounds.

Definition 7. *The* cardinality upper bound $ub(|S_i|)$ *and* lower bound $lb(|S_i|)$ *of a cardinality constraint $C_{|i|}$ are the maximum and minimum cardinalities, respectively, such that $Cost_{|i|} \oplus C_\emptyset < \top$.*

Definition 8. *A set variable S_i is* weighted cardinality consistent (WCC) *w.r.t. a cardinality constraint $C_{|i|}$ if (1) the cardinality upper bound $ub(|S_i|) \leq |PS(S_i)|$ and the cardinality lower bound $lb(|S_i|) \geq |RS(S_i)|$, (2) $Cost_{|i|}(lb(|S_i|)) \oplus C_\emptyset < \top$ and $Cost_{|i|}(ub(|S_i|)) \oplus C_\emptyset < \top$, and (3) $\exists n$ such that $lb(|S_i|) \leq n \leq ub(|S_i|)$ and $Cost_{|i|}(n) = \bot$.*

As in classical set CSPs, we do not reason about each domain value in the set domain due to its high complexity. Instead, we will enforce the consistency in the bounds of the set domain. Since our constraints are refined to a set of element cost functions, we consider the cost for the existence of each set element. In fact, this is equivalent to reasoning on the bounds of set domains.

Definition 9. *A set variable S is* weighted set bounds consistent (WSBC) *w.r.t. a constraint C if $RS(S) = \bigcap\{u\} \wedge PS(S) = \bigcup\{u\}$ where $u \in [RS(S), PS(S)]$ and $S = u$ can be extended to a tuple t such that $C(t) \oplus C_\emptyset < \top$.*

Theorem 2. *A set variable is* WSBC *w.r.t. unary constraint C_i (or binary constraint C_{ij} or ternary constraint C_{ijk}) if it is ENC (or EAC or EHAC).*

Proof. By the definition of $RS(S)$ and $PS(S)$ for set variable S, it is trivial that any set element in $RS(S)$ must exist and any set element not in $PS(S)$ must not exist. The following proves that no extra elements can be put in $RS(S)$ or taken out from $PS(S)$ when ENC (or EAC or EHAC) is enforced.

For unary constraint, suppose $\exists a \notin RS(S_i)$ such that $\forall u \in D(S_i), a \notin u \rightarrow C(u) \oplus C_\emptyset = \top$. When S_i is ENC, $a \notin RS(S_i)$ implies $C_\emptyset \oplus \varphi_{(i)/a}(f) < \top$. We can always construct a set value v for S_i such that $C_i(v) = 0$. Now, we set $a \notin S_i$ and form new a set value w with cost $\varphi_{(i)/a}(f)$, then $C_i(w) \oplus C_\emptyset < \top$ leads to contradiction. For binary (or ternary) constraint, when S_i is EAC (or EHAC) we can find a support for the set value w' for S_i where $a \notin w'$ with cost 0 w.r.t. the binary (ternary) constraint. Therefore, $a \notin RS(S_i)$.

On the other hand, suppose $\exists a \in PS(S_i)$ such that $\forall u \in D(S_i), a \in u \rightarrow C(u) \oplus C_\emptyset = \top$. Since $a \in PS(S_i)$, $C_\emptyset \oplus \varphi_{(i)/a}(t) < \top$. We can always construct a set value v for S_i such that $C_i(v) = 0$. Now we set $a \in S_i$ and form a set value w with cost $\varphi_{(i)/a}(t)$, then $C_i(w) \oplus C_\emptyset < \top$ leads to contradiction. Similar to the above case, we can find a support for w' for S_i where $a \in w'$ with cost 0 w.r.t. the binary (ternary) constraint. Therefore, $a \in PS(S_i)$. □

Theorem 3. *When a WCSP with set variables involves costs 0 and $\top$ only, WSBC = SBC.*

