Stronger Consistencies in WCSPs with Set Variables

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

Abstract

Lee and Siu made possible for the first time modeling and reasoning with set variables in weighted constraint satisfaction problems (WCSPs). In addition to an efficient set variable representation scheme, they also defined the notion of set bounds consistency, which is generalized from NC and AC* for integer variables in WCSPs, and their associated enforcement algorithms. In this paper, we adapt ideas from FDAC and EDAC for integer variables to achieve stronger consistency notions for set variables. The generalization is non-trivial due to the common occurrence of ternary set constraints. Enforcement algorithms for the new consistencies are proposed. Empirical results confirm the feasibility and efficiency of our proposal.*

1 Introduction

Set variables and constraints are ubiquitous in modeling many (soft) constraint problems in practice. While set variables in *weighted constraint satisfaction problems* (WCSPs) [4] can be simulated using 0-1 variables, Lee and Siu [5] showed such direct simulation is impractical. They proposed a compact representation scheme for set variables in WCSPs. Adapted from *star node consistency* (NC*) and *star arc consistency* (AC*) [4] for integer variables in WCSPs, basic consistency notions, including *element node consistency* (ENC), *element arc consistency* (EAC), and *element hyper-arc consistency* (EHAC), and their associated enforcement algorithms are defined and evaluated.

Stronger consistency can prune larger part of the search space. Larrosa et. al. introduced *full directional arc consistency* (FDAC) [3] and *existential directional arc consistency* (EDAC) [1] for integer variables, and demonstrated how reasoning overheads can be well justified by the extra reduction in the search space leading to a faster runtime. By enforcing stronger consistency, we can often solve the problems in shorter time. In this paper, we adapt FDAC and EDAC for set variables and constraints in WCSPs.

Consistency notions, including FDAC and EDAC, on WCSPs for integer variables are defined for unary and binary constraints only. Enforcing consistencies on constraints of higher arities are expensive. However, set models in WCSPs usually contain soften versions of set constraints such as membership ($e \in A$), subset ($A \subseteq B$), equality ($A = B$), union-equal($A \cup B = C$) and intersect-equal ($A \cap B = C$). In particular, union-equal and intersect-equal constraints are common. We need to define consistency notions for these *possibly ternary* set constraints. The constraints can obviously be generalized to arbitrary arity. To balance the cost of consistency enforcement and efficiency, we support consistency notions up to ternary constraints only. One challenge in deriving the set variable counterparts is thus in dealing with ternary constraints and the related consistency notions. We give also efficient enforcement algorithms for the new consistencies to make solving set problem with stronger consistencies viable. Experimental results confirm the benefits of maintaining these stronger consistencies. By enforcing the stronger consistency, we can obtain one order of magnitude in improvement on the problem solving.

2 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 take a value from its variable domain $D(x_i) \in \mathcal{D}$. An n-ary 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.

A *weighted constraint satisfaction problem* (WCSP) extends classical CSP. A WCSP is a tuple $\mathcal{P} = (k, \mathcal{X}, \mathcal{D}, \mathcal{C})$ which relies on a *valuation structure* $S(k) = ([0, 1, \ldots, k], \oplus, \geq)$, 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 $\geq$ among naturals. There are

minimum cost $\perp = 0$ and maximum cost $\top = k$. In addition, cost subtraction $\ominus$ is defined as follows:

$$a \ominus b = \begin{cases} a - b & \text{if } a \neq k \\ k & \text{otherwise} \end{cases}$$

where $a, b \in [0, \ldots, k]$ and $a \geq b$. $\mathcal{X}$ and $\mathcal{D}$ are the finite set of variables and the finite set of variable domains respectively. A tuple of assignments $(x_{i_1} \mapsto d_{i_1}, \ldots, x_{i_n} \mapsto d_{i_n})$ assigns values $d_{i_j} \in D(x_{i_j})$, where $D(x_{i_j}) \in \mathcal{D}$, to variable $x_{i_j} \in \mathcal{X}$, for $1 \leq j \leq n$. $\mathcal{C}$ is the set of *cost functions* which maps a tuple of assignments to costs. There is a zero-arity constraint $C_\emptyset$ representing the global lower bound of the problem.

The cost $\mathcal{V}(\tau)$ of a tuple of variable assignments τ is obtained by combining the costs for all constraints using $\oplus$:

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

The notation $var(\tau)$ is the set of variables appearing in τ and $\tau \downarrow_{\{x_{i_1},\ldots,x_{i_n}\}}$ projects τ on $\{x_{i_1}, \ldots, x_{i_n}\}$. When $\mathcal{V}(\tau) < \top$, the tuple τ is said to be *consistent*. In solving WCSP, our goal is to find a complete consistent assignment with minimum cost.

A WCSP with set variables [5] contains a finite set of set variables $S_i \in \mathcal{X}$. Each set variable S_i can contains elements from its own universe $\mathcal{U}_i$ defined by the problem. An *existence state* of a set element a w.r.t. set value u is defined as a boolean value of $a \in u$. While the existence state of 1 in the set $\{1, 3\}$ is t (true), the existence state of 2 is f (false). The set of *possible existence states* of a set element a for a set variable S_i is denoted as $E(S_i, a)$. Initially, $E(S_i, a) = \{t, f\}$. When S_i cannot contain a, t is removed from $E(S_i, a)$. Similar case applies to existence state f. The set constraints are cost functions mapping tuples of set variable assignments to costs. Since common set constraints, such as subset ($A \subseteq B$) and union-equal ($A \cup B = C$), consider the existence states of each set element individually, the cost of a set constraint is defined at the element level through *element cost functions*.

