Consistency Techniques for Polytime Linear Global Cost Functions in Weighted Constraint Satisfaction

J.H.M. Lee K.L. Leung Y.W. Shum
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong

January 4, 2014

Abstract

Lee and Leung make practical the consistency enforcement of global cost functions in Weighted Constraint Satisfaction Problems (WCSPs). The main idea of their approach lies in the derivation of polynomial time algorithms for the computation of the minimum cost of global cost functions. In this paper, we investigate how soft arc consistency can also be applied on global cost functions with no known efficient minimum cost computation algorithms. We propose *polytime linear projection-safe (PLPS)* cost functions, which have a polynomial size integer linear formulation and can maintain this good property across projection/extension operations. We observe that the minimum of the linear relaxation gives a good approximation to the minimum of the integer formulation. This is used as the basis for the enforcement of *relaxed forms* of existing soft arc consistencies. By using the linear formulations, we can easily enforce conjunctions of overlapping PLPS, which give stronger pruning power. We further propose *polytime integral linear projection-safe (PILPS)* cost functions, which are PLPS cost functions with guaranteed integral solutions to the linear relaxation. We prove theorems to compare the consistency strengths among PLPS, PILPS and their conjunctions. Extensive experimentations are conducted to compare our proposed algorithms against state of the art global cost functions consistency enforcement algorithms and integer programming. Empirical results agree with our theoretical predictions, and confirm orders of magnitude improvement in terms of pruning and runtime by our proposals.

1 Introduction

Weighted Constraint Satisfaction Problems (WCSPs) [46] is a soft constraint framework for modeling over-constrained problems and those with preferences. It provides a general model for different applications, such as *resource allocation* [12], *combi-*

natorial auctions, *electronic markets* [45], *bioinformatics* [44], *probabilistic reasoning* [37], *planning* [17], *protein design* [4], *crop allocation problem* [2], etc.

A WCSP consists of a finite set of variables, a finite domain of possible values for each variable and a conjunction of cost functions. A cost function returns a cost for each tuple. Each variable assignment is associated with a cost. The costs could be used to represent preferences to the variable assignments.

Solving a WCSP is to find an assignment to the variables with the minimum cost. Such an assignment often represents the most preferred or the least violated situation. The basic solution technique for WCSPs is branch-and-bound search augmented with various forms of consistencies, such as NC* [24], AC* [24], FDAC* [25], and EDAC* [20]. These consistency techniques retrieve hidden information from cost functions by transporting costs and remove infeasible values from variable domains to prune the search space.

A good library of global cost functions is essential for us to model complex real-life problems in WCSPs. A global cost function often has high arities but also a special semantics, which allows for the design of tailor-made and efficient algorithms to enforce consistencies. Lee and Leung [30, 28] suggest *projection safety*, which is based on three requirements for consistencies to be practically enforced on global cost functions. First, computation of the minimum must be efficient. Second, projections and extensions on the cost functions can be performed efficiently. Third, projections and extensions on the cost functions will not destroy the first two efficiency requirements. Lee and Leung [30, 28] further demonstrate that flow-based global cost functions [48] satisfy the first two requirements and give instances that are flow-based projection-safe. In addition, Lee *et al.* [31] show that another class of cost functions, called *polynomially decomposable* cost functions, can satisfy these three requirements and give instances of cost functions which are polynomially decomposable.

Our goal is to introduce more practical global cost functions into the existing catalog. Many global cost functions are useful, but either their minimum computations are NP-hard or no polynomial time algorithms for computing their minimum costs have been discovered yet. An example is the soft variants of the DISJUNCTIVE constraint [21], which schedule jobs without overlapping in a non-preemptive scheduling problem. Known algorithms for computing their minimum cost are exponential.

We first show that the efficient minimum computations of global cost functions depend on the efficient enforcement of *generalized arc consistency (GAC)* [11] of their hard constraint counterparts. There are previous results on the NP-hardness of enforcing GAC on several global constraints, which immediately lead to the same results for the minimum computation of their soft variants. It is natural to ask whether there are methods to still use such cost functions efficiently in different ways in WCSPs. We address this problem for *cost functions which can be modeled as integer linear programs with relaxed consistencies*. By solving the integer linear programs with linear relaxation, approximations of their minima are obtained and used in the enforcement of the relaxed consistencies. Such consistencies can be enforced efficiently by linear programming algorithms due to their excellent average case behavior. We call this class of cost functions *polytime linear projection-safe (PLPS)*[1] cost functions.

[1]Formerly "polynomially linear projection-safe" [33].

We also consider the conjunctions of PLPS cost functions since the integer linear programming formulations of PLPS cost functions allow them to be conjoined easily. We present empirical results to demonstrate the benefits of propagating on conjunctions in terms of both runtime and pruning in general.

We introduce and give sufficient conditions for a special subclass of PLPS cost functions, namely *polytime integral linear projection-safe (PILPS)*[2] cost functions. Our results show that propagating on individual PILPS cost functions using the exact (or relaxed since they are the same) consistencies is weaker than propagating on the conjunction of all these PILPS cost functions using the relaxed versions of the consistencies. These results give exact characterization on the strength of the relaxed and exact consistencies on conjunctions of PILPS cost functions as compared against the corresponding exact consistencies on individual PILPS cost functions.

We also give experiments on various benchmarks to show the efficiency of our approaches. We compare soft instances of the car sequencing problem, examination timetabling problems, and fair scheduling with different modelings. Empirical results of our experiments agree with our theoretical prediction: compared with enforcing exact consistencies on individual cost functions, enforcing relaxed consistencies on conjoined cost functions gives orders of magnitude improvements, both in runtime and search space reduction. The rest of the paper is organized as follows. Section 2 provides backgrounds and necessary definitions for the rest of the paper. Section 3 gives our problem statements. We mainly focus on cost functions whose minimum computation has no known polynomial time algorithm (yet). Section 4 defines *polytime linear projection-safe (PLPS)* cost functions and Section 5 gives a list of relaxed consistencies. PLPS cost functions can be modeled as integer linear programs. A good lower bound of an integer linear program's minimum can be computed using the program's linear relaxation. Such a lower bound can be used to define a weaker but more efficient form of (approximated) consistency notions. Section 6 discusses the conjunctions of PLPS cost functions, and gives theoretical and empirical results showing that propagating the conjoined cost functions is beneficial. Section 7 defines *polytime integral linear projection-safe (PILPS)* cost functions as a special subclass of PLPS cost functions. Section 8 shows that propagating on individual PILPS cost functions using the exact consistencies is weaker than propagating on the conjunction of all these PILPS cost functions using the relaxed versions of the consistencies. Section 9 shows by experiments that our approach give significant improvements on solving real-life problems. Section 10 gives related work, whereas Section 11 summarizes our work and gives possible directions for future works.

This paper combines and extends the work of Lee *et al.* [33, 32].

2 Background

We give the preliminaries on weighted constraint satisfaction problems (WCSPs), global cost functions, and integer linear programs.

[2]Formerly "polynomially integral linear projection-safe" [32].

2.1 Weighted Constraint Satisfaction Problems

Weighted constraint satisfaction is a special subclass of valued constraint satisfaction [46] based on a cost valuation structure $V(k) = ([0 \dots k], \oplus, \leq)$. The structure $V(k)$ contains a set of integers $[0, \dots, k]$, where $k > 0$, with standard integer ordering $\leq$. Addition $\oplus$ is defined by $a \oplus b = \min(k, a + b)$, while subtraction $\ominus$ is defined for any a and b, where $a \geq b$, by:

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

Definition 2.1 *[46] A* Weighted Constraint Satisfaction Problem (WCSP) *is a tuple* $(\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathrm{k})$*, where:*

- $\mathcal{X} = \{x_1, x_2, \dots, x_n\}$ *is a set of* variables*;*
- $D(x_i) \in \mathcal{D}$ *is the finite* domain *of values for each variable* $x_i \in \mathcal{X}$*, only one value of which can be assigned to* x_i*;*
- $\mathcal{C}$ *is a set of* cost functions $W_S \in \mathcal{C}$ *with scope* $S = \{x_{s_1}, x_{s_2}, \dots, x_{s_n}\} \subseteq \mathcal{X}$ *that map a tuple* $\ell \in \mathcal{L}(S)$*, where* $\mathcal{L}(S) = D(x_{s_1}) \times \cdots \times D(x_{s_n})$*, to a cost in* $V(k)$*.*

Without loss of generality, we assume $\mathcal{C} = \{W_\varnothing\} \cup \{W_i \,|\, x_i \in \mathcal{X}\} \cup \mathcal{C}^+$. $W_\varnothing$ is the constant nullary cost function, representing the lower bound of the WCSP. W_i is a unary cost function associated with variable $x_i \in \mathcal{X}$, returning the *unary cost* for each value $v \in D(x_i)$. $\mathcal{C}^+$ is a set of cost functions with scopes of two or more variables. For simplicity, we denote the minimum of a cost function W_S as $\min\{W_S\}$.

An assignment on a set of variables can be represented by a tuple ℓ. We denote $\ell[x_i]$ to be the value assigned to x_i, and $\ell[S]$ to be the tuple formed from the assignment on variables in the set $S \subseteq \mathcal{X}$.

Definition 2.2 *Given a WCSP* $(\mathcal{X}, \mathcal{D}, \mathcal{C}, k)$*, we define the* cost *of a tuple* $\ell \in \mathcal{L}(\mathcal{X})$ *as*

$$cost(\ell) = W_\varnothing \oplus \bigoplus_{x_i \in \mathcal{X}} W_i(\ell[x_i]) \oplus \bigoplus_{W_S \in \mathcal{C}^+} W_S(\ell[S])$$

A tuple ℓ *is* feasible *if* $cost(\ell) < k$*. A tuple* ℓ *is* solution *of the WCSP if its cost is minimum among all the feasible tuples.*

WCSPs are typically solved with basic branch-and-bound search augmented with different consistency techniques, which remove infeasible values from domains while preserving the equivalence of the problem, *i.e.* the cost of the solution is unchanged.

Definition 2.3 *A* projection *of cost* α*, where* $\alpha \leq \min\{W_S(\ell) \mid \ell[x_i] = v \wedge \ell \in \mathcal{L}(S)\}$*, from* W_S *to* W_i *with respect to* $v \in D(x_i)$*, is a transformation of* (W_S, W_i) *to* (W'_S, W'_i) *with respect to a value* $v \in D(x_i)$ *and a cost* α*, such that:*

$$W'_i(u) = \begin{cases} W_i(u) \oplus \alpha & \text{if } u = v, \\ W_i(u) & \text{otherwise.} \end{cases} \qquad W'_S(\ell) = \begin{cases} W_S(\ell) \ominus \alpha & \text{if } \ell[x_i] = v, \\ W_S(\ell) & \text{otherwise.} \end{cases}$$

An extension *of cost α , where $\alpha \leq W_i(v)$, from W_i to W_S with respect to $v \in D(x_i)$, is a transformation of (W_S, W_i) to (W_S'', W_i'') with respect to a value $v \in D(x_i)$ and a cost α, such that:*

$$W_i''(u) = \begin{cases} W_i(u) \ominus \alpha & \text{if } u = v, \\ W_i(u) & \text{otherwise.} \end{cases} \qquad W_S''(\ell) = \begin{cases} W_S(\ell) \oplus \alpha & \text{if } \ell[x_i] = v, \\ W_S(\ell) & \text{otherwise.} \end{cases}$$

Different consistency notions have been defined for WCSPs including NC* [24], (G)AC* [24, 18], FD(G)AC* [25, 30], and (weak) ED(G)AC* [20, 30].

Definition 2.4 *[24] Suppose we are given a WCSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C}, k)$.*

- *A variable $x_i \in \mathcal{X}$ is NC* iff:*
  - *$W_\varnothing \oplus W_i(v) < k$ for all values $v \in D(x_i)$, and;*
  - *there exists a value $v \in D(x_i)$ such that $W_i(v) = 0$.*
- *A WCSP is NC* iff all its variables are NC*.*

NC* [24] increases $W_\varnothing$ by projecting costs from unary cost functions and removes infeasible values. It helps the branch and bound search to detect unsatisfiability by checking if an empty domain exists or $W_\varnothing$ reaches the upper bound k.

The procedure `enforceNC*()` in Algorithm 1 enforces NC* for a WCSP $(\mathcal{X}, \mathcal{D}, \mathcal{C}, k)$ [26]. The algorithm first projects cost from each variable. Then it removes infeasible values according to the lower bound $W_\varnothing$. The procedure `unaryProject()` projects a suitable cost from W_i to $W_\varnothing$ to produce a unary support, while the procedure `pruneVal()` removes the infeasible values which are not NC*.

```
1  Procedure enforceNC*()
2      foreach x_i ∈ X do unaryProject(x_i);
3      foreach x_i ∈ X do pruneVal(x_i);
4
5  Procedure unaryProject(x_i)
6      α := min{W_i};
7      W_∅ := W_∅ ⊕ α;
8      foreach v ∈ D(x_i) do W_i(v) := W_i(v) ⊖ α;
9
10 Procedure pruneVal(x_i):Boolean
11     foreach v ∈ D(x_i) s.t. W_i(v) ⊕ W_∅ = k do
12         D(x_i) := D(x_i)\{v};
13
```

Algorithm 1: Enforcing NC* for a WCSP

The definition of GAC* [18, 28, 30] involves the minimum of cost functions.

Definition 2.5 *[18, 28, 30] Suppose we are given a WCSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C}, k)$, a cost function $W_S \in \mathcal{C}^+$, and a variable $x_i \in S$.*

- *A variable $x_i \in S$ is* GAC* *with respect to W_S iff*
  - *x_i is NC*, and;*
  - *for each value $v \in D(x_i)$,*

$$\min\{W_S(\ell) \mid \ell \in \mathcal{L}(S) \wedge \ell[x_i] = v\} = 0$$

- *A WCSP is* GAC* *iff all variables are GAC* with respect to all cost functions in $\{W_S \mid W_S \in \mathcal{C}^+ \wedge x_i \in S\}$.*

The definition is slightly different from the one given by Cooper *et al.* [15], which also requires for every tuple $\ell \in \mathcal{L}(S)$, $W_S(\ell) = k$ if $W_\varnothing \oplus \bigoplus_{x_i \in S} W_i(\ell[x_i]) \oplus W_S(\ell) = k$.

Based on NC* [24] and GAC* [18, 28, 30], FDGAC* [28, 30] and weak EDGAC* [29, 30] are defined. Enforcing FDGAC* [28, 30] and weak EDGAC* [29, 30] is more complex, but they still rely on minimum computation.

2.2 Global Constraints and Global Cost Functions

A *global constraint* [6, 5, 43], denoted by $\mathrm{GC}(S, A_1, \ldots, A_k)$, is a family of hard constraints with precise semantics, parametrized by the variable scope S and other possible extra arguments $A_1, \ldots, A_k$. Usually, global constraints cannot be propagated efficiently using generic consistency algorithms due to their high arity scope. Dedicated and efficient propagation algorithms are designed by exploiting their special structures. Examples of global constraints include ALLDIFFERENT [27], GCC [41], SAME [7], AMONG[5], and REGULAR [38] constraints.

Global cost functions [50, 30] are soft variants of global constraints. A global cost function, denoted by $\text{W\_GC}^\mu(S, A_1, \ldots, A_k)$, is a family of cost functions in WCSP which returns the results computed by the violation measure μ associated with the global constraint $\mathrm{GC}(S, A_1, \ldots, A_k)$. The cost function W_GC^μ returns 0 iff a given tuple $\ell \in \mathcal{L}(S)$ satisfies GC. If ℓ violates GC, W_GC^μ returns $\mu(\ell)$ using the violation measure to reflect how much the GC is violated. Similar to global constraints, efficient algorithms for consistency enforcement have been designed for global cost functions, based on flow networks [30, 29] and dynamic programming [31]. For examples, the cost functions $\text{W\_ALLDIFFERENT}^{var}$ and $\text{W\_ALLDIFFERENT}^{dec}$ [28, 30] are derived from two different violation measures, namely variable-based and decomposition-based [39, 48] measures, of ALLDIFFERENT respectively.