Proof. By the definition of $RS(S)$ and $PS(S)$ for set variable S, since $\forall a \in RS(S), C_\emptyset \oplus \varphi_{(i)/a}(f) = \top$, any set value must contain element a. In addition, since $\forall a \notin PS(S), C_\emptyset \oplus \varphi_{(i)/a}(t) = \top$, any set value must not contain a. According to Theorem 2, WSBC ensures that each set element $a \in PS(S) \backslash RS(S)$ can be extended to form a set value with cost $C_\emptyset \oplus \varphi_{(i)/a}(t) < \top$. Since there are costs 0 and $\top$ only, $C_\emptyset \oplus \varphi_{(i)/a}(t) = 0$ which implies that the set element a can be contained in the set value. □

```
1  Procedure ReviseCardinality (i)
2  lb(|S_i|) := max(lb(|S_i|), |RS(S_i)|);
3  ub(|S_i|) := min(ub(|S_i|), |PS(S_i)|);
4  while Cost_|i|(lb(|S_i|)) ⊕ C_∅ = ⊤ do
5      lb(|S_i|) := lb(|S_i|) + 1;
6  while Cost_|i|(ub(|S_i|)) ⊕ C_∅ = ⊤ do
7      ub(|S_i|) := ub(|S_i|) − 1;
8  if lb(|S_i|) = ub(|S_i|) then
9      if lb(|S_i|) = |RS(S_i)| then
10         for a ∈ PS(S_i)\RS(S_i) do push(Q, S_i, a);
11         PS(S_i) := RS(S_i);
12     if ub(|S_i|) = |PS(S_i)| then
13         for a ∈ PS(S_i)\RS(S_i) do push(Q, S_i, a);
14         RS(S_i) := PS(S_i);
15 Function BoundsChanged (i, a) : Boolean
16 change := false;
17 if C_∅ ⊕ φ_(i)/a(t) = ⊤ then
18     PS(S_i) := PS(S_i)\{a};   change := true;
19 else if C_∅ ⊕ φ_(i)/a(f) = ⊤ then
20     RS(S_i) := RS(S_i) ∪ {a};   change := true;
21 if change then ReviseCardinality (i);
22 return change;
23 Procedure WSBC (X, D, C)
24 for S_i ∈ X do  for a ∈ U_i do push(Q, S_i, a);
25 while Q ≠ ∅ do
26     (S_i, a) := pop(Q);
27     FindUnarySupports (i, a);
28     for C_ij ∈ C do
29         FindBinarySupports (i, j, a);
30         if BoundsChanged (j, a) then
31             Q := Q ∪ (x_j, a);
32     for C_ijk ∈ C do
33         FindTernarySupports (i, j, k, a);
34         if BoundsChanged (j, a) then
35             Q := Q ∪ (x_j, a);
36         if BoundsChanged (k, a) then
37             Q := Q ∪ (x_k, a);
```

Algorithm 1: Enforcing WSBC

Enforcing Consistency

Consistencies can be enforced in a similar way as those defined by Larrosa (2002). To enforce ENC, the cost $min\{\varphi_{(i)/a}(t), \varphi_{(i)/a}(f)\}$ is identified and added to the global lower bound. At the same time, such unit of cost will be subtracted from the unary element costs of a ($\varphi_{(i)/a}(t)$ and $\varphi_{(i)/a}(f)$). For the EAC and EHAC, we will find the minimum cost for each existence state of a set element. When the cost is identified, such cost will be added to the unary cost of the corresponding existence state. At the same time the related binary (resp. ternary) element costs will be deducted by the same amount. This operation will be done in `FindUnarySupports`, `FindBinarySupports` and `FindTernarySupports` respectively.

For example, in Figure 2(a), the set element 1 of S_1 is not ENC. The minimum cost of the existence states is 1, so 1 is sent to the global lower bound $C_\emptyset$ and each unary element cost is decreased by 1. Also, the binary constraint is not EAC because $1 \in S_2$ does not have a binary support. The minimum cost among $\varphi_{(1,2)/1}(t,t)$ and $\varphi_{(1,2)/1}(f,t)$, which is 1, is sent to the unary element cost of $\varphi_{(2)/1}(t)$. Figure 2(b) shows the WCSP which is ENC and EAC.