A *unary element cost function* $\varphi_{(i)/a}$ maps existence states of a set element a of set variable S_i to costs. The unary constraint of S_i is defined as $C_i(u) = \sum_{a \in \mathcal{U}_i} \varphi_{(i)/a}(a \in u)$. Set domain in classical CSP is represented as an interval with lower and upper bounds, namely *required set* (RS) and *possible set* (PS) respectively. Required set contains set elements which *must* belong to the set while possible set contains set elements which *may* belong to the set. With the unary element cost functions, we can deduce the required set and possible set of a set variable. A set element a is in required set of S_i if $\varphi_{(i)/a}(f) = \top$. A set element a is in possible set of S_i if $\varphi_{(i)/a}(t) < \top$.

Figure 1(a) shows a WCSP with set variables. The costs of lower bound $C_\emptyset$ and upper bound $\top$ is shown on top. There are two set variables S_1 and S_2, each with two possible set elements $\{1, 2\}$. The upper left and right tables represent unary constraints of S_1 and S_2 respectively. For example, the cost for S_1 to contain set element 1 is 1 and the cost for S_1 not to contain set element 1 is 0.

The binary and ternary set constraints are defined in a similar fashion. A binary set constraint is defined as $C_{i,j}(u, v) = \sum_{a \in \mathcal{U}_i \cup \mathcal{U}_j} \varphi_{(i,j)/a}(a \in u, a \in v)$. A ternary set constraint is defined as $C_{i,j,k}(u, v, w) = \sum_{a \in \mathcal{U}_i \cup \mathcal{U}_j \mathcal{U}_k} \varphi_{(i,j,k)/a}(a \in u, a \in v, a \in w)$. The dashed rectangle in Figure 1(a) represents a binary constraint $C_{1,2}$. A dotted line separates the table for each set element vertically. The table is a binary element cost function of $C_{1,2}$. The left portion of the table shows the costs for set element 1 while the right portion shows the costs for set element 2. From the table, the cost of $\varphi_{(1,2)/1}(t, t)$ is 2.

There are cardinality constraints $C_{|i|}$ mapping assignments of set variable S_i to costs according to the cardinality of S_i. In this paper, we only focus on unary, binary, and ternary set constraints which are involved in the stronger consistency notions.

3 Basic consistency notions

The framework of WCSP with set variables defined some basic consistency notions which are based on *star node consistency* and *star arc consistency* in the integer domain [4].

Definition 1 *An existence state α of set element b is* element node consistent *(ENC) w.r.t. unary constraint C_i if $C_\emptyset \oplus \varphi_{(i)/b}(\alpha) < \top$. A set element b is* ENC *if*

1. *$\forall \alpha \in E(S_i, b)$, α is ENC w.r.t. C_i, and*
2. *$\exists \alpha \in E(S_i, b)$ such that $\varphi_{(i)/b}(\alpha) = 0$.*

The existence state α is a support for the set element b. A set variable S_i is ENC *w.r.t. C_i if every set element is ENC. A WCSP is* ENC *if every set variable S_i is ENC.*

Definition 2 *An existence state α of set element b is* element arc consistent *(EAC) w.r.t. binary constraint $C_{i,j}$ if $\exists \beta \in E(S_j, b)$ such that $\varphi_{(i,j)/b}(\alpha, \beta) = 0$. An existence state β is a* simple support *of the existence state α. A set element b is* EAC *if $\forall \alpha \in E(S_i, b)$, α is EAC w.r.t. $C_{i,j}$. A set variable is* EAC *if every set element is EAC w.r.t. $C_{i,j}$. A WCSP is* EAC *if every set variable is EAC and ENC.*

Definition 3 *An existence state α of set element b is* element hyper-arc consistent *(EHAC) w.r.t. ternary constraint $C_{i,j,k}$ if $\exists \beta \in E(S_j, b)$ and $\exists \gamma \in E(S_k, b)$ such that $\varphi_{(i,j,k)/b}(\alpha, \beta, \gamma) = 0$. Existence states β and γ are* simple supports *of the existence state α. A set element b is* EHAC

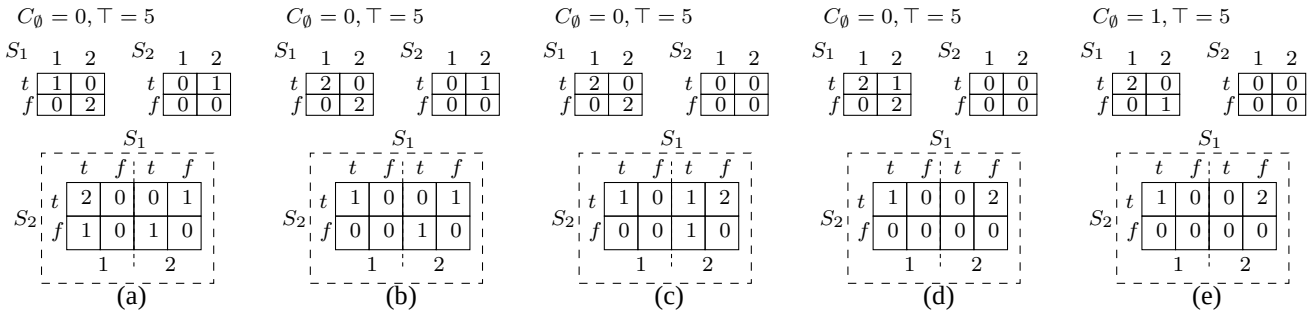

Figure 1. Equivalent WCSPs with set variables.

if $\forall \alpha \in E(S_i, b)$, α *is EHAC w.r.t.* $C_{i,j,k}$. *A set variable is* EHAC *if every set element is EHAC w.r.t.* $C_{i,j,k}$. *A WCSP is* EHAC *if every set variable is EHAC and ENC.*

For example, in Figure 1(a), the WCSP is ENC but not EAC. The existence state $1 \in S_1$ does not have support in $C_{1,2}$. By transferring the minimum costs of $\varphi_{(1,2)/1}(t,t)$ and $\varphi_{(1,2)/1}(t,f)$ to unary element cost $\varphi_{(1)/1}(t)$, the WCSP is transformed into the one in Figure 1(b), which is now EAC. Such operation is called *cost projection* which is used to maintain consistency. The WCSP is transformed into an equivalent WCSP by projecting the costs.