In the rest of the paper, we denote a global cost function by W_S, where S is the scope of the cost function.

2.3 Integer Linear Programming

Integer linear programs [49] is a special case of linear programs [19] where all variables are also required to be integral. Without loss of generality, we consider only the minimization of the integer linear programs.

Definition 2.6 *An integer linear program I is a mathematical optimization problem defined as follows:*

$$\min \mathbf{c}^T \mathbf{x}$$
$$\mathbf{A}\mathbf{x} \leq \mathbf{b}$$
$$\mathbf{x} \in \mathbb{Z}^n$$

The vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ is a vector of variables*, which take values from the set of integers $\mathbb{Z}$ based on a set of inequalities. The inequalities are defined by $\mathbf{A} \in \mathbb{Q}^{m \cdot n}$, $\mathbf{b} \in \mathbb{Q}^m$, with n being the number of variables, m the number of problem constraints, and $\mathbb{Q}$ being the set of rational numbers. The expression $\mathbf{c}^T \mathbf{x}$, where $\mathbf{c} \in \mathbb{Q}^n$, is the objective function of I to be minimized.*

For simplicity, we overload the definition of a tuple γ to represent an *assignment* that is taken by $\mathbf{x}$. We define solutions of the integer linear program as follows.

Definition 2.7 *Suppose we are given an integer linear program I as in Definition 2.6. A* feasible solution *is a tuple γ that satisfies the inequalities $\mathbf{A}\mathbf{x} \leq \mathbf{b}$. An* optimal feasible solution *is a feasible solution γ_{opt} which minimizes the objective function $\mathbf{c}^T \mathbf{x}$. We also define the* minimum *of I, written as $\min(I)$, to be the minimum value of the objective function attained by an optimal feasible solution.*

Solving an integer linear program is NP-hard in general [49]. One key technique to approximate the solution of an integer linear program is *linear relaxation* [49].

Definition 2.8 *[49] A* linear relaxation *of an integer linear program is the linear program where all the variables are no longer required to be integral.*

The linear relaxation of an integer linear program I removes the integrality requirement, enlarges the set of feasible solutions, and provides a lower bound on the minimum of I. Since a linear program without integrality constraints is polynomially solvable [19], linear relaxation provides polynomial-time approximation for an integer linear program. Under certain conditions, solving linear relaxation can also give the exact solution of an integer linear program [49].

3 NP-Hard Global Cost Functions

In this paper, we say that a global cost function is *NP-hard* iff it is NP-hard to compute its minimum cost. Many useful global cost functions are NP-hard. In particular, a special class of such global cost functions is derived from global constraints, the *generalized arc consistency* (GAC) [11] of which is NP-hard to maintain.

Definition 3.1 *[11] Given a CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, we say that a constraint $C_S \in \mathcal{C}$ is* GAC *iff for every $x_i \in S$ and for every value $v_i \in D(x_i)$, there exists a tuple $\ell \in \mathcal{L}(S)$ such that $\ell[x_i] = v_i$ and ℓ satisfies C_S.*

Lemma 3.2 *Suppose we are given a constraint C_S for which the following problem of enforcing GAC is NP-hard:*

IsItGAC(C_S)
Instance. *A constraint C_S, the variables $x_i \in S$, and the domains $D(x_i)$ for every $x_i \in S$*
Question. *For every $x_i \in S$ and for every value $v_i \in D(x_i)$, does there exist a tuple $\ell \in \mathcal{L}(S)$ such that $\ell[x_i] = v_i$ and ℓ satisfies C_S?*

Define a cost function W_S to be the soft variant of C_S. The following problem of computing the minimum cost of W_S

IsOptimal(W_S, p)
Instance. *A global cost function W_S, a fixed integer p, the variables $x_i \in S$, and the domains $D(x_i)$ for every $x_i \in S$*
Question. *Does there exist a tuple $\ell \in \mathcal{L}(S)$ such that $W_S(\ell) \leq p$?*

is NP-hard.

Proof Solving IsItGAC(C_S) is equivalent to solving IsOptimal($W_S, 0$) k times, where $k = O(|S|d_{max})$ and $d_{max} = \max_{x_i \in S}\{|D(x_i)|\}$, with $D(x_i) = \{v_i\}$ for each variable $x_i \in S$ and each value $v_i \in D(x_i)$. If IsOptimal($W_S, 0$) is in the complexity class $\mathcal{P}$, so is IsItGAC(C_S). The result follows by contradiction. □

We give an example using SlidingSum [35]. The global constraint SlidingSum(S, Π) is defined based on a set of *windows* $p_i \in \Pi$. A window p_i is a tuple $\langle l_i, u_i, S_i \rangle$, which places a restriction on the sum of a set of variables in $S_i \subseteq S$ not less than a lower bound l_i and not greater than an upper bound u_i.

Definition 3.3 *[9] The constraint* Sum(S, l, u)*, accepts a tuple $\ell \in \mathcal{L}(S)$ iff*

$$l \leq \bigoplus_{x_i \in S} \ell[x_i] \leq u$$

Definition 3.4 *[35] The constraint* SlidingSum(S, Π)*, where $\Pi = \{p_i \mid i = 1 \ldots m\}$ and $p_i = \langle l_i, u_i, S_i \rangle$, accepts a tuple $\ell \in \mathcal{L}(S)$ iff for every $p_i \in \Pi$,*

$$l_i \leq \bigoplus_{x_j \in S_i} \ell[x_j] \leq u_i$$

Soft variants of SlidingSum can be derived by the fact that SlidingSum is a conjunction of multiple Sum constraints [9].

Using the *decomposition-based* violation measure, we can define W_SlidingSum$^{dec}(S)$ by summing up the violation of each Sum constraints measured by value-based violation measure.

Definition 3.5 *The cost function* W_Sum$^{val}(S, l, u)$ *returns the cost of a tuple $\ell \in \mathcal{L}(S)$ as:*

$$\text{W\_Sum}^{val}(S, l, u)(\ell) = \max(\bigoplus_{x_i \in S} \ell[x_i] - u, l - \bigoplus_{x_i \in S} \ell[x_i], 0)$$

Definition 3.6 *The cost function* W_SLIDINGSUM$^{dec}(S, \Pi)$ *returns the cost of a tuple* $\ell \in \mathcal{L}(S)$ *as:*

$$\text{W\_SLIDINGSUM}^{dec}(S, \Pi)(\ell) = \bigoplus_{i=1}^{m} \text{W\_SUM}^{val}(S_i, l_i, u_i)(\ell[S_i])$$

Note that the definitions are similar to those given by Bessière *et al.* [35].

We show that W_SLIDINGSUM^{dec} is NP-hard by Lemma 3.2.

Theorem 3.7 *Computing the minimum of* W_SLIDINGSUM^{dec} *is NP-hard.*

Proof Enforcing GAC on a SUM constraint is NP-hard [9]. As the SLIDINGSUM constraint can be represented by a conjunction of multiple SUM, enforcing GAC on SLIDINGSUM is NP-hard. Since W_SLIDINGSUM^{dec} is derived from the SLIDINGSUM constraint, by Lemma 3.2, the result follows. □

In addition to NP-hard cost functions, there are also cost functions which have no known polynomial time algorithms for their minimum cost computation yet. One example is the W_DISJUNCTIVE^{val} cost function derived from the DISJUNCTIVE constraint [21]. To handle these cost functions in WCSPs, *Polytime Linear Projection-Safe* (*PLPS*) cost functions are proposed.

4 Polytime Linear Projection-Safety

Koster [23] suggests a method to formulate global cost functions into integer linear programs by treating them as table cost functions and modeling the cost of each tuple by an inequality. However, the number of linear inequalities used can be exponential in the size of the scope of the cost function, which is undesirable if we are looking for efficient ways to solve them. We limit the size of a *linear cost function* and give the definition of a *polytime linear cost function*.

Definition 4.1 *A* linear cost function W_S *is a cost function which can be represented by an integer linear program* I_{W_S} *such that* $\min\{W_S\} = \min(I_{W_S})$.
A Polytime linear *cost function* W_S *is a linear cost function with the corresponding integer linear program* I_{W_S}*, which has a polynomial number of inequalities and a polynomial number of variables with respect to* $|S|$ *and* $\max_{x_i \in S}\{|D(x_i)|\}$.

One example of polytime linear cost functions is W_SLIDINGSUM^{dec}.

Example 4.2 *The* W_SLIDINGSUM^{dec} *cost function is polytime linear. It can be expressed as an integer linear program* I_{W_S} *defined as*:

$$\begin{array}{cc} \min \sum_{p_j \in \Pi}(L_j + U_j) & s.t. \\ l_j \leq \sum_{x_h \in S_j} \sum_{v \in D(h)} v \cdot c_{x_h,v} - L_j + U_j \leq u_j & \forall p_j \in \Pi \\ \sum_{v \in D(x_i)} c_{x_i,v} = 1 & \forall i = 1 \ldots n \\ L_j \geq 0, U_j \geq 0 & \forall p_j \in \Pi \\ 0 \leq c_{x_i,v} \leq 1 & \forall x_i \in S, v \in D(x_i) \end{array}$$

Define $d_{max} = \max_{x_i \in S}\{|D(x_i)|\}$. *The integer linear program* I_{W_S} *uses* $d_{max} \cdot |S| + 2 \cdot |\Pi|$ *variables and* $3 \cdot |\Pi| + d_{max} \cdot (|S| + 1)$ *inequalities.*

However, enforcing consistencies in WCSPs requires projections and extensions. The cost function may not be polytime linear after such operations. Lee and Leung [30] propose the notion of $\mathcal{T}$ projection-safety which is a requirement for a cost function to preserve the property $\mathcal{T}$ after projections and extensions. We further study the case when $\mathcal{T}$ is *polynomial linearity*.

Definition 4.3 *A* Polytime Linear Projection-Safe (PLPS) *cost function* W_S *is a polytime linear cost function such that for all* W'_S *derived by a series of projections from and/or extensions to* W_S, W'_S *is polytime linear.*

In the following, we first define a set of conditions $\mathbb{PL}$, and show that $\mathbb{PL}$ is sufficient for polytime linear projection-safety.

Definition 4.4 *A cost function* W_S *satisfies* $\mathbb{PL}$ *if*:

1. W_S *is polytime linear with the corresponding integer linear program* I_{W_S},
2. *there exists a surjective function* Λ' *mapping each feasible solution* $\gamma_{I_{W_S}}$ *of* I_{W_S} *to each tuple* $\ell \in \mathcal{L}(S)$, *and;*
3. *for each value* $v \in D(x_i)$ *in each variable* $x_i \in S$, *there exists an injection mapping an assignment* $\{x_i \mapsto v\}$ *to a 0-1 variable* $c_{x_i,v}$ *in* I_{W_S} *such that if* $\ell = \Lambda'(\gamma_{I_{W_S}})$ *for a feasible solution* $\gamma_{I_{W_S}}$ *in* I_{W_S} *and a tuple* $\ell \in \mathcal{L}(S)$, *whenever* $\ell[x_i] = v$, $\gamma_{I_{W_S}}[c_{x_i,v}] = 1$; *whenever* $\ell[x_i] \neq v$, $\gamma_{I_{W_S}}[c_{x_i,v}] = 0$.

Lemma 4.5 *Suppose we are given* W_S *satisfying* $\mathbb{PL}$, $i \in S$ *and* $v \in D(i)$, *and that* W'_S *is obtained by projecting* α *from* W_S *to* $W_i(v)$, *or extending* α *from* $W_i(v)$ *to* W_S. *The resultant cost function* W'_S *satisfies* $\mathbb{PL}$.

Proof We only prove the part for projection, while the part for extension is similar. Assume W_S is a PLPS cost function and I_{W_S} is the corresponding integer linear program of W_S. We first consider the part concerning projection, *i.e.* W'_S is defined as:

$$W'_S(\ell) = \begin{cases} W_S(\ell) \ominus \alpha & \text{if } \ell[x_i] = v \\ W_S(\ell) & \text{otherwise} \end{cases}$$

We show that W'_S is also a polytime linear cost function (condition 1). After projection, we can construct a new integer linear program $I_{W'_S}$ from I_{W_S} by adding an additional term $-\alpha c_{i,v}$ to the objective function of I_{W_S}. The resulting integer linear program $I_{W'_S}$ corresponds to W'_S, since:

$$\begin{aligned} \min(I_{W'_S}) &= \min(I_{W_S}) \ominus \alpha c_{i,v} \\ &= \min\{W_S\} \ominus \alpha c_{i,v} \\ &= \begin{cases} \min\{W_S\} \ominus \alpha & \text{if } c_{i,v} = 1 \\ \min\{W_S\} & \text{if } c_{i,v} = 0 \end{cases} \\ &= \min\{W'_S\}. \end{aligned}$$

Thus, W'_S is linear with the corresponding integer linear program $I_{W'_S}$ and satisfies condition 1. Moreover, since $I_{W'_S}$ has the same set of variables and linear inequalities as I_{W_S} has, W'_S also satisfies conditions 2 and 3. The result follows. □

Theorem 4.6 *If a global cost function W_S satisfies $\mathbb{PL}$, it is a PLPS cost function.*

Proof Initially, W_S satisfies $\mathbb{PL}$. From Definition 4.4, W_S is polytime linear. Assume W'_S is obtained from a series of projection and extension operations. By Lemma 4.5, W'_S still satisfies $\mathbb{PL}$ and thus W'_S remains polytime linear. The result follows. □

Theorem 4.6 gives sufficient conditions for a global cost function to be PLPS. In addition, the proof of Lemma 4.5 demonstrates a general procedure for performing projections and extensions on PLPS cost functions.

As an example, we show that the W_SLIDINGSUM^{dec} cost function is a PLPS cost function.

Theorem 4.7 *The cost function* W_SLIDINGSUM^{dec} *is PLPS.*

Proof By Theorem 4.2, the W_SLIDINGSUM^{dec} cost function is polytime linear with the corresponding integer linear program I_{W_S} (conditions 1 and 2). For condition 3, we observe in I_{W_S} that, if $x_i = d$, $c_{x_i,d} = 1$; otherwise $c_{x_i,d} = 0$. By Theorem 4.6, the W_SLIDINGSUM^{dec} cost function is PLPS. □

Consider the following WCSP $P = (\mathcal{X}, \mathcal{D}, \{W_S\}, k)$:

- $\mathcal{X} = \{x_1, x_2, x_3\}$;
- $D(x_1) = D(x_2) = D(x_3) = \{1, 2, 3\}$;
- $W_S =$ W_SLIDINGSUM$^{dec}(\{x_1, x_2, x_3\}, \{p_1, p_2\})$, where $p_1 = \langle 3, 4, \{x_1, x_2\}\rangle$ and $p_2 = \langle 4, 5, \{x_2, x_3\}\rangle$

The corresponding integer linear program of W_S is:

$$\begin{gathered}
\min\ L_1 + U_1 + L_2 + U_2\ s.t. \\
3 \le c_{x_1,1} + 2c_{x_1,2} + 3c_{x_1,3} + c_{x_2,1} + 2c_{x_2,2} + 3c_{x_2,3} - L_1 + U_1 \le 4 \\
4 \le c_{x_2,1} + 2c_{x_2,2} + 3c_{x_2,3} + c_{x_3,1} + 2c_{x_3,2} + 3c_{x_3,3} - L_2 + U_2 \le 5 \\
c_{x_1,1} + c_{x_1,2} + c_{x_1,3} = 1 \\
c_{x_2,1} + c_{x_2,2} + c_{x_2,3} = 1 \\
c_{x_3,1} + c_{x_3,2} + c_{x_3,3} = 1 \\
L_1 \ge 0, U_1 \ge 0, L_2 \ge 0, U_2 \ge 0 \\
0 \le c_{x_i,d} \le 1 \quad \forall x_i \in S, d \in D(x_i)
\end{gathered}$$

Suppose a cost of 2 is projected from W_S to W_{x_1} with respect to $x_1 = 1$, such that W_S becomes W'_S. The corresponding integer linear program of W'_S can be constructed from that of W_S by adding a term to its objective function as underlined below. The other parts of the corresponding integer linear program of W'_S are the same as that

of W_S and so W'_S is also PLPS. The corresponding integer linear program of W'_S becomes:

$$\begin{gathered}\min\ L_1 + U_1 + L_2 + U_2 \underline{-2c_{x_1}}\ s.t.\\ 3 \leq c_{x_1,1} + 2c_{x_1,2} + 3c_{x_1,3} + c_{x_2,1} + 2c_{x_2,2} + 3c_{x_2,3} - L_1 + U_1 \leq 4\\ 4 \leq c_{x_2,1} + 2c_{x_2,2} + 3c_{x_2,3} + c_{x_3,1} + 2c_{x_3,2} + 3c_{x_3,3} - L_2 + U_2 \leq 5\\ c_{x_1,1} + c_{x_1,2} + c_{x_1,3} = 1\\ c_{x_2,1} + c_{x_2,2} + c_{x_2,3} = 1\\ c_{x_3,1} + c_{x_3,2} + c_{x_3,3} = 1\\ L_1 \geq 0, U_1 \geq 0, L_2 \geq 0, U_2 \geq 0\\ 0 \leq c_{x_i,d} \leq 1 \quad \forall x_i \in S, d \in D(x_i)\end{gathered}$$

Solving integer linear programs is NP-hard in general, but linear relaxation allows the minimum of the corresponding integer linear programs of PLPS cost functions to be approximated in polynomial time. Accordingly, relaxed consistency notions for WCSPs can be defined. They are weaker but can be enforced more efficiently.