For the cardinality consistency, it is required to enforce whenever there is a change in the bounds of the domain. This can be done by `ReviseCardinality`. In this procedure, the bounds of the cardinality constraint is first revised to ensure the bounds are consistent. When the bounds of cardinality are the same and it is equal to either the cardinality of $PS(S)$ or $RS(S)$, it will force the elements in $PS(S) \backslash RS(S)$ to exist or not to exist in the set value accordingly. Algorithm 1 illustrates the process of maintaining ENC, EAC, EHAC and WCC so as to enforce WSBC.

In `WSBC`, all the elements from the universal set of each set variable are pushed to the queue. Suppose there are n set variables with maximum number of elements e. There are ne elements in the queue. For each element, there can be $\mathcal{O}(n)$ binary constraints and $\mathcal{O}(n^2)$ ternary constraints related to it. Since each set element only has two existence states, `FindUnarySupports`, `FindBinarySupports` and `FindTernarySupports` take $\mathcal{O}(1)$ to scan for supports. Each set element affect the bounds once when it is assigned to exist or not exist in the set. The maximum queue size is $2ne$. The cardinality constraint is scanned once for the whole iteration for each set variable, which has complexity $\mathcal{O}(ne)$. The complexity is $\mathcal{O}(2ne(n+n^2)+ne) = \mathcal{O}(n^3e)$.

Experimental Results

We modified ToolBar, a generic WCSP solver, to adopt set variables and conducted experiments to verify the feasibility of our proposal. The comparison is made among our prototype implementation, the original ToolBar and ILOG Solver 6.0 (for classical cases only). While our implementation (labeled as TB-Set) and ILOG use set variables in modeling, the problems are transformed to use 0-1 variables for the original ToolBar (labeled as TB-01) to solve. The 0-1 versions of unary, binary and ternary constraints are similar to the corresponding element cost functions. However, the cardinality constraint on a set variable S_i with $|\mathcal{U}_i| = n$ becomes an n-ary constraint which maps the existence states of n set elements to the cost of its cardinality.

We experiment on the Steiner Triple System and the So-

cial Golfer Problem, which are well known set CSP benchmarks. We solve for all solutions to make our results independent of search heuristics. The problems are softened in two ways. The first version *Restricted* is modified by assigning unary cost randomly generated from 0 to 9 to each of the unary element cost. The second version *Relaxed* is modified by replacing the cost $\top$ with a randomly generated cost from 1 to $\top$. The version *Restricted* reduces the search space of the problem while *Relaxed* increases the search space. To measure the runtime for these two versions, we generated 10 instances for each problem instance and report the average runtime. The experiments were conducted on a Sun Blade 2500 (2 $\times$ 1.6GHz US-IIIi) machine with 2GB memory.

Steiner Triple System (CSPLib044)

This problem of order n is to find a set of $n(n-1)/6$ triples of distinct integer elements in $\{1, \ldots, n\}$ such that no two triples have more than one common element.

	Classical			Restricted		Relaxed	
n	**ILOG**	**TB-Set**	**TB-01**	**TB-Set**	**TB-01**	**TB-Set**	**TB-01**
6	0.10	0.05	1.64	0.05	1.68	0.21	2.84
7	31.52	16.84	-	5.40	263.51	46.17	-

Table 2: Runtime (in sec) for solving Steiner Triple System

The runtime of solving the problem for all solutions is listed in Table 2 with '-' indicating the program does not stop in 600 seconds. Because of the long solving time for the problem of order 9, we only focus on the problem up to order 7. The results show that TB-Set is faster than ILOG in solving the problem. When the problem is modeled with 0-1 variables, TB-01 requires longer time for solving. This is because the existing consistency enforcing algorithms do not perform well on the 0-1 variables. In addition, the 0-1 representation of the cardinality constraints as non-binary constraints further degrades the performance of TB-01.

The Social Golfer Problem (CSPLib010)

This problem is to schedule g groups of s golfers over w weeks so that no two golfers play in the same group twice.