When all the above consistency notions are enforced, the problem is said to be *weighted set bounds consistent* (WSBC), which is a generalized notion of *set bounds consistency* (SBC) in classical set CSP defined in [2]. The complexity of WSBC enforcing algorithm in [5] is $\mathcal{O}(n^3 e)$ when there are n set variables with maximum number of elements e in the problem.

4 Stronger consistency notions

We now propose our stronger consistency notions in WCSPs with set variables. A consistency A is *stronger* than a consistency B, written $A \succ B$, if whenever a problem is A consistent, the problem is B consistent. Enforcing stronger consistency can reduce the search space of a problem.

In the integer domain, there are two stronger consistency notions: *full directional arc consistency* (FDAC) [3] and *existential directional arc consistency* (EDAC) [1]. Both of them are based on *directional arc consistency* [3] and require an ordering on the variables. Without loss of generality, we order the variables in WCSP as $x_i < x_j$ if $i < j$. In maintaining directional arc consistency, different level of consistencies are applied on $C_{i,j}$ and $C_{j,i}$, which are effectively the same constraint but with different order of variables, because it is impractical to enforce such strong consistency in both directions [1].

The stronger consistency notions for WCSP with set variables are as follows.

Definition 4 *A set element* b *of a set variable* S_i *is* element full directional arc consistent *(EFDAC) w.r.t. binary constraint* $C_{i,j}$, *where* $i < j$, *if* $\forall \alpha \in E(S_i, b), \exists \beta \in E(S_j, b)$ *such that* $\varphi_{(i,j)/b}(\alpha, \beta) \oplus \varphi_{(j)/b}(\beta) = 0$. *The existence state* β *is a* full support *of* α. *The set element* b *is* EFDAC *w.r.t.* C_{ij}, *where* $j < i$, *if* $\forall \alpha \in E(S_i, b), \exists \beta \in E(S_j, b)$ *such that* $\varphi_{(i,j)/b}(\alpha, \beta) = 0$. *A set variable is* EFDAC *if all its set elements are EFDAC w.r.t. binary constraint* $C_{i,j}$. *The problem is* EFDAC *if every set variable is EFDAC and ENC.*

The notion of EFDAC is a direct generalization of FDAC to the set domain. To extend the notion to ternary, we need to have the notion of full support for ternary constraints, which, as far as we know, is not available in the literature. The notion of EFDAC is generalized to ternary set constraint with our definition of ternary full support as follows:

Definition 5 *A set element* b *of a set variable* S_i *is* element full directional hyper-arc consistent (EFDHAC) *w.r.t. ternary constraint* $C_{i,j,k}$*:*

1. *for* $i < j < k$, *if* $\forall \alpha \in E(S_i, b)$, $\exists \beta \in E(S_j, b)$ *and* $\exists \gamma \in E(S_k, b)$ *such that* $\varphi_{(i,j,k)/b}(\alpha, \beta, \gamma) \oplus \varphi_{(j)/b}(\beta) \oplus \varphi_{(k)/b}(\gamma) = 0$. *The existence states* β *and* γ *are the* full supports *of* α.
2. *for* $j < i < k$, *if* $\forall \alpha \in E(S_i, b), \exists \beta \in E(S_j, b), \gamma \in E(S_k, b)$ *such that* $\varphi_{(i,j,k)/b}(\alpha, \beta, \gamma) \oplus \varphi_{(k)/b}(\gamma) = 0$. *The existence states* β *and* γ *are the* simple support *and* full support *of* α *respectively.*
3. *for* $j < k < i$, *if* $\forall \alpha \in E(S_i, b), \exists \beta \in E(S_j, b), \gamma \in E(S_k, b)$ *such that* $\varphi_{(i,j,k)/b}(\alpha, \beta, \gamma) = 0$. *The existence states* β *and* γ *are the* simple supports *of* α.

A set variable is EFDHAC *if all its set elements are EFDHAC w.r.t.* $C_{i,j,k}$. *The problem is* EFDHAC *if every set variable is EFDHAC and ENC.*

Our definition of ternary full support is consistent with the definition of binary full support when we consider the

smallest and largest variables in the constraint: the smallest variable requires full supports for all other variables and the largest variable requires simple supports only. A variable always finds simple support from the smaller variables and finds full support from the larger variables.

The problem in Figure 1(b) is not EFDAC because t for element 2 of set variable S_1 does not have full support in S_2 in $C_{1,2}$. Unlike finding simple support, there is no cost for projection. To ensure a support for $2 \in S_1$, we *extend* the unary cost of $\varphi_{(2)/2}(t)$ to binary costs $\varphi_{(1,2)/2}(t,t)$ and $\varphi_{(1,2)/2}(f,t)$ as in Figure 1(c). Then, we can project the cost to $\varphi_{(1)/2}(t)$ which is shown in Figure 1(d). The problem is not ENC and we project the unary cost of set element 2 of set variable S_1 to $C_\emptyset$. The results depicted in Figure 1(e) is now EFDAC.