Readers are referred to the Appendix for more examples of PLPS cost functions.

5 Relaxed Consistency Notions

Given a PLPS cost function W_S and its corresponding integer linear program I_{W_S}, we define $\mathrm{relaxed\_min}(I_{W_S})$ to be the minimum of I_{W_S} with linear relaxation. We have the following theorem according to the properties of linear relaxation.

Lemma 5.1 *[49] Given an integer linear program I_{W_S}, the following always holds:*

$$\mathrm{relaxed\_min}(I_{W_S}) \leq \lceil \mathrm{relaxed\_min}(I_{W_S}) \rceil \leq \min(I_{W_S})$$

The pair of $\lceil\ \rceil$ symbols represents the ceiling function, where $\lceil x \rceil$ gives the smallest integer not less than x.

Proof It is straightforward to see that $\mathrm{relaxed\_min}(I_{W_S}) \leq \min(I_{W_S})$ with the properties of linear relaxation. On the other hand, $\min(I_{W_S})$ is defined as an integer in WCSPs, and so the smallest integer not less than $\mathrm{relaxed\_min}(I_{W_S})$ must be smaller than or equal to $\min(I_{W_S})$. The result follows. □

The definition of relaxed consistencies is based on the *approximated minimum* of cost functions. The *approximated minimum* of a PLPS cost function is defined as follows.

Definition 5.2 *Given a PLPS cost function W_S and its corresponding integer linear program I_{W_S}, we define the* approximated minimum *of W_S, written as* $\mathrm{approx\_min}\{W_S\}$, *to be:*

$$\mathrm{approx\_min}\{W_S\} = \mathrm{relaxed\_min}(I_{W_S})$$

Suppose we are given a PLPS cost function W_S. By definition, $\min(I_{W_S}) = \min\{W_S\}$. The next lemma is a re-statement of Lemma 5.1.

Lemma 5.3 *Given a PLPS cost function W_S and its corresponding integer linear program I_{W_S}, we have the following inequalities:*

$$\text{approx\_min}\{W_S\} \leq \lceil \text{approx\_min}\{W_S\} \rceil \leq \min\{W_S\}$$

The definition of approximated minimum leads to the definition of a relaxed version of GAC* [18, 28, 30] called *relaxed GAC**.

Definition 5.4 *Suppose we are given a WCSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C}, k)$, a cost function $W_S \in \mathcal{C}^+$ and a variable $x_i \in S$.*

- *A variable $x_i \in S$ is* relaxed GAC* *with respect to W_S iff:*
  - *x_i is NC*, and;*
  - *for each value $v_i \in D(x_i)$,*

$$\text{approx\_min}\{W_S(\ell) \mid \ell \in \mathcal{L}(S) \wedge \ell[x_i] = v_i\} \leq 0$$

- *A WCSP is* relaxed GAC* *iff all variables are relaxed GAC* with respect to all cost functions in $\{W_S \mid W_S \in \mathcal{C}^+ \wedge x_i \in S\}$.*

Unlike GAC*, supporting tuples are not required in relaxed GAC* since we cannot guarantee the existence of a tuple ℓ' such that $W_S(\ell')$ is equal to the approximated minimum. We only ensure that the approximated minimum is always smaller than or equal to the minimum cost $\min\{W_S(\ell)\}$. On the other hand, suppose a cost $\alpha = \min\{W_S(\ell)\}$ is projected from the cost function $W_S(\ell)$, where $\min\{W'_S(\ell)\}$ is greater than 0 after projecting α in enforcing GAC*, it is possible for $\text{relaxed\_min}(I_{W'_S(\ell)}) \leq 0$. So relaxed GAC* allows $\text{approx\_}\min W_S(\ell)$ to be less than or equal to 0.

To compare the strength of GAC* and relaxed GAC*, we adopt the following definition from Lee and Leung [30].

Definition 5.5 *[30] Given a problem P representable by two models $\phi(P)$ and $\psi(P)$, we say that a consistency Φ on $\phi(P)$ is* strictly stronger *than another consistency Ψ on $\psi(P)$ iff $\psi(P)$ is Ψ whenever $\phi(P)$ is Φ, but not vice versa.*

By Lemma 5.3, we immediately have the following theorem.

Theorem 5.6 *GAC* [18, 28, 30] is strictly stronger than relaxed GAC*.*

The procedure `enforceRelaxedGAC*()` in Algorithm 2 enforces relaxed GAC* for a WCSP $(\mathcal{X}, \mathcal{D}, \mathcal{C}, k)$ and is a simple adaptation of `enforceGAC*()` in Algorithm 4 given by Lee and Leung [30]. The differences (compared to [30]) between the procedures of enforcing GAC* and relaxed GAC* are underlined.

The procedure `enforceRelaxedGAC*()` in Algorithm 2 is correct and must terminate. Its complexity can be analyzed by abstracting the worst-case time complexity of `projectApproxMinCost()` as $f_{approxMin}$, which mainly consists of solving a linear program. The ellipsoid method can be used to solve linear programs which can be characterized by polynomial complexity. Using an argument similar to the proof of Larrosa and Schiex's [26] Theorems 12 and 21, the complexity can be stated as follows.

```
1  Procedure enforceRelaxedGAC*()
2      Q := X;
3      while Q ≠ ∅ do
4          x_j := pop(Q);
5          flag := false;
6          foreach W_S s.t. {x_j} ⊂ S do
7              foreach x_i ∈ S \ {x_j} do
8                  flag := flag ∨ projectApproxMinCost(W_S, x_i);
9                  pruneVal(x_i);
10                 if D(x_i) is changed then Q := Q ∪ {x_i};
11         if flag then
12             foreach x_i ∈ X do
13                 pruneVal(x_i);
14                 if D(x_i) is changed then Q := Q ∪ {x_i};
15
16 Function projectApproxMinCost(W_S, x_i):Boolean
17     flag := false;
18     foreach v ∈ D(x_i) do
19         α := max(⌈approx_min{W_S(ℓ)|ℓ ∈ L(S) ∧ ℓ[x_i] = v}⌉, 0);
20         if W_i(v) = 0 ∧ α > 0 then flag := true;
21         W_i(v) := W_i(v) ⊕ α;
22         foreach ℓ ∈ L(S) s.t. ℓ[x_i] = v do W_S(ℓ) := W_S(ℓ) ⊖ α;
23     unaryProject(x_i);
24     return flag;
25
```

Algorithm 2: Enforcing relaxed GAC* for a WCSP

Theorem 5.7 *The procedure* `enforceRelaxedGAC*()` *must terminate and has a time complexity of* $O(r^2ed_{approxMin} + n^2d^2)$*, where* r *is the maximum arity of all cost functions,* d *is the maximum domain size,* $e = |W_S|$ *and* $n = |\mathcal{X}|$*.*

Proof The while loop at line 3 iterates at most $O(nd)$ times. A variable is pushed into the queue $\mathcal{Q}$ only if the domain of a variable has changed at line 10 or line 14, which happens at most nd times. So, the procedure must terminate. We consider the time complexity at each iteration. Line 8 executes at most $O(r \cdot |N(j)|)$ times, where $N(j)$ is the set of cost functions restricting x_j. The overall time complexity is $O(r^2ed_{approxMin} + n^2d^2)$ Thus, it must terminate. □

Corollary 5.8 *The procedure* `enforceRelaxedGAC*()` *must terminate. The resultant WCSP is relaxed GAC*, and equivalent to the original WCSP.*

Algorithm 2 can also transform any WCSP to an equivalent one which is relaxed GAC*, the proof is similar to that of enforcing GAC* [30].

Note that $\text{approx\_min}\{W_S\}$ does not always return an integer. The value $\alpha' = \max(\lceil \text{approx\_min}\{W_S(\ell) | \ell \in \mathcal{L}(S) \wedge \ell[x_i] = v\} \rceil, 0)$ is used instead. The correctness of using a' in projections can be shown as follows.

We are given that I_{W_S} is an integer linear program corresponding to a PLPS cost function W_S and there exists a cost $\alpha = \min\{W_S\}$ to be projected in enforcing GAC*. Suppose W''_S is the resultant cost function after projecting $\alpha' = \lceil \text{approx\_min}\{W_S\} \rceil$ from W_S, we have the following theorem.

Theorem 5.9 $\min\{W''_S\} \geq 0$.

Proof First of all, $\alpha = \min\{W_S\} = \min(I_{W_S})$. Suppose $\min\{W'_S\}$ is the resultant cost function after projecting α from W_S in enforcing GAC*, $\min\{W'_S\} \geq 0$. Solving I_{W_S} by linear relaxation obtains an approximated minimum $\text{relaxed\_min}(I_{W_S}) = \text{approx\_min}\{W_S\}$ to be projected. By Lemma 5.1, $\alpha' = \lceil \text{relaxed\_min}(I_{W_S}) \rceil \leq \min(I_{W_S}) = \alpha$. Thus, $\min\{W''_S\} \geq \min\{W'_S\} \geq 0$. The result follows. □

We can also define the relaxed version of FDGAC* [29, 30] and weak EDGAC* [29, 30], called *relaxed FDGAC** and *relaxed weak EDGAC** respectively.

Definition 5.10 *Suppose we are given a WCSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C}, k)$, and a cost function $W_S \in \mathcal{C}^+$ and a variable $x_i \in S$.*

- *A variable $x_i \in S$ is* relaxed DGAC* *with respect to W_S if*:
  - *x_i is NC*, and;*
  - *for each value $v_i \in D(x_i)$,*

$$\text{approx\_min}\{W_S(\ell) \oplus \bigoplus_{x_j \in S \wedge j > i} W_j(\ell[x_j]) \,|\, \ell \in \mathcal{L}(S) \wedge \ell[x_i] = v_i\} \leq 0$$

- *A WCSP is* relaxed FDGAC* *iff all variables are relaxed GAC* and relaxed DGAC* with respect to all cost functions in $\{W_S \mid W_S \in \mathcal{C}^+ \wedge x_i \in S\}$.*

Similar to weak EDGAC* [29, 30], the definition of relaxed weak EDGAC* is based on *cost-providing partitions* [29, 30].

Definition 5.11 *[29, 30] Given a WCSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C}, k)$, we define a* cost-providing partition $\mathcal{B}_{x_i}$ *for a variable $x_i \in \mathcal{X}$ to be a set of sets $\{B_{x_i,W_S} \mid x_i \in S\}$ such that:*

- *$|\mathcal{B}_{x_i}|$ is the number of cost functions whose scopes include x_i;*
- *$B_{x_i,W_S} \subseteq S$;*
- *$B_{x_i,W_{S_j}} \cap B_{x_i,W_{S_k}} = \varnothing$ for any two different constraints $W_{S_k}, W_{S_j} \in \mathcal{C}^+$, and;*
- *$\bigcup_{B_{x_i,W_S} \in \mathcal{B}_{x_i}} B_{x_i,W_S} = (\bigcup_{W_S \in \mathcal{C}^+ \wedge x_i \in S} S) \setminus \{x_i\}$.*

Definition 5.12 *Suppose we are given a WCSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C}, k)$, and a variable $x_i \in S$ with the associated cost-providing partition $\mathcal{B}_{x_i}$.*

- *A variable $x_i \in S$ is* relaxed weak EGAC* *if*:

– x_i *is NC*, and;*
– *there exists a value* $v_i \in D(x_i)$ *such that:*

$$\text{approx\_min}\{\bigoplus_{x_i \in S} W_S(\ell) \oplus \bigoplus_{x_j \in B_{x_i,W_S}} W_j(\ell[x_j]) | \ell \in \mathcal{L}(S) \wedge \ell[x_i] = v_i\} \leq 0$$

- *A WCSP* is relaxed weak EDGAC* *iff it is relaxed FDGAC* and all variables are relaxed weak EDGAC*.*

By Lemma 5.3, we immediately have the following theorems.

Theorem 5.13 *FDGAC* [29, 30] is strictly stronger than relaxed FDGAC*.*

Theorem 5.14 *Weak EDGAC* [29, 30] is strictly stronger than relaxed weak EDGAC*.*

The procedures of enforcing relaxed FDGAC* and relaxed weak EDGAC* are similar to those of enforcing FDGAC* [28, 30] and weak EDGAC* [29, 30] respectively. The `findSupport()` function in the `findFullSupport()` function in algorithm 5 from Lee and Leung [30] is replaced by the `projectApproxMinCost()` function, which is similar to that of enforcing relaxed GAC*.

6 Conjoining PLPS Cost Functions

If two constraints or cost functions share more than one variable, they *overlap*. In the rest of the paper, we consider conjunctions of overlapping cost functions.

Definition 6.1 *Given two cost functions* W_{S_1} *and* W_{S_2}*, we define* $W_{S_1} \oplus W_{S_2}$ *to be their* conjunction*, i.e. for each tuple* $\ell \in \mathcal{L}(S_1 \cup S_2)$*:*

$$(W_{S_1} \oplus W_{S_2})(\ell) = W_{S_1}(\ell[S_1]) \oplus W_{S_2}(\ell[S_2])$$

In general, enforcing a consistency on individual cost functions may not imply the same consistency on their conjunction. An example is given by Bessière *et al.* [10]: enforcing GAC on two overlapping ALLDIFF [27] constraints does not imply GAC on their conjunction. It is easy to check that a similar result also holds for cost functions. By discovering extra pruning opportunities, propagating on conjunctions of cost functions may reduce more search space than propagating on individual cost functions can.

This issue is especially important for two reasons. First, many global cost functions can be decomposed into conjunctions of overlapping and simpler (global) cost functions. Second, every PLPS cost function has an associated integer linear program. PLPS cost functions can be conjoined together easily by combining their corresponding integer linear programs in a straightforward manner.

Definition 6.2 *Given two integer linear programs* $I_{W_{S_1}}$ *and* $I_{W_{S_2}}$*, we define* $I_{W_{S_1}} \wedge I_{W_{S_2}}$ *to be their combination by taking the union of their linear inequalities and adding up their objective functions.*

We are demonstrating that it is worthwhile computationally to propagate on conjunctions of PLPS cost functions using relaxed consistencies.

The following theorem ensures that conjunctions of PLPS cost functions are PLPS.