	Classical			Restricted		Relaxed	
g-s-w	**ILOG**	**TB-Set**	**TB-01**	**TB-Set**	**TB-01**	**TB-Set**	**TB-01**
3-2-4	1.26	0.60	52.94	0.13	9.87	0.98	58.15
3-2-5	8.66	4.12	-	1.40	192.84	8.00	-
3-3-3	0.28	0.15	11.27	0.06	3.51	0.37	15.58
3-3-4	2.22	1.29	231.83	0.58	91.33	5.77	402.20
4-2-3	58.49	24.84	-	0.97	63.51	30.27	-
4-3-2	1.46	0.69	43.12	0.05	1.45	0.95	46.63
4-4-2	13.10	6.28	545.97	0.27	12.36	14.23	-
5-2-2	1.99	0.82	59.93	0.05	1.77	0.89	61.95
6-2-2	142.51	55.60	-	0.81	47.51	59.05	-

Table 3: Runtime (in sec) for solving Social Golfer Problem

Table 3 shows the results of solving the problem for all solutions. The symbol '-' indicates the program cannot stop in 600 seconds. The results are consistent with the previous benchmark. When comparing the original ToolBar and our modification for set variables, the runtime of TB-Set for solving both the *Restricted* and *Relaxed* versions is two orders of magnitude faster than TB-01. This verifies the feasibility and performance of our proposal.

Conclusion and Future Work

Problems involving set variables are common. Set constraint solving techniques are well studied in classical CSPs. The integer WCSP framework can handle soft problems efficiently on the integer domain. However, the current definitions for local consistency is impractical to process set variables in WCSPs. We have proposed our definition of set variables with some local consistency notions. Since the satisfiability of common set constraints depends on the existence of set elements among the set variables in each constraint, we define and specify set variables and constraints based on the (in)existence of set elements. We also define weighted set bounds consistency and show how it can be enforced by maintaining local consistency at the element level. On the other hand, we introduce the cardinality constraint for set variables, which otherwise needs to be composed by an n-ary constraint. Experiments show that our proposal is two times faster than ILOG in solving most classical set problems and two orders of magnitude faster than original ToolBar in solving both classical and soft set problems.

Set-based WCSPs open up possibilities for future research. Hawkins, Lagoon, & Stuckey (2005) show how set variables in classical CSPs can be represented by reduced ordered binary decision diagrams (ROBDDs), and give efficient algorithm to enforce domain consistency. It will be interesting to study if the same principle can be extended for WCSPs. Another research direction is to investigate techniques for cardinality reasoning (Müller & Müller 1997; Azevedo & Barahona 2000).

Acknowledgments

We thank the anonymous referees for constructive comments. The work described in this paper was substantially supported by a grant (CUHK4358/02E) from the Research Grants Council of Hong Kong SAR.

References

Azevedo, F., and Barahona, P. 2000. Modelling digital circuits problems with set constraints. In *Computational Logic*, 414–428.

Bouveret, S.; Heras, F.; de Givry, S.; Larrosa, J.; Sanchez, M.; and Schiex, T. 2004. ToolBar: a state-of-the-art platform for WCSP. http://www.inra.fr/bia/T/degivry/ToolBar.pdf.

Cleary, J. G. 1987. Logical arithmetic. *Future Computing Systems* 2(2):125–149.

Gervet, C. 1997. Interval propagation to reason about sets: definition and implementation of a practical language. *Constraints* 1(3):191–244.

Hawkins, P.; Lagoon, V.; and Stuckey, P. J. 2005. Solving set constraint satisfaction problems using ROBDDs. *Journal of Artificial Intelligence Research* 24:109–156.

ILOG. 2003. *ILOG Solver 6.0 Reference Manual*.

Larrosa, J. 2002. Node and arc consistency in weighted CSP. In *Proceedings of AAAI 2002*, 48–53.

Müller, T., and Müller, M. 1997. Finite set constraints in Oz. In *13. Workshop Logische Programmierung*, 104–115.