Before we introduce the stronger notion of *element existential directional hyper-arc consistency* (EEDHAC), we first give our generalization of *star existential arc consistency* (EAC*) in the integer domain [1], namely *element existential hyper-arc consistency* (EEHAC). The definition of EEDHAC is based on the EEHAC. We immediately define the notion at hyper-arc level instead of arc level since all the constraints in the problem are under consideration.

Definition 6 *A set element b of a set variable S_i is* element existential hyper-arc consistent (EEHAC) *if $\exists \alpha \in E(S_i, b)$ such that $\varphi_{(i)/b}(\alpha) = 0$ and α has a full support in every binary constraint $C_{i,j}$ and every ternary constraint $C_{i,j,k}$. A set variable is* EEHAC *if all its set elements are EEHAC. The problem is* EEHAC *is every set variable is EEHAC and ENC.*

Definition 7 *A problem is* element existential directional hyper-arc consistent (EEDHAC) *if it is EFDAC, EFDHAC, and EEHAC.*

Figure 2(a) shows a WCSP which is EFDAC. It is not EEDHAC since the existence state t of the set element 1 of S_3 does not have full support in $C_{1,3}$ and f of the same set element does not have full support in $C_{2,3}$. The unary costs of $\varphi_{(1)/1}(f)$ and $\varphi_{(2)/1}(t)$ are extended to $C_{1,3}$ and $C_{2,3}$ respectively as shown in Figure 2(b). The costs in these two constraints are then projected to the unary cost of C_3 as depicted in Figure 2(c). Since the problem is not ENC, the unary cost of C_3 is further projected to $C_\emptyset$. The WCSP in Figure 2(d) shows the result which is now EEDHAC.

We can incorporate EFDAC, EFDHAC, and EEDHAC into the weighted set bounds consistency.

Definition 8 *A WCSP is* full directional weighted set bounds consistent (FDWSBC) *if it is weighted set bounds consistent, EFDAC, and EFDHAC.*

Definition 9 *A WCSP is* existential directional weighted set bounds consistent (EDWSBC) *if it is weighted set bounds consistent and EEDHAC.*

The definitions of FDWSBC and EDWSBC subsume the definition of WSBC. Additionally, the definition of EEDHAC also subsumes EFDAC and EFDHAC. We immediately have the following theorem.

Theorem 1 *EDWSBC $\succ$ FDWSBC $\succ$ WSBC.*

Proof 1 *Given a WCSP with set variables $\mathcal{P}$, we can transform $\mathcal{P}$ into equivalent WCSPs $\mathcal{P}_1$, which is EDWSBC, and $\mathcal{P}_2$, which is FDWSBC. For each constraint C_1 in P_1 and its corresponding constraint C_2 in P_2, C_1 and C_2 are of the same consistency level if they are unary or cardinality constraints. Otherwise, C_1 is EEDHAC and C_2 is EFDAC (or EFDHAC if C_2 is a ternary constraint). As EEDHAC is stronger than EFDAC or EFDHAC by definition, EDWSBC is also stronger than FDWSBC. By similar argument, FDWSBC is stronger than WSBC.*

5 Consistency enforcement

Procedure `findUSup(`i, a`)`
$c := \min_{\alpha \in E(S_i,a)}(\varphi_{(i)/a}(\alpha))$;
$C_\emptyset := C_\emptyset \oplus c$;
for $\alpha \in E(S_i, a)$ **do**
  $\varphi_{(i)/a}(\alpha) := \varphi_{(i)/a}(\alpha) \ominus c$;

Procedure `projTern(`$i,\ j,\ k,\ a,\ \gamma,\ c$`)`
$\varphi_{(k)/a}(\gamma) := \varphi_{(k)/a}(\gamma) \oplus c$;
for $\alpha \in E(S_i, a), \beta \in E(S_j, a)$ **do**
  $\varphi_{(i,j,k)/a}(\alpha, \beta, \gamma) := \varphi_{(i,j,k)/a}(\alpha, \beta, \gamma) \ominus c$;

Procedure `extTern(`$i,\ j,\ k,\ a,\ \gamma,\ c$`)`
for $\alpha \in E(S_i, a), \beta \in E(S_j, a)$ **do**
  $\varphi_{(i,j,k)/a}(\alpha, \beta, \gamma) := \varphi_{(i,j,k)/a}(\alpha, \beta, \gamma) \oplus c$;
$\varphi_{(k)/a}(\gamma) := \varphi_{(k)/a}(\gamma) \ominus c$;

Figure 3. Finding unary support and projection/extension of ternary costs.

The algorithms for enforcing EFDAC, EFDHAC, and EEDHAC are based on cost extension and projection [4] to find the required supports. The algorithm shown in Figure 3 describes the procedure to find the unary supports. In procedure `findUSup`, the minimum cost among the possible existence states is subtracted from the unary element costs and projected to $C_\emptyset$. As a result, there are at least one possible existence state which has unary cost 0 to satisfy the element node consistency. The procedures for finding binary simple and full supports are omitted here since we focus on ternary case. The procedure `projTern` is used to project cost c from the ternary element cost function $\varphi_{(i,j,k)/a}$ to unary element cost function $\varphi_{(k)/a}$ for the existence state