Lemma 6.3 *Suppose W_{S_1} and W_{S_2} are PLPS cost functions. The conjunction $W_{conj} \equiv W_{S_1} \oplus W_{S_2}$ is also PLPS.*

Proof Suppose W_{S_1} and W_{S_2} have their corresponding integer linear program $I_{W_{S_1}}$ and $I_{W_{S_2}}$ respectively. The integer linear program $I_{W_{conj}}$ for W_{conj} can simply be formed by $I_{W_{conj}} \equiv I_{W_{S_1}} \wedge I_{W_{S_2}}$. It is easy to check that W_{conj} satisfies the conditions in Definition 4.4. □

An immediate question is whether a conjunction of PLPS cost functions always gives a stronger bound than using the individual PLPS cost functions, given that the same level of consistency is maintained.

Definition 6.4 *Given a WCSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C}, k)$, we define the* conjoined WCSP P_{conj} *to be a tuple $(\mathcal{X}, \mathcal{D}, \mathcal{C}_{conj}, k)$, where $\mathcal{C}_{conj} = \{W_{conj}\}$ and $W_{conj} \equiv \bigwedge_{W_S \in \mathcal{C}} W_S$ with the corresponding scope $S_{conj} = \bigcup_{W_S \in \mathcal{C}} S$.*

We show that GAC*, FDGAC* and weak EDGAC* on P_{conj} are strictly stronger than their counterparts on P respectively by the following theorem.

Theorem 6.5 *Suppose α-consistency is one of GAC*, FDGAC*, and weak EDGAC*. Then α-consistency on P_{conj} is strictly stronger than α-consistency on P.*

Proof We prove the part for GAC*. The proofs for the other consistencies are similar.

Assume P_{conj} is GAC*, but P is not GAC*. There exists a variable $x_i \in \mathcal{X}$ with a value $a \in D(x_i)$ and a cost function $W_S \in \mathcal{C}$ in P such that $\min\{W_S(\ell) \,|\, \ell[x_i] = a \wedge \ell \in \mathcal{L}(S)\} > 0$. Therefore,

$$\begin{array}{rl} & \min\{W_{conj} \mid \ell[x_i] = a \wedge \ell \in \mathcal{L}(S_{conj})\} \\ \geq & \bigoplus_{W_S \in \mathcal{C}} \min\{W_S(\ell) \mid \ell[x_i] = a \wedge \ell \in \mathcal{L}(S)\} > 0 \end{array}$$

So no simple support exists for a and x_i cannot be GAC* with respect to W_{conj}.

The converse is not true. A counter-example is the W_ALLDIFF$^{var}(S)$ cost function [39] , which returns the minimum number of variable assignments in S that are needed to be changed so that S contains only distinct values.
Consider W_{S_1} = W_ALLDIFF$^{var}\{x_1, x_2, x_3\}$ and W_{S_2} = W_ALLDIFF$^{var}\{x_2, x_3, x_4\}$, where $D(x_1) = \{a, b\}$, $D(x_2) = D(x_3) = \{a, b, c\}$ and $D(x_4) = \{b, c\}$. It is easy to check that W_{S_1} and W_{S_2} are GAC*, but $W_{S_1} \oplus W_{S_2}$ is not since the minimum when $x_1 = a$ is 1. The result follows. □

When exact consistencies are replaced by their relaxed versions, the result similar to that of Theorem 6.5 does not hold unfortunately. We use an example to explain.

Consider $\mathcal{C}_{PLPS} = \{W_{S_1}, W_{S_2}\}$, where W_{S_1} and W_{S_2} are both PLPS. Suppose P_{conj} is relaxed GAC*. For a given $x_i \in S_1 \cap S_2$ and $a \in D(x_i)$:

$$\begin{aligned} 0 &\geq \text{approx\_min}\{W_{conj} \mid \ell[x_i] = a \wedge \ell \in \mathcal{L}(S_{conj})\} && (1) \\ &\geq \text{approx\_min}\{W_{S_1}(\ell) \mid \ell[x_i] = a \wedge \ell \in \mathcal{L}(S_1)\} \oplus \\ &\quad \text{approx\_min}\{W_{S_2}(\ell) \mid \ell[x_i] = a \wedge \ell \in \mathcal{L}(S_2)\} && (2) \end{aligned}$$

Since the approximated minimum can be negative, we cannot conclude from equation (2) that both W_{S_1} and W_{S_2} are relaxed GAC*. One can lead to positive approximated minimum while another one leads to negative. However, this peculiar bad situation does not happen often in practice and we will demonstrate that it is worthwhile to propagate on conjunctions instead of individual cost functions in the following experiment. Additional experimental results can be found in Section 9.

In our experiments, the consistencies GAC*, FDGAC*, weak EDGAC*, and their relaxed versions are implemented in Toulbar2 v0.9. IBM ILOG CPLEX Optimizer 12.2 is used as the (integer) linear programs solvers in Toulbar2.

Variables with smaller domains and values with lower unary costs are assigned first. The experiments are conducted on an Intel Core2 Duo E7400 (2 x 2.80GHz) machine with 4GB RAM. In each benchmark, we use different parameter settings to construct different instances and 10 random cases are generated with each parameter setting.

The average number of backtrack (bt) and the average runtime in seconds (time) for solved cases are reported. The runtime includes the CPU time used by both Toulbar2 and CPLEX. Next to the runtime, we also report separately in brackets the CPU time used by CPLEX. The best result is highlighted in bold.

In this experiment, we use the Generalized Car Sequencing Problem (Generalizing prob001 in CSPLib), which aims to find a sequence for n cars of different types to be built. Each type of cars $u \in U$ has a specific set of options $I_u \subseteq I$ to equip the car with. To equip an option $i \in I$ on a specific type of cars $u \in U$, a cost $c_{u,i}$ is required. Each assembly line is allowed to spend a maximum of m_i cost on each option i for every s_i cars in total.

We model this problem with $n = \{x_j \mid j = 1, \ldots, n\}$ variables with domains u. Each variable x_j represents the j^{th} car to be built in the sequence. A GCC constraint [41] is used to ensure that the correct number of cars of each type is assembled according to the plan. The W_SLIDINGSUM^{dec} cost functions, which are PLPS, are used to ensure the restrictions for each assembly line. We further soften the problems by assigning a random unary cost from 0 to 9 to each value in the domain of each variable, and replacing the hard global constraints with their soft variants. We assume that there are preferences. For example, an assembly line may prefer the same type of cars to be assembled consecutively. Such preferences can be modeled by table cost functions. We fix $|I| = 5$ and $u = n/2$ and use instances with different n in our experiments.

As each instance consists of (only) PLPS cost functions, we can compare 2 different models respectively, which are using: (1) individual PLPS cost functions, and (2) a single PLPS cost function formed by conjoining all PLPS cost functions.

Results are shown in Table 1. First and foremost, applying relaxed consistencies on the conjoined PLPS cost functions is substantially better than applying the correspond-

Table 1: The generalized car sequencing problem using W_SLIDINGSUM^{dec}

(1) Modeling with PLPS cost functions						
n	relaxed GAC*		relaxed FDGAC*		relaxed weak EDGAC*	
	bt	time (CPLEX)	bt	time (CPLEX)	bt	time (CPLEX)
10	805.7	135.78 (42.74)	7.4	2.37 (1.23)	6.9	1.94 (0.97)
12	*	*	15.1	7.17 (3.84)	14.4	4.99 (2.49)
14	*	*	34.7	69.42 (39.62)	29.0	41.85 (23.53)
16	*	*	*	*	*	*
18	*	*	*	*	*	*
(2) Modeling with conjoined PLPS cost functions						
n	relaxed GAC*		relaxed FDGAC*		relaxed weak EDGAC*	
	bt	time (CPLEX)	bt	time (CPLEX)	bt	time (CPLEX)
10	21.8	**0.43 (0.18)**	5.0	**0.43 (0.25)**	**2.6**	0.50 (0.32)
12	31.1	**0.74 (0.4)**	12.2	0.95 (0.61)	**6.8**	1.14 (0.77)
14	114.0	3.33 (2.00)	20.6	**2.02 (1.39)**	**7.7**	2.22 (1.64)
16	2297.5	108.25 (73.76)	123.2	15.16 (10.95)	**58.1**	**11.65 (8.58)**
18	*	*	746.4	78.24 (52.14)	**491.5**	**68.89 (44.94)**

ing consistencies on the PLPS cost functions individually both in terms of pruning and runtime. Analyzing the cases with conjoined cost functions further, we observe that in instances with smaller size, relaxed FDGAC* and relaxed weak EDGAC* do not infer a much better bound than relaxed GAC*. The reduction in search space does not compensate for the overhead, and the simpler and less costly relaxed GAC* gives better results in such instances. On the other hand, relaxed weak EDGAC* performs better in terms of the number of backtracks and run-time in bigger instances. Enforcement of the stronger consistencies becomes worthwhile in such cases.

7 Polytime Integral Linear Projection-Safety

The experimental results show that it is often beneficial to conjoin the PLPS cost functions in terms of runtime. In this section, we define a subclass of PLPS cost functions called *Polytime Integral Linear Projection-Safe* (*PILPS*) cost functions with which we can give exact pruning characteristic of relaxed consistencies on conjunctions, while improvements in pruning cannot be guaranteed theoretically for conjunctions of PLPS cost functions.

Definition 7.1 *A* polytime integral linear *cost function W_S is a polytime linear cost function such that the linear relaxation of its corresponding integer linear program I_{W_S} always gives an integral minimum.*

An immediate observation is that the exact minimum of a polytime integral linear cost function can be obtained by solving the linear relaxation of their corresponding integer linear programs, which can be done in polynomial time [49].

Lemma 7.2 *If W_S is a polytime integral linear cost function,*

$$\min\{W_S\} = \text{approx\_min}\{W_S\}$$

Theorem 7.3 *Minimum computation of polytime integral linear cost functions is polynomial.*

Proof Follows directly from Lemma 7.2.

Recall the notion of $\mathcal{T}$ projection-safety. In addition to flow-basedness [28, 30] and polytime linearity [32], polytime integral linearity is another good property $\mathcal{T}$ that should be maintained across projections/extensions. Therefore, it makes sense to require cost functions to be *Polytime Integral Linear Projection-Safe* (*PILPS*), which is a special subclass of PLPS cost functions.

Definition 7.4 *A* Polytime Integral Linear Projection-Safe (PILPS) *cost function W_S is a polytime integral linear cost function such that for all W'_S derived from a series of projections from and/or extensions to W_S, W'_S is polytime integral linear.*

We give a possible sufficient condition to identify PILPS cost functions.

Theorem 7.5 *A cost function W_S is PILPS if it satisfies the following conditions:*

1. *W_S satisfies the conditions given in Definition 4.4 with the associated linear program I_{W_S}, and;*
2. *I_{W_S} is totally dual integral or the associated matrix of I_{W_S} is totally unimodular.*

Proof By Theorem 4.6, W_S is PLPS. Moreover, if a linear program is *totally dual integral* or its associated matrix is *totally unimodular*, its minimum must be integral [36]. The result follows. □

One example of PILPS cost functions is W_ALLDIFF^{var} [39].

Theorem 7.6 *The cost function* W_ALLDIFF^{var} *is PILPS.*

Proof Assume $S = \{x_1, \ldots, x_n\}$. The cost function W_ALLDIFF$^{var}(S)$ can be represented by the following integer linear program I_{W_S} with associated totally unimodular matrix:

$$\begin{array}{cc} \min \sum_{a \in D_S} u_a & s.t. \\ \sum_{i=1}^{n} c_{x_i,a} + u_a \leq 1 & \forall a \in D_S \\ u_a \geq 0 & \forall a \in D_S \\ 0 \leq c_{x_i,a} \leq 1 & \forall a \in D_S, 1 \leq i \leq n \end{array}$$

where $D_S = \bigcup_n^{i=1} D(x_i)$. The minimum of $I_{W_S} = \min\{\text{W\_ALLDIFF}^{var}(S)\}$.

Flow-based projection-safe cost functions [28, 30] are PILPS functions, which we state in the following theorem.

Theorem 7.7 *Flow-based projection-safe cost functions [28, 30] are PILPS.*

Proof Every flow-based projection-safe cost function has a corresponding minimum cost network flow problem, which in turn has a corresponding integer linear program with a totally unimodular matrix [36]. The cost function, the flow problem, and the integer linear program share the same minimum. Since the integer linear program always has integral solutions when solved by linear relaxation, the result follows. □

Corollary 7.8 *The following cost functions are PILPS:*

- W_ALLDIFF^{var} *and* W_ALLDIFF^{dec} *[39];*
- W_GCC^{var} *and* W_GCC^{val} *[48];*
- W_SAME^{var} *[48];*
- W_REGULAR^{var} *[38, 48] and* W_REGULAR^{edit} *[48], and;*
- W_AMONG^{var} *[47].*

Proof All cost functions except W_AMONG^{var} [47] are flow-based projection-safe by Lee and Leung [28, 30]. W_AMONG^{var} [47] can be modeled by W_REGULAR^{var} [38] using deterministic finite automata with the number of states polynomial in n. The results follow by Theorem 7.7.

8 Conjoining PILPS Cost Functions

Polytime integral linear projection-safe cost functions are interesting since their conjunctions are PLPS. By Lemmas 6.3 and 7.2, we have the following corollaries.

Corollary 8.1 *Suppose W_{S_1} and W_{S_2} are PILPS cost functions. The conjunction $W_{conj} \equiv W_{S_1} \oplus W_{S_2}$ is PLPS.*

Corollary 8.2 *Suppose W_S is PILPS, and α-consistency is one of GAC*, FDGAC* and weak EDGAC*. Relaxed α-consistency on W_S is equivalent to α-consistency on W_S.*

In general, it is NP-hard to compute the minimum of the conjunction of overlapping PILPS cost functions, since the conjunction of their corresponding linear programs may not always give an integral minimum [49]. As conjunctions of PILPS cost functions remain PLPS, linear programming techniques allow its approximated minimum to be computed efficiently, and relaxed forms of exact consistencies can thus be enforced. We have the following result when relaxed consistencies are enforced on the conjunction of PILPS cost functions compared to the corresponding (non-relaxed) consistencies enforced on the individual cost functions.

Suppose we are given a WCSP $P_{PILPS} = (\mathcal{X}, \mathcal{D}, \mathcal{C}_{PILPS}, k)$, where each cost function $W_S \in \mathcal{C}_{PILPS}$ is PILPS with corresponding integer linear program I_{W_S}. We show that relaxed (FD)GAC* and relaxed weak EDGAC* on the conjoined WCSP P_{PILPS}, defined as P_{conj}, are strictly stronger than (FD)GAC* and weak EDGAC* on individual cost functions in P_{PILPS} respectively by the following theorem.

Theorem 8.3 *Suppose α-consistency is one of GAC*, FDGAC* and weak EDGAC*.*

1. *α-consistency on P_{conj} is strictly stronger than relaxed α-consistency on P_{conj};*
2. *Relaxed α-consistency on P_{conj} is strictly stronger than α-consistency on P_{PILPS}.*

Proof Since relaxed consistencies are the weaker forms of exact consistencies, Result 1 holds.

We prove Result 2 on GAC*, while those for other consistencies are similar.

Assume P_{conj} is relaxed GAC*, but P_{PILPS} is not GAC*. There exists a variable $x_i \in \mathcal{X}$ with a value $a \in D(x_i)$ and a cost function $W_S \in \mathcal{C}_{PILPS}$ in P_{PILPS} such that $\min\{W_S(\ell) \mid \ell[x_i] = a \wedge \ell \in \mathcal{L}(S)\} > 0$. Since all cost functions $W_S \in \mathcal{C}_{PILPS}$ are PILPS,

$$\begin{array}{rl} & \text{approx\_min}\{W_{conj} \mid \ell[x_i] = a \wedge \ell \in \mathcal{L}(S)\} \\ \geq & \bigoplus_{W_S \in \mathcal{C}_{PILPS}} \text{approx\_min}\{W_S \mid \ell[x_i] = a \wedge \ell \in \mathcal{L}(S)\} \\ = & \bigoplus_{W_S \in \mathcal{C}_{PILPS}} \min\{W_S(\ell) \mid \ell[x_i] = a \wedge \ell \in \mathcal{L}(S)\} > 0 \end{array}$$

Thus, a cannot have any simple support and x_i cannot be relaxed GAC* with respect to W_{conj} in P_{conj}.

The same counter example as in the proof of Theorem 6.5 can show that the converse is not true. The result follows. □

We further give theoretical results to support Theorem 8.3. We show that when compared with P_{conj}, an exponential number of extra steps are needed during the branch-and-bound search for P_{PILPS} to discover the same bound. This example is similar to the one given by Bessière *et al.* [10].

Theorem 8.4 *Suppose α-consistency is one of GAC*, FDGAC* and weak EDGAC*. There exists a WCSP P_{PILPS} such that if we enforce relaxed α-consistency on P_{conj} and α-consistency on P_{PILPS} in branch-and-bound search, an exponential size of search tree is needed to be explored for P_{PILPS} to infer the same minimum cost as in the case of P_{conj}.*

Proof We prove the part for relaxed GAC*. The proofs for the other consistencies are similar. Consider a WCSP $P'_{PILPS} = (X \cup Y \cup Z, \mathcal{D}, \mathcal{C}_{PILPS}, k)$, where,

- $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_{2n}\}$, $Z = \{z_1, \ldots, z_n\}$;
- $D(x_i) = [1, 2n-1]$ for $x_i \in X$, $D(y_i) = [1, 4n-1]$ for $y_i \in Y$, and $D(z_i) = [2n, 4n-1]$ for $z_i \in Z$;
- $\mathcal{C}_{PILPS} = \{\text{W\_ALLDIFF}^{var}(X \cup Y), \text{W\_ALLDIFF}^{var}(Y \cup Z)\}$.

Consider the WCSP $P'_{conj} = (X \cup Y \cup Z, \mathcal{D}, \mathcal{C}_{conj}, k)$ where $\mathcal{C}_{conj} = \{W_{conj}\}$ and $W_{conj} \equiv \text{W\_ALLDIFF}^{var}(X \cup Y) \oplus \text{W\_ALLDIFF}^{var}(Y \cup Z)$. The former gives an increase of $W_\varnothing$ by 1,which can be inferred by enforcing relaxed GAC* on $\mathcal{C}_{conj}$. In P'_{PILPS}, every subset of n or fewer variables has at least $2n-1$ values in their domains, and every subset of $n+1$ to $3n$ variables has $4n-1$ values in their domains. Thus, to infer an increment of $W_\varnothing$ by 1 in P'_{PILPS} by enforcing GAC* on $\mathcal{C}_{PILPS}$, we must instantiate at least $n-1$ variables. □

An immediate application of Theorem 8.3 is on existing global cost functions with polytime minimum computation which we have mentioned above. We note that their dedicated polynomial time algorithms are usually more efficient than linear programming approaches. In many cases, however, the minimum computation for their conjunctions is NP-hard. For example, Bessière *et al.* [10] show the above result on the hard ALLDIFF constraints [27] and it can be generalized to the W_ALLDIFF^{var} [39], W_ALLDIFF^{dec} [39], W_GCC^{var} [48], W_GCC^{val} [48], and W_SAME^{var} [48] cost functions. Régin [42] also shows the above result on the hard AMONG [6] constraints. The result can be generalized to the W_AMONG^{var} [47], W_REGULAR^{var} [38, 48], and W_REGULAR^{edit} [48] cost functions.

Theorem 8.3 suggests that enforcing the relaxed consistencies on the conjunction of such PILPS cost functions can be more efficient and worthwhile than handling them individually using exact consistencies. By propagating the conjunction of PILPS cost functions, extra pruning opportunities can be discovered which may reduce more search space than propagating the individual cost functions.

In the next section, we show by experiments that modeling cost functions as PILPS cost functions and propagating their conjunctions are more efficient then propagating them separately.

9 Experimental Results

In this section, we report experiments on PLPS and PILPS cost functions using the same settings used in Section 6. We demonstrate the efficiency of our framework by comparing performances against flow-based projection-safe cost functions [30] and integer linear programming. Again, the average number of backtracks (bt) and the average runtime in seconds (time) for solved cases are reported. We put an asterisk (*) for an entry if the execution of one of the 10 instances exceeds the timeout of 3600 seconds, and bold the best results for each benchmark.

9.1 Using PLPS cost functions

Our first set of experiments aims at demonstrating the feasibility and efficiency of PLPS cost functions and their conjunctions. We observe that some PLPS cost functions can be decomposed into flow-based projection-safe cost functions, on which we can enforce exact consistencies using flow algorithms [28, 29, 30]. However, flow-based projection-safe cost functions are also PLPS (actually even PILPS) by Theorem 7.7. Exact consistency can also be maintained using our linear programming approach. Last but not least, we can maintain exact consistencies on PLPS cost functions and their conjunctions by the integer programming approach.

We thus consider the following 6 possible scenarios for each benchmark. Each *scenario* includes three components: the model, relaxed versus exact consistencies, and algorithms used.

(a) modeling by individual PLPS cost functions and enforcing relaxed consistencies using a linear programming approach;

(b) modeling by individual PLPS cost functions and enforcing exact consistencies by computing exact minimum using an integer linear programming approach;
(c) modeling by conjoined PLPS cost functions and enforcing relaxed consistencies using a linear programming approach;
(d) modeling by conjoined PLPS cost functions and enforcing exact consistencies by computing exact minimum using an integer linear programming approach;
(e) (if possible) modeling by the flow-based projection-safe cost functions obtained by decomposing the PLPS cost functions used in (a), and enforcing exact consistencies using flow algorithms, and;
(f) (if possible) modeling by the flow-based projection-safe cost functions obtained by decomposing the PLPS cost functions used in (a), and enforcing exact consistencies using linear programming approach.

For each benchmark, we do the following micro comparisons in addition to identifying the best approach. First, we compare scenarios (a) against (b) and (c) against (d) to see the trade-offs between higher pruning of the integer programming approach versus lower overhead of the linear programming approach. Second, when possible, we compare scenarios (e) against (f) to see the overhead of calling a linear programming solver when flow-based algorithms are available to compute exact consistencies in both cases. Third, we compare (a) against (c) and (b) against (d) to see the power of conjunction under the same consistency enforcement. Fourth, we compare (e) against (a) and (c) to see the advantage of the PLPS approach in modeling with more complex cost functions over the flow-based cost functions.

9.1.1 Generalized Car Sequencing Problem

Our first benchmark is the Generalized Car Sequencing Problem described at the end of Section 6. In scenarios (a) and (b), we use individual W_SLIDINGSUM^{dec} cost functions in the model, whereas the W_SLIDINGSUM^{dec} cost functions are conjoined into one cost function in scenarios (c) and (d). In scenarios (e) and (f), W_SLIDINGSUM^{dec} are decomposed into multiple W_SUM^{val} cost functions, which are in turn modeled by the flow-based projection-safe W_REGULAR^{var} [48] cost functions.

Results are shown in Table 2. We observe that enforcing stronger consistencies can give better performance in general. However, in scenario (b), FDGAC* gives better runtime in scenario (d) when $n = 10$. In scenario (c), relaxed GAC* can outperform relaxed FDGAC* and weak EDGAC*, even relaxed GAC* is a weaker consistency. The reason is that the better lower bound inferred by stronger consistencies may not compensate for the extra overhead in comparison to weaker consistencies when the instances are small and relatively easier to solve. However, as the instances grow larger, stronger consistency is more beneficial as the lower bound inferred is good enough to compensate the extra overhead.

Scenario (b) gives slightly better results in terms of the number of backtracks than (a). Exact consistencies are enforced in scenario (b) by the integer programming approach and give better bounds. We observe that enforcing exact consistencies by integer programming requires much more time than enforcing the corresponding relaxed consistencies by linear programming. This becomes the dominating factor of

the runtime, as shown by the CPLEX processing time. However, the bounds inferred after such great effort are not good enough to compensate the extra time needed. As observed, scenario (b) can prune the search space for less than 1.23 times, but the runtime increases up to 23 times more than (a). We also observe that the reduction of backtracks becomes smaller and the increase in runtime becomes greater when the instances grow larger. Similar results can be found when we compare scenarios (c) and (d). As observed, scenario (d) can prune the search space for less than 1.16 times, but the runtime increases up to 20 times more than (c).

Scenarios (e) and (f) are modeled by the W_REGULAR^{var} cost functions. Both models give the same number of backtracks, but scenario (e) outperforms (f) in terms of runtime. According to Corollary 8.2, the linear programming approach has the same strength as flow-based algorithms in terms of the consistencies on PILPS cost functions. Thus, both scenarios give the same number of backtracks in every instance. However, calling a linear programming solver incurs higher overheads than applying the flow algorithm. Thus, scenario (f) takes 1.7-2.3 times longer to solve the instances.

We have already discussed in Section 6 that conjoining PLPS cost functions gives a better lower bound in most cases. Scenario (c) prunes more and requires less time than (a) by conjoining PLPS cost functions. When enforcing (relaxed) weak EDGAC*, scenario (c) prunes up to 4 times more and runs 19 times faster than (a). Similarly, scenario (d) performs better than (b) because (d) also conjoins PLPS cost functions. When enforcing (relaxed) weak EDGAC*, scenario (d) prunes up to 4 times more and runs up to 62 times faster than (b). We also observe that the reduction in search spaces increases when the instances grow larger.

Our approaches (scenarios (a) and (c)) give better performances than the state-of-the-art flow-based approach (scenario (e)) since scenario (e) decomposes the cost functions used in (a) and (c). When enforcing (relaxed) weak EDGAC*, scenario (a) runs up to 2.8 times faster with number of backtracks reduced up to 6.2 times of those in (e). Scenario (c) performs much better, which runs up to 155 times faster with number of backtracks reduced up to 23 times. As more cost functions are used in scenario (e), worse lower bounds are given when consistencies are enforced on the individual decomposed cost functions.

Throughout this benchmark, scenario (c) outperforms all other scenarios. When compared with the flow-based approach of handling global cost functions in WCSP (scenario (e)), scenario (c) can solve the instances up to 20 times faster than the flow-based approach when (relaxed) weak EDGAC* is used. Scenario (c) also outperforms other scenarios using the (integer) linear programming approach. It runs up to 19 times faster than scenario (a), 444 times faster than (b), 18 times faster than (d), and 37 times faster than (f) when (relaxed) weak EDGAC* is enforced. In addition, with our method, large instances like $n = 18$ can be solved within the given time limit, while the flow-based approach cannot.

9.1.2 Magic Series Problem

The Magic Series Problem (prob019 in CSPLib) is to find a sequence of n variables which forms a magic series. A non-empty finite series $S(n) = (s_0, s_1, \ldots, s_n)$ is *m*agic if and only if there are s_i occurrences of $i \in S(n)$ for each integer i ranging

Table 2: The generalized car sequencing problem using W_SLIDINGSUM^{dec}

(a) W_SLIDINGSUM^{dec}						
n	Relaxed GAC*		Relaxed FDGAC*		Relaxed weak EDGAC*	
	bt	time (CPLEX)	bt	time (CPLEX)	bt	time (CPLEX)
10	805.7	135.78 (42.74)	7.4	2.37 (1.23)	6.9	1.94 (0.97)
12	*	*	15.1	7.17 (3.84)	14.4	4.99 (2.49)
14	*	*	34.7	69.42 (39.62)	29.0	41.85 (23.53)
16	*	*	*	*	*	*
18	*	*	*	*	*	*
(b) W_SLIDINGSUM^{dec} with exact consistencies						
n	GAC*		FDGAC*		weak EDGAC*	
	bt	time (CPLEX)	bt	time (CPLEX)	bt	time (CPLEX)
10	724.1	386.60 (246.77)	7.1	14.07 (13.12)	5.6	15.89 (14.68)
12	*	*	13.0	32.46 (28.51)	12.4	31.06 (27.54)
14	*	*	31.7	1107.29 (1078.93)	26.9	986.10 (969.25)
16	*	*	*	*	*	*
18	*	*	*	*	*	*
(c) Conjoined W_SLIDINGSUM^{dec}						
n	Relaxed GAC*		Relaxed FDGAC*		Relaxed weak EDGAC*	
	bt	time (CPLEX)	bt	time (CPLEX)	bt	time (CPLEX)
10	21.8	**0.43 (0.18)**	5.0	**0.43 (0.25)**	2.6	0.50 (0.32)
12	31.1	**0.74 (0.4)**	12.2	0.95 (0.61)	6.8	1.14 (0.77)
14	114.0	3.33 (2.00)	20.6	**2.02 (1.39)**	7.7	2.22 (1.64)
16	2297.5	108.25 (73.76)	123.2	15.16 (10.95)	58.1	**11.65 (8.58)**
18	*	*	746.4	78.24 (52.14)	**491.5**	**68.89 (44.94)**
(d) Conjoined W_SLIDINGSUM^{dec} with exact consistencies						
n	GAC*		FDGAC*		weak EDGAC*	
	bt	time (CPLEX)	bt	time (CPLEX)	bt	time (CPLEX)
10	20.3	4.83 (4.52)	4.8	2.16 (1.96)	**2.5**	2.21 (2.05)
12	29.5	5.13 (4.78)	11.3	4.27 (3.91)	**6.5**	3.72 (3.43)
14	103.6	104.97 (103.60)	18.2	27.33 (26.61)	**7.1**	15.79 (15.03)
16	*	*	106.2	298.70 (294.58)	**54.6**	205.13 (202.46)
18	*	*	*	*	*	*
(e) W_REGULAR^{var} with exact consistencies using flow algorithms						
n	GAC*		FDGAC*		Weak EDGAC*	
	bt	time	bt	time	bt	time
10	2166.0	66.45	8.6	6.80	7.9	4.34
12	*	*	30.4	24.39	28.3	14.37
14	*	*	202.6	57.20	180.0	43.41
16	*	*	*	*	*	*
18	*	*	*	*	*	*
(f) W_REGULAR^{var}						
n	Relaxed GAC*		Relaxed FDGAC*		Relaxed weak EDGAC*	
	bt	time (CPLEX)	bt	time (CPLEX)	bt	time (CPLEX)
10	2166.0	125.58	8.6	12.78 (6.05)	7.9	10.17 (4.07)
12	*	*	30.4	41.67 (20.32)	28.3	31.20 (14.07)
14	*	*	202.6	122.78 (60.61)	180.0	81.04 (34.37)
16	*	*	*	*	*	*
18	*	*	*	*	*	*

from 0 to n. For example, $S(6) = (3, 2, 1, 1, 0, 0, 0)$ is an example of a magic series as there are three 0's, two 1's, a 2, a 3, and no 4, 5, and 6 in $S(6)$.

To model this problem, a sequence of variables $S = \{s_i\}$ is used to represent the series. For each variable, we place a W_EGCC$^{var}(S, s_i)$ cost function (please refer to the Appendix) to restrict the occurrences of each value. We assume preferences over variable values, which are modeled as table cost functions. In scenarios (a) and (b), we use the individual W_EGCC^{var} cost functions in the model, whereas in scenarios (c) and (d), the W_EGCC^{var} cost functions are conjoined into a single cost function.

Definition 9.1 *The cost function* W_AMONG_VAR$^{var}(S, n, V)$ *returns the cost of a tuple* $\ell \in \mathcal{L}(S \cup \{n\})$ *as:*

$$\text{W\_AMONG\_VAR}^{var}(S, n, V)(\ell) = |\ell[n] - occ(V, \ell[S])|$$

where $occ(V, \ell) = |\{i \mid \ell[x_i] \in V\}|$.

In scenarios (e) and (f), W_EGCC^{var} are decomposed into multiple W_AMONG_VAR^{var} cost functions, which are in turn modeled by the flow-based projection-safe W_REGULAR^{var} [38, 48] cost functions.

Results are shown in Table 3. We observe that enforcing stronger consistencies can give better performance. With similar reasoning as in the previous section, scenarios (a) and (c) give higher number of backtracks but require less time to solve than (b) and (d) respectively. Scenario (b) can reduce the search space for less than 1.13 times, but the runtime increases up to 17 times more than (a). Similarly, scenario (d) can reduce the search space for less than 1.13 times, but the runtime increases up to 88 times more than (c).