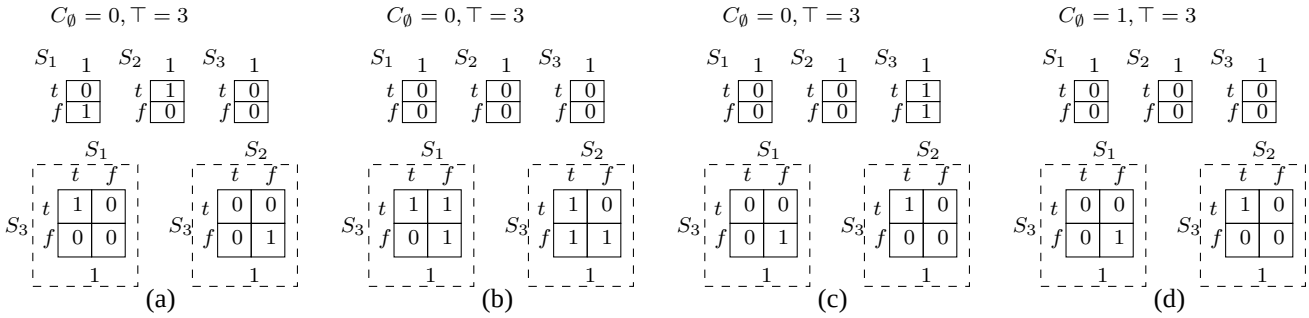

Figure 2. Transformation of WCSP from EFDAC to EEDHAC.

of γ for set variable S_k. The procedure `extTern` works in another direction. It extends the unary element cost of γ for set variable S_k from $\varphi_{(k)/a}$ to the ternary element cost function $\varphi_{(i,j,k)/a}$. All these three procedures are useful in enforcing ternary supports.

It is more complex to enforce full supports in ternary constraints. There are three possible cases as shown in Figure 4. First, it only requires to enforce simple ternary supports. The procedure `findTernSimpSup` is used to enforce simple supports for variable S_k. After getting the minimum cost in ternary element cost function, `projTern` is called to project the minimum cost to unary element cost function of S_k. Second, it requires to find a simple support in one variable and a full support in another variable. The procedure `findTernSemiFullSup` handles this case. It first calculates the costs to be projected in an array P for each possible existence states of S_j. Since some costs may need to be extended from S_k, the costs to be extended is stored in an array E. After getting these two arrays, the procedure can extend and project costs using `projTern` and `extTern`. In the last case, it enforces full support on both of two variables with the use of `findTernFullSup`. The structure of them are similar to `findTernSemiFullSup`. However, this time we may need to extend the cost from unary element cost functions of both other variables.

In enforcing binary full support, when we want to find a full support in S_1 for a value in S_2 and there is insufficient cost in binary constraint $C_{1,2}$ to project to unary constraint C_2, we can first extend the unary cost in C_1 to $C_{1,2}$. Then the binary cost in $C_{1,2}$ can be project to C_2. The procedure is straightforward as there is only one source of costs for extension. However, in ternary constraint, when we enforce a full support for a variable, there are two source of costs. An example is shown in Figure 5(a). The dashed rectangle represents a ternary constraint $C_{1,2,3}$. Inside the rectangle, the upper table represents the costs when $1 \in S_1$ while the lower table represents the costs when $1 \notin S_1$. Thus the cost of $\varphi_{(1,2,3)/1}(t, f, f)$ is 2. Now, a full support of $1 \in S_1$ is required. There are three costs can be projected to the unary element cost $1 \in S_1$. If we choose to extend the unary element cost $1 \notin S_2$ to the ternary element function, we will have the problem in Figure 5(b). As the costs of $\varphi_{(1,2,3)/1}(t, t, t)$ and $\varphi_{(1,2,3)/1}(t, f, t)$ is less than three, there must be some cost required to extend to $1 \in S_3$. The problem after the extension is shown in Figure 5(c). All the unary element costs in the problem are extended to the ternary element cost function. Alternatively, we can extend the unary element cost of $1 \in S_3$ before extending the cost of $1 \notin S_2$ which results in the problem in Figure 5(d). While the cost to be projected from $\varphi_{(1,2,3)/1}$ to $\varphi_{(1)/1}$ is the same, there is some cost left in $\varphi_{(2)/1}(f)$. A cost in unary element function is more explicit than in ternary constraint. When later in the search, a cost is projected to $\varphi_{(2)/1}(t)$, the unary element cost of $1 \in S_2$ can be immediately projected to $C_\emptyset$ to raise the global lower bound. However, deciding which costs to be extended first will increase the complexity of cost extension. In our implementation, the order of cost extension is arranged lexicographically starting with the smallest variable.

It is observed that the cost extension and projection can affect the availability of existing supports. Figure 6 shows an example. In Figure 6(a) the ternary set constraint is not element full directional hyper-arc consistent since there is no full supports in S_2 and S_3 for $1 \in S_1$. To enforcing full supports, the unary element costs of $1 \notin S_2$ and $1 \in S_3$ are extended to $\varphi_{(1,2,3)/1}$. The result is shown in Figure 6(b). In the next step, the costs of the ternary element cost function are *projected* to the unary cost of $1 \in S_1$ as shown in Figure 6(c). These two steps ensure full support in binary constraint. However, it is not the case for ternary constraint as the simple support for $1 \notin S_1$ is lost after the above steps. We have to ensure simple supports for $1 \notin S_1$. To circumvent the situation, the algorithm invokes procedures to ensure the availability of more basic supports at the end. For example, at the end of `findTernFullSup`, it invokes `findTernSemiFullSup` to ensure there is a support for other variables in the same constraint.