Again, scenario (f) gives the same number of backtracks as (e) but requires more time to solve, due to the higher overhead in solving linear programs. Scenario (f) takes 12 times longer to solve the instances, and unable to solve instances with $n \geq 15$ within the given time limit.

Conjoining PLPS cost functions also gives better results in this benchmark, as observed in Table 3. Scenarios (c) and (d) give higher number of backtracks but require less time to solve than (b) and (d) respectively. Scenario (c) backtracks up to 4.1 times less and runs 52.5 times faster than (a). Similarly, scenario (d) reduces the number of backtracks up to 3.8 times more, and run 13.5 times faster than (b).

When compared with the flow-based approach (scenario (e)), using conjunctions of cost functions (scenarios (a) and (c)) gives better results when instances are large. Scenario (a) can reduce the number of backtracks up to 3.8 times more than (e). When comparing the runtime, scenario (c) outperforms (e) in all cases. Scenario (c) runs up to 149 times faster with search space reduction up to 16 times more than (e). However, results vary between scenarios (a) and (e). when applying different consistencies with different instance sizes. With GAC*, scenario (a) runs faster than (e), and able to solve larger instances within the given time limit. With stronger consistency like FDGAC* and EDGAC*, scenario (e) runs faster than (a) for smaller instances, due to the extra time required to enforce the consistency. As the instance grows larger, we observe scenario (a) is much more beneficial. With FDGAC*, scenario (a) outperforms (e) when $n \geq 18$. With weak EDGAC*, scenario (a) can outplay (e) when $n \geq 15$. We also observe that scenario (a) runs up to 3.6 times faster than (e) when (a) outplays (e).

Table 3: The magic square problem using W_EGCC^{var}

(a) W_EGCC^{var}

n	Relaxed GAC*		Relaxed FDGAC*		Relaxed weak EDGAC*	
	bt	time (CPLEX)	bt	time (CPLEX)	bt	time (CPLEX)
12	2713.2	229.41 (116.53)	125.7	85.03 (31.29)	87.9	17.20 (7.26)
15	5830.0	359.79 (208.21)	201.3	179.06 (81.28)	120.8	35.66 (16.38)
18	*	*	352.6	218.67 (95.47)	283.9	143.69 (63.91)
21	*	*	*	*	*	*
24	*	*	*	*	*	*

(b) W_EGCC^{var} with exact consistencies

n	GAC*		FDGAC*		weak EDGAC*	
	bt	time (CPLEX)	bt	time (CPLEX)	bt	time (CPLEX)
12	*	*	113.0	792.01 (740.13)	83.1	190.13 (180.34)
15	*	*	186.2	1036.53 (941.22)	110.2	517.90 (498.31)
18	*	*	*	*	251.2	2391.05 (2320.94)
21	*	*	*	*	*	*
24	*	*	*	*	*	*

(c) Conjoined W_EGCC^{var}

n	Relaxed GAC*		Relaxed FDGAC*		Relaxed weak EDGAC*	
	bt	time (CPLEX)	bt	time (CPLEX)	bt	time (CPLEX)
12	357.4	6.69 (2.11)	73.5	1.88 (0.87)	35.5	**1.82 (0.82)**
15	1242.0	29.15 (12.36)	96.1	3.41 (1.71)	59.1	**2.16 (1.14)**
18	2654.2	65.22 (27.80)	136.0	5.43 (2.53)	69.7	**3.55 (1.83)**
21	5646.2	171.26 (83.80)	196.9	8.14 (4.32)	93.4	**6.24 (3.34)**
24	22100.4	755.12 (383.41)	831.6	45.91 (26.08)	**211.4**	**19.26 (9.59)**

(d) Conjoined W_EGCC^{var} with exact consistencies

n	GAC*		FDGAC*		weak EDGAC*	
	bt	time (CPLEX)	bt	time (CPLEX)	bt	time (CPLEX)
12	316.5	503.68 (499.27)	70.2	136.50 (135.61)	**33.9**	78.21 (77.24)
15	*	*	93.0	187.25 (185.63)	**56.7**	116.04 (115.09)
18	*	*	134.1	351.94 (349.18)	**66.1**	176.95 (175.36)
21	*	*	183.2	719.61 (715.90)	**90.5**	382.80 (379.53)
24	*	*	*	*	*	*

(e) W_REGULAR^{var} with exact consistencies using flow algorithms

n	GAC*		FDGAC*		Weak EDGAC*	
	bt	time	bt	time	bt	time
12	13028.0	288.03	182.0	20.17	154.9	14.59
15	*	*	497.1	105.17	314.3	57.12
18	*	*	1166.4	667.77	1094.0	529.10
21	*	*	*	*	*	*
24	*	*	*	*	*	*

(f) W_REGULAR^{var}

n	Relaxed GAC*		Relaxed FDGAC*		Relaxed weak EDGAC*	
	bt	time (CPLEX)	bt	time (CPLEX)	bt	time (CPLEX)
12	*	*	*	*	154.9	179.16 (73.92)
15	*	*	*	*	*	*
18	*	*	*	*	*	*
21	*	*	*	*	*	*
24	*	*	*	*	*	*

Scenario (c) also outperforms other scenarios in addition to (e), which uses the flow-based approach. Scenario (c) runs up to 40 times faster than (a), 675 times faster than (b), 61 times faster than (d), and 98 times than (f). Under scenario (c), the instances can be solved up to 40.5 times faster than (a), 61.3 times faster then (d), and 10.4 times faster than (f). Scenario (c) can also solve larger instances like $n = 24$ within the given time limit, which is not possible in other scenarios.

9.1.3 Weighted Tardiness Scheduling Problem

The Weighted Tardiness Scheduling Problem [34] is to find a schedule of n jobs to be processed, and avoid two jobs processing at the same time. Each i^{th} job requires a processing time p_i, and to be processed without interruption. Each i^{th} job also has a due date d_i by which it should ideally be finished. In the problem, a set of total available time slots T are given to process all jobs. If a job cannot be finished exactly on the due date, an earliness/tardiness penalty is given. If two jobs run at the same time, a penalty is also given for extra resources.

To model this problem as WCSP, n variables $\{x_i \mid i = 1, \ldots, n\}$ with domains T denote respectively when the i^{th} job starts. To model earliness/tardiness penalties, we place a unary cost function $t(x_i)$ on each variable x_i, which returns 0 only when $x_i + p_i = d_i$, and random non-zero values otherwise. To model penalties for running more than one job at the same time slot, a W_DISJUNCTIVE^{val} cost function (please refer to the Appendix) is placed on each pair of jobs. We also place preferences on the starting time for each job, which can be modeled as table cost functions. In our experiment, we assume the number of total available time slots $|T| = 4n$.

The W_DISJUNCTIVE^{val} cost functions are modeled differently in each scenario. In scenarios (a) and (b), we use the individual W_DISJUNCTIVE^{val} cost functions in the model, whereas in scenarios (c) and (d), the W_DISJUNCTIVE^{val} cost functions are conjoined into one cost function. Note that scenarios (e), (f) are omitted, since there are no efficient ways to decompose W_DISJUNCTIVE^{val} into simpler cost functions to the best of our knowledge.

Results are shown in Table 4. Again, we observe that enforcing stronger consistencies can give better performance. Similar to Section 9.1.1, scenarios (a) and (c) give higher number of backtracks but require less time to solve than (b) and (d) respectively. For example, when $n = 7$, scenario (b) reduces the number of backtracks only 1.06 times more than (a), but the runtime is increased by 126 times. Similar results can be found between scenarios (c) and (d). When $n = 7$, the number of backtracks in scenario (c) is reduced by at most 1.12 times, but the runtime is increased up to 74 times more than (d).

The results also show conjoining cost functions is beneficial. The number of backtracks in scenario (c) is reduced up to 2.4 times more, but the runtime is up to 3.1 times faster than (a). Similarly, scenario (d) reduces the number of backtracks up to 2.1 times than (b). Unlike scenario (c), (d) shows its benefit on runtime only when $n \geq 6$. For example, when $n = 7$, scenario (d) runs 3.9 times faster than (b).

Similar to Table 3, the results show that scenario (c) outperforms all other scenarios. With (relaxed) weak EDGAC*, scenario (c) runs up to 3.1 times faster than (a), 183 times faster than (b), and 64 times faster than (d). Instances with $n = 9$ can also be

Table 4: The weighted tardiness scheduling problem using W_DISJUNCTIVE^{val}

(a) W_DISJUNCTIVE^{val}						
n	Relaxed GAC*		Relaxed FDGAC*		Relaxed weak EDGAC*	
	bt	time (CPLEX)	bt	time (CPLEX)	bt	time (CPLEX)
5	91.0	2.77 (1.38)	45.1	2.42 (1.14)	32.0	2.20 (1.03)
6	1335.3	17.11 (8.75)	832.1	11.47 (5.81)	184.3	4.15 (1.74)
7	4928.1	30.40 (16.43)	2462.3	17.95 (7.91)	623.5	5.82 (2.92)
8	13074.8	132.92 (68.32)	4928.6	66.38 (31.06)	1128.3	25.74 (12.58)
9	*	*	11769.9	385.32 (196.76)	4590.6	141.71 (74.91)
(b) W_DISJUNCTIVE^{val} with exact consistencies						
n	GAC*		FDGAC*		Weak EDGAC*	
	bt	time (CPLEX)	bt	time (CPLEX)	bt	time (CPLEX)
5	87.4	15.01 (13.63)	41.7	12.37 (11.07)	28.1	8.76 (7.58)
6	1208.9	140.31 (131.38)	729.9	109.42 (103.14)	159.9	80.35 (78.19)
7	*	*	*	*	584.2	738.10 (735.21)
8	*	*	*	*	*	*
9	*	*	*	*	*	*
(c) Conjoined W_DISJUNCTIVE^{val}						
n	Relaxed GAC*		Relaxed FDGAC*		Relaxed weak EDGAC*	
	bt	time (CPLEX)	bt	time (CPLEX)	bt	time (CPLEX)
5	69.3	1.90 (0.84)	40.9	2.02 (0.95)	28.0	**1.85 (0.89)**
6	652.3	11.78 (5.06)	465.3	7.08 (3.48)	117.8	**2.77 (1.38)**
7	3563.2	20.68 (10.03)	1273.6	10.55 (4.92)	415.6	**4.02 (2.08)**
8	5643.8	103.83 (52.92)	2763.1	59.08 (32.89)	**763.1**	**16.57 (8.78)**
9	16153.9	582.76 (317.90)	5018.4	214.50 (139.42)	**1935.3**	**45.51 (24.37)**
(d) Conjoined W_DISJUNCTIVE^{val} with exact consistencies						
n	GAC*		FDGAC*		weak EDGAC*	
	bt	time (CPLEX)	bt	time (CPLEX)	bt	time (CPLEX)
5	66.1	19.25 (17.97)	35.6	15.93 (14.28)	**26.8**	11.42 (10.16)
6	572.0	170.39 (163.93)	382.3	82.35 (78.69)	**116.2**	20.45 (19.08)
7	*	*	1137.2	783.59 (778.04)	**400.4**	256.64 (254.83)
8	*	*	*	*	*	*
9	*	*	*	*	*	*

solved in scenario (c) within the given time limit by enforcing GAC* only, which is not possible for other scenarios.

9.2 Using PILPS Cost Functions

To demonstrate the efficiency of using PILPS cost functions, we consider existing flow-based projection-safe cost functions which are also PILPS. Exact consistencies on such cost functions can be maintained using flow algorithms. Or we can also enforce relaxed consistencies on their conjunctions as in the best scenario (c) in Section 9.1.

We consider the following 2 scenarios for each benchmark.

(g) modeling by conjoined flow-based projection-safe cost functions and enforcing relaxed consistencies using a linear programming approach, and;
(h) modeling by the flow-based projection-safe cost functions and enforcing exact consistencies using flow algorithms.

To utilize the global cost functions described above, we soften the following problems by replacing the global constraints by their soft variants. Random preferences are added to the instances in the form of table cost functions.

9.2.1 Car Sequencing Problem

The car sequencing problem (prob001 in CSPLib) is a specialized version of the generalized car sequencing problem, where the cost of an option to equip on one type of cars is always 1, *i.e.* each assembly line is only allowed to equip an option i on at most m_i cars for every s_i cars. Instead of using W_SLIDINGSUM^{dec}, overlapping W_AMONG^{var} [47] cost functions are used to ensure the new restrictions. Instances with different n are used to compare the runtime and the number of backtracks. In scenario (h), we use the individual flow-based projection-safe W_AMONG^{var} cost functions in the model, whereas, in scenario (g), the W_AMONG^{var} cost functions are conjoined into one cost function.

Results are shown in Table 5. In both scenarios, (relaxed) weak EDGAC* gives better performance with exceptions. In scenario (h), instances with $n = 12$ and $n = 14$ can be solved faster with FDGAC* than weak EDGAC*, due to the high overhead of weak EDGAC*. However, weak EDGAC* outruns FDGAC* when $n \geq 16$, as the pruning in the search space compensates the extra overhead.

We also observe that conjoining cost functions are beneficial when instance is large. Conjunction gives stronger pruning power. Scenario (g) can reduce the number of backtracks up to 500 times more than (h). However, conjunction also incurs extra overhead, which cannot be compensated when the instances are small. For example, when $n = 12$, scenario (g) runs slower than (h). When the instance size grows, the reduction in the search space is large enough to compensate the overhead. For example, when $n = 18$, scenario (g) runs up to 20 times faster than (h).

Table 5: The car sequencing problem with W_AMONG^{var}

(g) Conjoined W_AMONG^{var}						
n	Relaxed GAC*		Relaxed FDGAC*		Relaxed weak EDGAC*	
	bt	time (CPLEX)	bt	time (CPLEX)	bt	time (CPLEX)
12	159.1	3.30 (1.60)	11.9	0.66 (0.41)	**8.4**	0.86 (0.54)
14	326.4	8.43 (4.23)	13.8	2.59 (1.76)	**11.1**	1.38 (0.87)
16	1366.2	44.96 (25.25)	16.5	3.44 (2.17)	**13.1**	2.49 (1.66)
18	1875.1	89.45 (54.23)	21.3	8.27 (5.59)	**18.4**	**3.74 (2.36)**
20	*	*	239.2	240.13 (152.24)	**78.0**	**61.29 (43.05)**
(h) W_AMONG^{var} with exact consistencies using flow algorithms						
n	GAC*		FDGAC*		Weak EDGAC*	
	bt	time	bt	time	bt	time
12	1582.7	0.30	19.7	**0.02**	18.4	0.04
14	58528.6	14.69	43.5	**0.08**	42.9	0.15
16	*	*	836.8	6.40	561.9	**2.46**
18	*	*	10657.8	86.10	9129.9	75.04
20	*	*	*	*	*	*

9.2.2 Examination Timetabling Problem

The examination timetabling problem is to find a schedule for n examinations over d days for s groups of students. Each group of students is required to attend a particular set of examinations. The aim of the problem is to minimize the total number of days in which one student attends more than one examination. To model this problem, we use n variables $\{x_i \mid i = 1, \ldots, n\}$ with domain d. Each variable x_i represents when the i^{th} examination is scheduled on. For each group of students attending a particular set of examinations, $\{e_1, \ldots, e_n\}$, a W_ALLDIFF$^{var}(S)$ [39] cost function, where $S = \{x_{e_1}, \ldots, x_{e_n}\}$, is used to denote the violation cost of students having to attend two or more examinations on the same day. We also add preferences on examinations. For example, some examinations are preferred to be scheduled on the same day. The preferences can be modeled as table cost functions. We fix $s = n/2$ and $d = n/2$, and use different n in our experiment. In scenario (h),we use the individual flow-based projection-safe W_ALLDIFF$^{var}(S)$ cost functions in the model, whereas in scenario (g), the W_ALLDIFF$^{var}(S)$ cost functions are conjoined into one cost function.