Figure 7 shows the algorithm for enforcing EEDHAC. We use the procedure `findExistSup` to find an existence

Procedure `findTernSimpSup(`i,j,k,a`)`
$flag := false$;
for $\gamma \in E(S_k, a)$ **do**
  $c := \min_{\alpha \in E(S_i,a), \beta \in E(S_j,a)}(\varphi_{(i,j,k)/a}(\alpha, \beta, \gamma))$;
  if $c > \bot$ **then**
    if $\varphi_{(k)/a}(\gamma) = \bot$ **then** $flag := true$;
    `projTern(`i,j,k,a,γ,c`)`;
`findUSup(`k,a`)`;
return $flag$;

Procedure `findTernSemiFullSup(`i,j,k,a`)`
$flag := false$;
for $\beta \in E(S_j, a)$ **do**
  $P[\beta] := \min_{\alpha \in E(S_i,a), \gamma \in E(S_k,a)}(\varphi_{(i,j,k)/a}(\alpha, \beta, \gamma) \oplus \varphi_{(k)/a}(\gamma))$;
  if $P[\beta] > \bot \wedge \varphi_{(j)/a}(\beta) = \bot$ **then** $flag := true$;
for $\gamma \in E(S_k, a)$ **do**
  $E[\gamma] := \max_{\alpha \in E(S_i,a), \beta \in E(S_j,a)}(P[\beta] \ominus \varphi_{(i,j,k)/a}(\alpha, \beta, \gamma))$;
for $\gamma \in E(S_k, a)$ **do**
  `extTern(`$i,j,k,a,\gamma,E[\gamma]$`)`;
for $\beta \in E(S_j, a)$ **do**
  `projTern(`$i,k,j,a,\beta,P[\beta]$`)`;
`findUSup(`j,a`)`;
$flag := flag \vee$ `findTernSimpSup(`i,j,k,a`)`;
return $flag$;

Procedure `findTernFullSup(`i,j,k,a`)`
$flag := false$;
for $\alpha \in E(S_i, a)$ **do**
  $P[\alpha] = \min_{\beta \in E(S_j,a), \gamma \in E(S_k,a)}(\varphi_{(i,j,k)/a}(\alpha, \beta, \gamma) \oplus \varphi_{(j)/a}(\beta) \oplus \varphi_{(k)/a}(\gamma))$;
  if $P[\alpha] > \bot \wedge \varphi_{(i)/a}(\alpha) == \bot$ **then** $flag := true$;
for $\beta \in E(S_j, a)$ **do**
  $E_j[\beta] := \max_{\alpha \in E(S_i,a), \gamma \in E(S_k,a)}(P[\alpha] \ominus \varphi_{(i,j,k)/a}(\alpha, \beta, \gamma))$;
  $E_j[\beta] := \min(\varphi_{(j)/a}(\beta), E_j[\beta])$;
for $\gamma \in E(S_k, a)$ **do**
  $E_k[\gamma] := \max_{\alpha \in E(S_i,a), \beta \in E(S_j,a)}(P[\alpha] \ominus E_j[\beta] \ominus \varphi_{(i,j,k)/a}(\alpha, \beta, \gamma))$;
for $\beta \in E(S_j, a)$ **do**
  `extTern(`$i,k,j,a,\beta,E_j[\beta]$`)`;
for $\gamma \in E(S_k, a)$ **do**
  `extTern(`$i,j,k,a,\gamma,E_k[\gamma]$`)`;
for $\alpha \in E(S_i, a)$ **do**
  `projTern(`$j,k,i,a,\alpha,P[\alpha]$`)`;
`findUSup(`i,a`)`;
$flag := flag \vee$ `findTernSemiFullSup(`i,j,k,a`)`;
return $flag$;

Figure 4. Finding ternary supports.

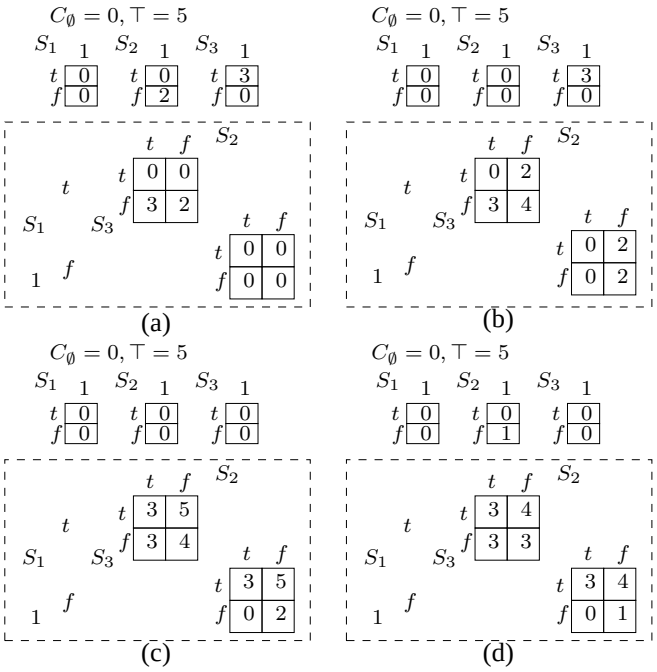

Figure 5. Different choices of cost extension in ternary constraint.

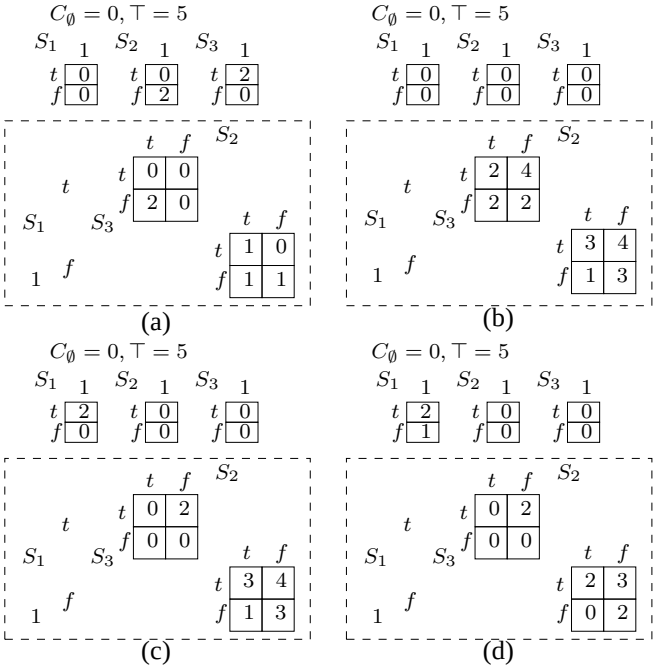

Figure 6. Enforcing full supports in ternary constraints can lose simple support.