Results are shown in Table 6. In most cases, (relaxed) weak EDGAC* outperforms all others consistencies. Although we find that relaxed FDGAC* outplays relaxed weak EDGAC* when $n = 20$ in scenario (g), the 0.12-second difference is insignificant. By conjoining PILPS cost functions, scenario (g) outperforms (h). Scenario (g) prunes the search space up to 174 times more and runs up to 22 times faster than (h). We also observe that the instances with $n \geq 30$ can be solved within the time limit in scenario (g) but not in (h).

9.2.3 Fair Scheduling

The fair scheduling problem [5] is to schedule n persons into s shifts over d days such that the schedule is *fair, i*.e. each person should be assigned to the same number of the i^{th} shift. We model the problem by nd variables $\{x_{ij} \mid i = 1, \ldots, n \wedge j = 1, \ldots, d\}$

Table 6: The soft examination timetabling problem with W_ALLDIFF^{var}

(g) Conjoined W_ALLDIFF^{var}						
n	Relaxed GAC*		Relaxed FDGAC*		Relaxed weak EDGAC*	
	bt	time (CPLEX)	bt	time (CPLEX)	bt	time (CPLEX)
20	8921.7	425.84 (253.94)	28.5	**4.18 (2.83)**	**12.3**	4.30 (2.87)
25	*	*	51.8	17.67 (10.95)	**22.8**	**16.14 (9.37)**
30	*	*	*	*	**47.6**	**50.86 (28.33)**
35	*	*	*	*	**68.0**	**91.75 (46.58)**
(h) W_ALLDIFF^{var} with exact consistencies using flow algorithms						
n	GAC*		FDGAC*		Weak EDGAC*	
	bt	time	bt	time	bt	time
20	*	*	3594.8	77.65	306.7	10.16
25	*	*	9024.0	390.81	2530.5	138.74
30	*	*	*	*	*	*
35	*	*	*	*	*	*

with domain s. Each variable $\{x_{ij}\}$ denotes the shift for the i^{th} person on the j^{th} day. We use the W_SAME$^{var}(\{x_{p_1,j} \mid j = 1,\ldots,d\},\{x_{p_2,j} \mid j = 1,\ldots,d\})$ [48] cost functions between each pair of persons p_1 and p_2 to model the restrictions. In our experiment, we also add preferences on personal choices, e.g. someone may prefer to be scheduled on some particular shifts. Such preferences are modeled by table cost functions. We fix $s = 5$ and $d = 5$ and use different n in our experiment. In scenario (h), we use the individual flow-based projection-safe W_SAME^{var} cost functions in the model, whereas in scenario (g), the W_SAME^{var} cost functions are conjoined into one cost function.

Table 7: The soft fair scheduling problem with W_SAME^{var}

(g) Conjoined W_SAME^{var}						
n	Relaxed GAC*		Relaxed FDGAC*		Relaxed weak EDGAC*	
	bt	time (CPLEX)	bt	time (CPLEX)	bt	time (CPLEX)
6	1693.8	110.22 (43.25)	74.3	11.26 (5.29)	**24.1**	**7.94 (3.30)**
8	*	*	115.9	25.32 (12.68)	**57.1**	**14.55 (7.23)**
10	*	*	194.5	83.34 (47.83)	**76.6**	**44.23 (21.34)**
12	*	*	783.9	590.81 (261.74)	**175.3**	**86.42 (40.63)**
(h) W_SAME^{var} with exact consistencies using flow algorithms						
n	GAC*		FDGAC*		Weak EDGAC*	
	bt	time	bt	time	bt	time
6	*	*	292.7	13.42	163.7	9.15
8	*	*	1292.2	89.53	793.6	50.73
10	*	*	5063.8	292.07	1154.6	189.47
12	*	*	*	*	*	*

Results are shown in Table 7. In all scenarios, (relaxed) weak EDGAC* outperforms other consistencies. Similar to Table 6, scenario (g) runs up to 4.3 times faster and prunes up to 26 times more than (h). Larger instance like $n = 12$ can also be solved within the given time limit in scenario (g), while it is not the case in (h).

9.2.4 Comparing the WCSP and the Integer Linear programming Approaches

Earlier, we use slightly easier problem instances so that we can make sensible comparisons with the weaker consistencies and the flow-based implementations. Note that integer linear programming solver can also solve our benchmarks competitively. We now use more difficult instances with more preferences (table cost functions) to compare the performances of modeling the problem with *integer linear programs* (ILPs) solved by the IBM ILOG CPLEX Optimizer 12.2 with both of the scenarios above. Those instances consist of non-linear parts, which are generated by adopting the features of the instances used by Allouche *et al.* [4]. We use the encoding method of Koster [23] to formulate binary cost functions as integer linear programs, while global cost functions are formulated as integer linear programs using the similar method as for PILPS cost functions.

Table 8: Comparison with integer linear programming

(a) Soft car sequencing

n	(h) & weak EDGAC*		(g) & relaxed weak EDGAC*		ILPs
	bt	time	bt	time (CPLEX)	time
12	1169.6	**2.827**	**66.5**	4.10 (2.90)	63.28
14	4663.7	18.18	**664.4**	**8.71 (5.83)**	469.09
16	41383.1	310.87	**4087.5**	**20.59 (8.99)**	*
18	*	*	**30658.0**	**117.17 (20.68)**	*
20	*	*	**322598.7**	**1517.01 (28.94)**	*

(b) Soft examination timetabling

n	(h) & weak EDGAC*		(g) & relaxed weak EDGAC*		ILPs
	bt	time	bt	time (CPLEX)	time
20	15009.0	18.24	**14881.7**	**18.21 (3.38)**	*
25	24852.6	73.87	**24191.8**	**47.81 (12.78)**	*
30	28991.1	313.56	**27597.3**	**98.13 (33.88)**	*
35	58346.6	1110.21	**54515.3**	**212.08 (59.47)**	*

(c) Soft fair scheduling

n	(h) & weak EDGAC*		(g) & relaxed weak EDGAC*		ILPs
	bt	time	bt	time (CPLEX)	time
6	8493.2	47.43	**7279.5**	**21.25 (4.99)**	1339.19
8	16553.1	133.07	**13637.9**	**79.43 (24.34)**	*
10	39838.9	632.94	**29654.8**	**141.45 (42.04)**	*
12	89478.5	2215.57	**59231.7**	**317.94 (98.46)**	*

Results are shown in Tables 8. We only show the results for scenarios (g) and (h) using (relaxed) weak EDGAC* as they perform the best among the other (relaxed) consistencies in the same scenario under the same setting. Similar to Tables 5, 6, and 7, our WCSP scenario using conjunctions of PILPS cost functions (scenario (g))

runs faster and prunes more than the scenario with individual cost functions with the flow-based approach (scenario (h)). We observe that, in scenario (g), the number of backtracks can be reduced up to 17 times, and the solution can be found up to 15 times faster. On the other hand, our scenarios run faster in general when compared with the ILP models using CPLEX as the integer linear program solver. For example, in soft car scheduling problem, the instances can be solved under scenario (g) up to 53 times faster than using the ILP approach. In soft fair scheduling, scenario (g) gives the optimal solutions 63 times faster than using the ILP approach. We also observe that in soft examination timetabling, modeling by ILP fails to solve all instances at the given time limits, while WCSP can.

10 Related Work

Our research extends previous work by (a) also considering global cost functions with no known polynomial time algorithms for minimum cost computation, and (b) improving pruning by considering conjunctions of global cost functions.

Lee and Leung [28, 30] make practical the processing of global cost functions by defining the framework of $\mathcal{T}$ *projection-safety*. They study the special case when $\mathcal{T}$ is flow-basedness, and propose efficient flow algorithms for enforcing GAC* and FDGAC* of global cost functions. Lee and Leung [29] define and give efficient algorithms for the enforcement of weak EDGAC*, which is stronger than both GAC* and FDGAC*. Lee *et al.* [31] give theoretical properties of r-projections with respect to tractable projection-safety, and propose new consistency enforcement algorithms for *polynomially decomposable cost functions* based on dynamical programming.

On the other hand, Allouche *et al.* [3] give examples on the global cost functions that can be *decomposed into binary or ternary table cost functions*. They show that if the hyper-graph representing the cost functions from decomposing a global cost function is Berge-acyclic, *enforcing Terminal Directional Arc Consistency [3] and Virtual Arc Consistency [14, 15] on the decomposed cost functions and the original cost function are equivalent*.

11 Conclusions

Our contributions are five-fold. First, we define the *polytime linear projection-safe (PLPS) cost functions* based on their *integer linear program* formulations with size polynomial in their number of variables and maximum domain size. The minima of PLPS cost functions can be computed by solving their related integer linear programs. We also give sufficient conditions for polytime linear projection-safety. Second, we propose *relaxed consistencies* on PLPS cost functions, which are weaker but the enforcement can be much more efficient compared to the exact counterparts. The approximated minimum of PLPS cost functions can be computed by approximating their related integer linear programs with linear relaxation. We give proofs for the feasibility of projecting the smallest integral cost which is not less than the approximated minimum. Thus, we can define the relaxed version for the exact consistency notions,

including GAC*, FDGAC*, and weak EDGAC*, by reformulating their requirements based on the minima of a set of cost functions replaced by their approximated minima. Third, we propose the use of the conjunctions of PLPS cost functions. We show that the conjunctions of PLPS cost functions remain PLPS, so that relaxed consistencies can still be applied on them. We show that propagating on a conjunction using the exact consistencies is stronger than propagating on the individual cost functions. Although it is not always true when relaxed consistencies are enforced, the benefits of using the conjunctions of PLPS cost functions are shown experimentally. Fourth, we define *poly-time integral linear projection-safe (PILPS) cost functions*, which is a subclass of PLPS cost functions. PILPS cost functions are special PLPS cost functions and their exact minima can be computed by solving their corresponding integer linear programs with linear relaxation. In addition, the minimum of an PILPS function can be computed in polynomial time. The same is not necessarily true for the conjunctions of PILPS cost functions, which we show to be still PLPS. Our central results show that propagating on individual PILPS cost functions using the exact (or relaxed since they are the same) consistencies is weaker than propagating on the conjunction of all these PILPS cost functions using the relaxed versions of the consistencies, which is in turn weaker than propagating on the conjunction using the exact consistency. The latter is NP-hard in general. Therefore, it is always more desirable to propagate on conjunctions of PILPS cost functions using even just relaxed consistencies. The results are useful when we have cost functions whose minimum computation is polynomial time but that for conjunctions of such cost functions is not. We show that flow-based projection safe [28, 30] cost functions are PILPS, but minimum computation of their conjunctions is NP-hard in general. Fifth, we demonstrate the practicality of our framework with empirical results. We conduct experiments on several examples of PLPS and PILPS cost functions, together with their conjunctions, against the current approaches as well as pure integer programming approach. We observe orders of magnitude in runtime and search space improvements when the conjunctions of PLPS or PILPS cost functions are used together with relaxed consistencies. The results agree with our theoretical predictions.

We highlight three possible directions of future work. The first question is whether we can enhance the relaxed consistencies for stronger consistency notions like optimal soft arc consistency (OSAC) [16, 15], virtual arc consistency (VAC) [14, 15] and k-consistency [13]. Currently, we only give the relaxed versions of GAC* [44], FDAC* [28, 30], and weak EDGAC* [29, 30]. Those consistency notions can be relaxed by replacing the minima into approximated minima in their conditions. It might not be straightforward to relax consistency notions with different kinds of conditions and those involving rational costs such as OSAC [16, 15] and VAC [14, 15]. It is interesting to see if there are different ways to relax the consistency notions. The second question is whether we can give exact characterizations of the conditions leading to PLPS and PILPS cost functions. Currently, we only give sufficient conditions but necessary conditions may allow us to identify many more useful and yet efficiently implementable global cost functions. The third question looks at the possible connection between P(I)LPS cost functions and the well-studied class of submodular functions, which might the required integrality property.

Acknowledgements

We are grateful to the anonymous referees for their constructive comments. The work described in this paper was substantially supported by grant CUHK413710 from the Research Grants Council of Hong Kong SAR and grant F-HK019/12T from the Consulate General of France of Hong Kong and the Research Grants Council of Hong Kong SAR.

References

[1] A. Aggoun and N. Beldiceanu. Extending CHIP in Order to Solve Complex Scheduling and Placement Problems. *Mathematical and Computer Modelling*, 17(7):57–73, 1993.

[2] M. Akplogan, S. de Givry, J.-P. Métivier, G. Quesnel, A. Joannon, and F. Garcia. Solving the Crop Allocation Problem using Hard and Soft Constraints. *RAIRO - Operations Research*, 47(2):151–172, 2013.

[3] D. Allouche, C. Bessière, P. Boizumault, S. de Givry, P. Gutierrez, S. Loudni, J.-P. Métivier, and T. Schiex. Filtering Decomposable Global Cost Functions. In *Proceedings of AAAI'12*, pages 407–413, 2012.

[4] D. Allouche, S. Traoré, I. André, S. de Givry, G. Katsirelos, S. Barbe, and T. Schiex. Computational Protein Design as a Cost Function Network Optimization Problem. In *Proceedings of CP'12*, pages 840–849, 2012.

[5] N. Beldiceanu, M. Carlsson, and J.X. Rampon. Global Constraint Catalog. Technical Report T2005-08, Swedish Institute of Computer Science, 2005. Available at http://www.emn.fr/x-info/sdemasse/gccat/.

[6] N. Beldiceanu and E. Contejean. Introducing Global Constraints in CHIP. *Mathematical and Computer Modelling*, 20(12):97–123, 1994.

[7] N. Beldiceanu, I. Katriel, and S. Thiel. Filtering Algorithms for the Same Constraints. In *Proceedings of CPAIOR'04*, pages 65–79, 2004.

[8] C. Bessière, E. Hebrard, B. Hnich, and T. Walsh. The Complexity of Reasoning with Global Constraints. *CONSTRAINTS*, 12(2):239–259, 2007.

[9] C. Bessière and P. V. Hentenryck. To Be or Not to Be . . . a Global Constraint. In *Proceedings of CP'03*, pages 789–794, 2003.

[10] C. Bessière, G. Katsirelos, N. Narodytska, C.-G. Quimper, and T. Walsh. Propagating Conjunctions of ALLDIFFERENT Constraints. In *Proceedings of AAAI'10*, pages 27–32, 2010.

[11] C. Bessière and J.-C. Régin. Arc Consistency for General Constraint Networks: Preliminary Results. In *Proceedings of IJCAI'97*, pages 398–404, 1997.

[12] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J.P. Warners. Radio Link Frequency Assignment. *CONSTRAINTS*, 4(1):79–89, 1999.

[13] M. C. Cooper. High-Order Consistency in Valued Constraint Satisfaction. *CONSTRAINTS*, 10(3):283–305, 2005.

[14] M. C. Cooper, S. de Givry, M. Sànchez, T. Schiex, and M. Zytnicki. Virtual Arc Consistency for Weighted CSP. In *Proceedings of AAAI'08*, pages 253–258, 2008.

[15] M. C. Cooper, S. de Givry, M. Sànchez, T. Schiex, M. Zytnicki, and T. Werner. Soft Arc Consistency Revisited. *Artificial Intelligence*, 174(7-8):449–478, 2010.

[16] M. C. Cooper, S. de Givry, and T. Schiex. Optimal Soft Arc Consistency. In *Proceedings of IJCAI'07*, pages 68–73, 2007.

[17] M. C. Cooper, M. de Roquemaurel, and P. Régnier. A Weighted CSP Approach to Cost-Optimal Planning. *AI Communications*, 24(1):1–29, 2011.

[18] M. C. Cooper and T. Schiex. Arc Consistency for Soft Constraints. *Artifical Intelligence*, 154:199–227, 2004.

[19] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

[20] 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 IJCAI'05*, pages 84–89, 2005.

[21] J. N. Hooker. *Integrated Methods for Optimization*. Springer Science + Business Media, LLC, 2007.