Procedure findExistSup(i, a)
$flag := false$;
$c :=$
$\min_{\alpha \in E(S_i,a)}(\varphi_{(i)/a} \bigoplus_{\varphi_{(i,j)/a}} \text{s.t. } j<i \min_{\beta \in E(S_j,a)}(\varphi_{(i,j)/a} \oplus \varphi_{(j)/a})$
$\bigoplus_{\varphi_{(i,j,k)/a}} \text{s.t. } j<i<k \min_{\beta \in E(S_j,a),\gamma \in E(S_k,a)}(\varphi_{(i,j,k)/a} \oplus \varphi_{(j)/a}))$
$\bigoplus_{\varphi_{(i,j,k)/a}} \text{s.t. } j<k<i \min_{\beta \in E(S_j,a),\gamma \in E(S_k,a)}(\varphi_{(i,j,k)/a} \oplus \varphi_{(j)/a} \oplus \varphi_{(k)/a}))$;
if $c > \bot$ **then**
  for $\varphi_{(i,j)/a}$ s.t. $j < i$ **do**
    $flag := flag \vee$ findBinFullSup(i, j, a);
  for $\varphi_{(i,j,k)/a}$ s.t. $j < i < k$ **do**
    $flag := flag \vee$ findTernSemiFullSup(k, i, j, a);
  for $\varphi_{(i,j,k)/a}$ s.t. $j < k < i$ **do**
    $flag := flag \vee$ findTernFullSup(i, j, k, a);
return $flag$;

Procedure EEDHAC($\mathcal{X}$,$\mathcal{D}$,$\mathcal{C}$)
for $S_i \in \mathcal{X}$ **do**
  for $a \in \mathcal{U}_i$ **do**
    $R := R \cup (i, a)$;
    $S := S \cup (i, a)$;
while $R \neq \emptyset \vee S \neq \emptyset$ **do**
  $P := \{(j, a) \mid (i, a) \in S, j > i, C_{ij} \in \mathcal{C} \vee C_{ijk} \in \mathcal{C}\} \cup S$;
  $S := \emptyset$;
  while $P \neq \emptyset$ **do**
    $(i, a) :=$ popMin(P);
    if findExistSup(i,a) **then**
      $R := R \cup \{(i, a)\}$;
      for $C_{ij} \in \mathcal{C} \vee C_{ijk} \in \mathcal{C}$ s.t. $j > i$ **do**
        $P := P \cup \{(j, a)\}$;
  while $R \neq \emptyset$ **do**
    $(i, a) :=$ popMax(R);
    for $\varphi_{(i,j)/a}$ s.t. $i < j$ **do**
      if FindBinFullSup(i,j,a) **then**
        $R := R \cup \{(j, a)\}$;
        $S := S \cup \{(j, a)\}$;
    for $\varphi_{(i,j,k)/a}$ s.t. $j < i < k$ **do**
      if findTernSemiFullSup(j,i,k,a) **then**
        $R := R \cup \{(k, a)\}$;
        $S := S \cup \{(k, a)\}$;
    for $\varphi_{(i,j,k)/a}$ s.t. $i < j < k$ **do**
      if findTernFullSup(i,j,k,a) **then**
        $R := R \cup \{(j, a), (k, a)\}$;
        $S := S \cup \{(j, a), (k, a)\}$;

Figure 7. Enforcing EEDHAC.

state satisfying the property of EEDHAC. If no such existence state is found, procedures are invoked to force full supports on the constraints involving the set element.

Procedure EEDHAC, which is integrated as the last step of enforcing WSBC, enforces EEDHAC on the problem. Thus, the problem is assumed to be ENC, EAC, and EHAC. There are two priority queues holding the set elements which may not be consistent. Since enforcing full support in one direction can lose full support in another direction, set element of other variables need to enqueue again. *We can obtain an algorithm for enforcing EFDAC and EFDHAC by removing the first inner while-loop of the EEDHAC algorithm.* In the following, we give the complexity of only the EEDHAC algorithm.

Theorem 2 *The complexity of* EEDHAC *is* $\mathcal{O}(c(ne)^3)$, *where there are* c *constraints and* n *set variables with maximum number of possible elements* e.

Proof 2 *The procedures for finding supports, projection and extension are of constant time as their complexity is determined by the fixed number of possible existence states. Now, the procedure* findExistSup *requires examining each constraints when finding or enforcing supports, so it has complexity* $\mathcal{O}(c)$. *For* EEDHAC, *the while-loop with queue* P *contains all the set variable and set element pairs. When there is a change to the cost of the pair, the pair will re-enqueue. Therefore, it can iterate at most* $(ne)^3$ *times. Each time it calls* findExistSup *with* $\mathcal{O}(c)$. *The while-loop with queue* R *iterates less time than the one with queue* P *as only the pairs in the same constraint are re-enqueued. Thus, the complexity of* EEDHAC *is* $\mathcal{O}(c(ne)^3)$.