[22] I. Katriel and S. Thiel. Complete Bound Consistency for the Global Cardinality Constraint. *CONSTRAINTS*, 10(3):115–135, 2005.

[23] A. M. Koster. *Frequency Assignment: Models and Algorithms*. PhD thesis, University of Maastricht, 1999.

[24] J. Larrosa. Node and Arc Consistency in Weighted CSP. In *Proceedings of AAAI'02*, pages 48–53, 2002.

[25] J. Larrosa. In the Quest of the Best Form of Local Consistency for Weighted CSP. In *Proceedings of IJCAI'03*, pages 239–244, 2003.

[26] J. Larrosa and T. Schiex. Solving Weighted CSP by Maintaining Arc Consistency. *Artificial Intelligence*, 159(1-2):1–26, 2004.

[27] J.-L. Lauriere. A Language and a Program for Stating and Solving Combinatorial Problems. *Artificial Intelligence*, 10(1):29–127, 1978.

[28] J. H. M. Lee and K. L. Leung. Towards Efficient Consistency Enforcement for Global Constraints in Weighted Constraint Satisfaction. In *Proceedings of IJCAI'09*, pages 559–565, 2009.

[29] J. H. M. Lee and K. L. Leung. A Stronger Consistency for Soft Global Constraints in Weighted Constraint Satisfaction. In *Proceedings of AAAI'10*, pages 121–127, 2010.

[30] J. H. M. Lee and K. L. Leung. Consistency Techniques for Flow-Based Projection-Safe Global Cost Functions in Weighted Constraint Satisfaction. *Journal of Artificial Intelligence Research*, 43:257–292, 2012.

[31] J. H. M. Lee, K. L. Leung, and Y. Wu. Polynomially Decomposable Global Cost Functions in Weighted Constraint Satisfaction. In *Proceedings of AAAI'12*, pages 507–513, 2012.

[32] J. H. M. Lee, K.L. Leung, and Y. W. Shum. Propagating Polynomially (Integral) Linear Projection-Safe Global Cost Functions in WCSPs. In *Proceedings of ICTAI'12*, pages 9–16, 2012.

[33] J. H. M. Lee and Y. W. Shum. Modeling Soft Global Constraints as Linear Programs in Weighted Constraint Satisfaction. In *Proceedings of ICTAI'11*, pages 305–312, 2011.

[34] J. Lenstra, A.R. Kan, and P. Brucker. Complexity of Machine Scheduling Problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[35] M.J. Maher, N. Narodytska, C.-G. Quimper, and T. Walsh. Flow-Based Propagators for the SEQUENCE and Related Global Constraints. In *Proceedings of CP'08*, pages 159–174, 2008.

[36] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[37] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.

[38] G. Pesant. A Regular Language Membership Constraint for Finite Sequences of Variables. In *Proceedings of CP'04*, pages 482–495, 2004.

[39] T. Petit, J.-C. Régin, and C. Bessière. Specific Filtering Algorithm for Over-Constrained Problems. In *Proceedings of CP'01*, pages 451–463, 2001.

[40] C.-G. Quimper, A. López-Ortiz, P.V. Beek, and A. Golynski. Improved Algorithms for the Global Cardinality Constraint. In *Proceedings of CP'04*, pages 542–556, 2004.

[41] J.-C. Régin. Generalized Arc Consistency for Global Cardinality Constraints. In *Proceedings of AAAI'96*, pages 209–215, 1996.

[42] J.-C. Régin. Combination of Among and Cardinality Constraints. In *Proceedings of CPAIOR'05*, pages 288–303, 2005.

[43] F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.

[44] M. Sànchez, S. de Givry, and T. Schiex. Mendelian Error Detection in Complex Pedigrees Using Weighted Constraint Satisfaction Techniques. *CONSTRAINTS*, 13(1-2):130–154, 2008.

[45] T. Sandholm. An Algorithm for Optimal Winner Determination in Combinatorial Auctions. In *Proceedings of IJCAI'99*, pages 542–547, 1999.

[46] T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proceedings of IJCAI'95*, pages 631–639, 1995.

[47] C. Solnon, V. Cung, A. Nguyen, and C. Artigues. The Car Sequencing Problem: Overview of State-of-the-Art Methods and Industrial Case-Study of the ROADDEF'2005 Challenge Problem. *European Journal of Operational Research*, 191(3):912–927, 2008.

[48] W.-J. van Hoeve, G. Pesant, and L.-M. Rousseau. On Global Warming: Flow-based Soft Global Constraints. *J. Heuristics*, 12(4-5):347–373, 2006.

[49] L. Wolsey. *Integer Programming*. Wiley, 1998.

[50] M. Zytnicki, C. Gaspin, and T. Schiex. Bounds Arc Consistency for Weighted CSPs. *Journal of Artificial Intelligence Research*, 35:593–621, 2009.

Appendix

More Examples of PLPS Cost Functions

We give two additional examples of Polytime Linear Projection-Safe (PLPS) cost functions: the W_EGCC and W_DISJUNCTIVE/W_CUMULATIVE cost functions.

The W_EGCC Cost Function

The $\text{EGCC}(S_X, S_Y)$ global constraint [40, 22] accepts a tuple $\ell \in \mathcal{L}(S_X \cup S_Y)$ iff $occ(v, \ell[S_X]) = y_v$ for every $y_v \in S_Y$, where $occ(j, \ell)$ is the number of occurrences of j in ℓ. Its soft variant $\text{W\_EGCC}^{var}(S_X, S_Y)$ is defined based on the variable-based violation measure.

Definition 1 *The cost function* $\text{W\_EGCC}^{var}(S_X, S_Y)$ *returns the cost of a tuple* $\ell \in \mathcal{L}(S_X \cap S_Y)$ *as:*

$$\text{W\_EGCC}^{var}(S_X, S_Y)(\ell) = \bigoplus_{y_v \in S_Y} |\ell[y_v] - occ(v, \ell[S_X])|$$

Note that $\text{W\_EGCC}^{var}(S_X, S_Y)$ is NP-hard. Quimper *et. la* [40] have shown that enforcing GAC on EGCC is NP-hard.

$\text{W\_EGCC}^{var}(S_X, S_Y)$, where $S_X = \{x_1, \ldots, x_n\}$ and $S_Y = \{y_{v_1}, \ldots, y_{v_m}\}$, can be modeled as a PLPS cost function. The corresponding integer linear program is defined as follows.

$$\begin{array}{cc}
\min \sum_{j=1}^{m} (L_j + U_j) & \text{s.t.} \\
\sum_{i=1}^{n} c_{x_i,v_j} - (\sum_{h \in D(y_{v_j})} h \cdot c_{y_{v_j},h}) - L_j + U_j = 0 & \forall j = 1 \ldots m \\
\sum_{j=1}^{m} c_{x_i,v_j} = 1 \quad \forall v_j \in D(x_i) & \forall i = 1 \ldots n \\
\sum_{j=1}^{m} c_{x_i,v_j} = 0 \quad \forall v_j \notin D(x_i) & \forall i = 1 \ldots n \\
\sum_{h \in D_{y_{v_j}}} c_{y_{v_j},h} = 1 & \forall j = 1 \ldots m \\
L_j \geq 0, U_j \geq 0 & \forall j = 1 \ldots m \\
0 \leq c_{x_i,v_j} \leq 1 & \forall x_i \in S_X, v_j \in D(x_i) \\
0 \leq c_{y_{v_j},h} \leq 1 & \forall y_{v_j} \in S_Y, h \in D_{y_{v_j}}
\end{array}$$

Define d_{max} to be the maximum domain size for the variables in $S = S_X \cap S_Y$. The corresponding integer linear program uses $(|S_X| + |S_Y| + 2) \cdot d_{max}$ variables and $4 \cdot |S_Y| + 2 \cdot |S_X| + d_{max} \cdot (|S_X| + |S_Y|)$ inequalities. If $x_i = v_j$, $c_{x_i,v_j} = 1$; otherwise $c_{x_i,v_j} = 0$. If $y_{v_j} = h$, $c_{y_{v_j},h} = 1$; otherwise $c_{y_{v_j},h} = 0$.

For example, consider the following WCSP $P = \{\mathcal{X}, \mathcal{D}, \{W_S\}, k\}$:

- $\mathcal{X} = \{x_1, x_2, y_a, y_b\}$;
- $D(x_1) = D(x_2) = \{a, b\}$, $D(y_a)D = (y_b) = \{0, 1, 2\}$;
- $W_S = \text{W\_EGCC}^{var}(\{x_1, x_2\}, \{y_a, y_b\})$.

$W_S(a, a, 2, 1) = 1$ since $|y_a - occ(a, (x_1, x_2))| + |y_b - occ(b, (x_1, x_2))| = 1$. The corresponding integer linear program is shown as follows.

$$\begin{gathered}
\min L_1 + U_1 + L_2 + U_2 \;\; s.t. \\
c_{x_1,a} + c_{x_2,a} - c_{y_a,1} - 2c_{y_a,2} - L_1 + U_1 = 0 \\
c_{x_1,b} + c_{x_2,b} - c_{y_b,1} - 2c_{y_b,2} - L_2 + U_2 = 0 \\
c_{x_1,a} + c_{x_1,b} = 1 \\
c_{x_2,a} + c_{x_2,b} = 1 \\
c_{y_a,0} + c_{y_a,1} + c_{y_a,2} = 1 \\
c_{y_b,0} + c_{y_b,1} + c_{y_b,2} = 1 \\
L_1 \geq 0, U_1 \geq 0, L_2 \geq 0, U_2 \geq 0 \\
0 \leq c_{x_i,j} \leq 1 \quad \forall x_i, j, x_i \in S, j \in D(x_i)
\end{gathered}$$

The W_DISJUNCTIVE/W_CUMULATIVE **Cost Function**

The DISJUNCTIVE(S, Π) global constraint [21] accepts a tuple $\ell \in \mathcal{L}(S)$ iff $(\ell[x_i] + p_i \leq \ell[x_j]) \vee (\ell[x_j] + p_j \leq \ell[x_i])$ for every pair of $x_i, x_j \in S$ and $p_i, p_j \in \Pi$. Its soft variant W_DISJUNCTIVE$^{val}(S, \Pi)$ is defined based on the value-based violation measure.

Definition 2 *The* W_DISJUNCTIVE$^{val}(S, \Pi)$ *returns the cost of a tuple* $\ell \in \mathcal{L}(S)$ *as:*

$$\text{W\_DISJUNCTIVE}^{val}(S, \Pi)(\ell) = \bigoplus_{t=0}^{T} \bigoplus_{i=1, p_i \in \Pi}^{n} \max(|\{i | \ell[x_i] \leq t \leq \ell[x_i] + p_i\}| - 1, 0)$$

The CUMULATIVE(S, Π, K) constraint [1] accepts a tuple $\ell \in \mathcal{L}(S)$ iff $|\{i \mid \ell[x_i] \leq t \leq \ell[x_i] + p_i\}| \leq K$ for every $x_i \in S$ and $p_i \in \Pi$, which is a generalized version of DISJUNCTIVE(S, Π). Its soft variant is defined based on the value-based violation measure.

Definition 3 *The* W_CUMULATIVE$^{val}(S, \Pi, K)$ *returns the cost of a tuple* $\ell \in \mathcal{L}(S)$ *as:*

$$\text{W\_CUMULATIVE}^{val}(S, \Pi, K)(\ell) = \bigoplus_{t=0}^{T} \bigoplus_{i=1, p_i \in \Pi}^{n} \max(|\{i | \ell[x_i] \leq t \leq \ell[x_i] + p_i\}| - K, 0)$$

Note that the W_DISJUNCTIVE$^{val}(S, \Pi)$ and W_CUMULATIVE$^{val}(S, \Pi, K)$ cost functions are NP-hard. Aggoun and Beldiceanu [1] have shown that enforcing GAC on the DISJUNCTIVE and CUMULATIVE constraints is NP-hard.

W_DISJUNCTIVE$^{val}(S, \Pi)$, where $S = \{x_1, \ldots, x_n\}$, can be modeled as a PLPS cost function. The corresponding integer linear program is defined as follows.

$$\begin{array}{cc}
\min \sum_{t \in T} U_t & \text{s.t.} \\
\sum_{i=1}^{n} \sum_{j=\max(t-p_i,0)}^{t} c_{x_i,j} - U_t \leq 1 & \forall t \in T \\
\sum_{d \in D(x_i)} c_{x_i,d} = 1 & \forall i = 1, 2, \ldots, n \\
0 \leq c_{x_i,d} \leq 1 & \forall x_i \in S, d \in D(x_i) \\
U_t \geq 0 & \forall t \in T
\end{array}$$

Define d_{max} to be the maximum domain size for the variables in S, the corresponding integer linear program uses $|S| \cdot d_{max} + |T|$ variables and $|T| + |S| + |S| \cdot d_{max}$ inequalities. If $x_i = d$, $c_{x_i,d} = 1$; otherwise $c_{x_i,d} = 0$.

W_CUMULATIVE$^{val}(S, \Pi, K)$, where $S = \{x_1, \ldots, x_n\}$, can be modeled as a PLPS cost function. The corresponding integer linear program is defined as follows.

$$\begin{array}{cc} \min \sum_{t \in T} U_t & \text{s.t.} \\ \sum_{i=1}^{n} \sum_{j=\max(t-p_i,0)}^{t} c_{x_i,j} - U_t \leq K & \forall t \in T \\ \sum_{d \in D(x_i)} c_{x_i,d} = 1 & \forall i = 1, 2, \ldots, n \\ 0 \leq c_{x_i,d} \leq 1 & \forall x_i \in S, d \in D(x_i) \\ U_t \geq 0 & \forall t \in T \end{array}$$

Define d_{max} to be the maximum domain size for the variables in S, the corresponding integer linear program uses $|S| \cdot d_{max} + |T|$ variables and $|T| + |S| + |S| \cdot d_{max}$ inequalities. If $x_i = d$, $c_{x_i,d} = 1$; otherwise $c_{x_i,d} = 0$.

For example, consider the WCSP $P = \{\mathcal{X}, \mathcal{D}, W_S, k\}$:

- $\mathcal{X} = \{x_1, x_2\}$;
- $D(x_1) = D(x_2) = \{0, 1, 2, 3\}$;
- $W_S =$ W_DISJUNCTIVE$^{val}(\{x_1, x_2\}, \{2, 3\})$.

$W_S(2, 0) = 1$ since when $t = 2$, $x_1 \leq t \leq x_1 + 2$ and $x_2 \leq t \leq x_2 + 3$ and the two jobs overlap each other, and $\bigoplus_{t=0}^{T} \sum_{i=1}^{n} |\{i | x_i \leq 2 \leq x_i + p_i\}| = 1$. The corresponding integer linear program is shown as follows.

$$\begin{gathered} \min U_0 + U_1 + U_2 + U_3 + U_4 \;\; s.t. \\ c_{x_1,0} + c_{x_2,0} - U_0 \leq 1 \\ c_{x_1,0} + c_{x_1,1} + c_{x_2,0} + c_{x_2,1} - U_1 \leq 1 \\ c_{x_1,1} + c_{x_1,2} + c_{x_2,0} + c_{x_2,1} + c_{x_2,2} - U_2 \leq 1 \\ c_{x_1,2} + c_{x_1,3} + c_{x_2,1} + c_{x_2,2} + c_{x_2,3} - U_3 \leq 1 \\ c_{x_1,3} + c_{x_1,4} + c_{x_2,2} + c_{x_2,3} + c_{x_2,4} - U_4 \leq 1 \\ c_{x_1,0} + c_{x_1,1} + c_{x_1,2} + c_{x_1,3} = 1 \\ c_{x_2,0} + c_{x_2,1} + c_{x_2,2} + c_{x_2,3} = 1 \\ U_0 \geq 0, U_1 \geq 0, U_2 \geq 0, U_3 \geq 0, U_4 \geq 0 \\ 0 \leq c_{x_i,d} \leq 1 \quad \forall x_i \in S, d \in D(x_i) \end{gathered}$$