6 Experimental results

We conducted experiments with implementation of different consistency enforcing algorithms to compare their performance. We modified ToolBar[1], which is a generic WCSP solver in the integer domain, to adopt WCSP with set variables. We tested on three different levels of consistencies: (1) WSBC, (2) FDWSBC, and (3) EDWSBC.

We experimented on the *weighted versions* of two benchmark problems: Steiner Triple System and Social Golfer Problem. To eliminate the effect of search heuristics, we solved the problems for all optimal solutions using the lexicographic ordering on both variable and value selections. The experiments were conducted on Sun Blade 2500 (2 $\times$ 1.6GHz US-IIIi) machines with 2GB memory. The average runtime in seconds and number of fails of 10 instances are measured. The shortest runtime and smallest number of fails are in bold. The '-' indicates the instance did not terminate in 1800 seconds.

[1] Available at http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro.

Table 1. Runtime in seconds and number of fails to solve Steiner Triple System

	WSBC		FDWSBC		EDWSBC	
n	**Time**	**Fails**	**Time**	**Fails**	**Time**	**Fails**
6	0.0	2508.0	0.0	2051.0	0.0	**2037.6**
7	2.6	403698.3	1.9	295890.7	**1.8**	**283001.7**
9	812.5	32760909.4	40.4	1298722.0	**18.6**	**907508.6**
10	-	-	221.1	3625737.8	**25.4**	**1101655.2**

Table 2. Runtime in seconds and number of fails to solve Social Golfer Problem

	WSBC		FDWSBC		EDWSBC	
g-s-w	**Time**	**Fails**	**Time**	**Fails**	**Time**	**Fails**
4-2-4	13.78	423451.3	**2.07**	21080.1	2.30	**15652.0**
4-2-5	363.53	6459221.4	**27.93**	193848.5	35.36	**153661.3**
4-3-3	9.50	439445.0	**1.79**	38747.5	1.80	**28241.9**
4-3-4	-	-	426.55	5560572.7	**263.37**	**2089071.9**
4-4-3	151.07	6229119.3	39.78	831585.9	**33.83**	**544083.1**
5-2-3	19.84	973711.9	3.82	54306.2	**3.15**	**33693.8**
5-3-3	-	-	205.48	2927103.8	**154.43**	**1623065.9**
5-4-2	13.11	1047593.8	**3.45**	242389.4	3.94	**229203.4**
5-5-2	115.11	9115070.9	11.73	445503.0	**10.21**	**320592.9**
6-2-3	-	-	193.19	2013072.5	**33.09**	**228857.9**
6-3-2	38.15	3223477.9	5.48	175822.7	**2.06**	**49102.7**
6-4-2	-	-	206.35	6098467.7	**156.51**	**3766946.9**

6.1 Steiner Triple System (CSPLib044)

The 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 any pair of triples have *at most one* common element. We form an over-constrained version by requiring that *no* common element between any pair of triples. Each violation contributes a cost to the problem. Unary costs were added randomly to simulate the suitability of an element in a triple.

The result is depicted in Table 1. We can observe that EDWSBC always have the smallest number of fails as EDWSBC is the strongest consistency notion in the experiment. With the increase of the problem size, FDWSBC and EDWSBC prune more search space. The time required to solve the problem with FDWSBC and EDWSBC is shorter when compared with WSBC.

6.2 Social Golfer Problem (CSPLib010)

The problem is to schedule g groups of s golfers over w weeks so that no golfer plays in the same group with any other golfer in more than one occasion. Extra random costs are added to reflect the preferences of golfers to each group. We have to optimize the schedule to respect the preferences.

Table 2 shows the result of solving the problems. FDWSBC and EDWSBC have smaller number of fails than WSBC. With the large reduction in number of fails, the larger pruning power of the stronger consistency is confirmed and thus the solving times are much shorter. As EDWSBC enforcement has higher complexity than FDWSBC, in some instances, FDWSBC has shorter runtime even with larger number of fails. However, when the search space is effectively pruned, such overhead of EDWSBC can be justified resulting in the shortest runtime, especially for large instances. Generally, enforcing stronger consistency leads to an order of magnitude improvement in runtime than enforcing WSBC. While some instances cannot be solved within time limit with WSBC, it is feasible to solve them by enforcing FDWSBC and EDWSBC.

7 Conclusion and future work

WCSPs with set variables enable modeling of optimization and over-constrained problems involving sets of objects naturally. While the existing consistency notions for set variables in WCSPs are basic ones, there are more sophisticated notions in the integer domain. In this paper, we extend FDAC and EDAC to the set domain. One challenge is to extend the notions for ternary constraints and come up with efficient consistency algorithms.

Algorithms are designed to enforce the adopted EFDAC, EFDHAC, and EEDHAC. Experimental results show that maintaining the stronger consistencies can improve one order of magnitude in problem solving when compared with WSBC. However, there is large overhead of maintaining EEDHAC in some instances, which requires improvement.

8 Acknowledgments

We thank the anonymous referees for constructive comments. The work described in this paper was substantially supported by grants (CUHK413207 and CUHK413808) from the Research Grants Council of Hong Kong SAR.

References

[1] S. de Givry, F. Heras, M. Zytnicki, and J. Larrosa. Existential arc consistency: Getting closer to full arc consistency in weighted CSPs. In *Proceedings of the 19th IJCAI*, pages 84–89, 2005.

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

[3] J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted CSP. In *Proceedings of the 18th IJCAI*, pages 239–244, 2003.

[4] J. Larrosa and T. Schiex. Solving weighted CSP by maintaining arc consistency. *Artificial Intelligence*, 159(1-2):1–26, 2004.

[5] J. H. M. Lee and C. F. K. Siu. Weighted constraint satisfaction with set variables. In *Proceedings of the 21st AAAI*, pages 80–85, 2006.