Theoretical and Empirical Analysis of Cost-Function Merging for Implicit Hitting Set WCSP Solving

Javier Larrosa, Conrado Martínez, Emma Rollon
Computer Science Department, Universitat Politècnica de Catalunya
{larrosa, conrado, erollon}@cs.upc.edu

Abstract

The *Implicit Hitting Set* (HS) approach has shown very effective for MaxSAT solving. However, only preliminary promising results have been obtained for the very similar Weighted CSP framework. In this paper we contribute towards both a better theoretical understanding of the HS approach and a more effective HS-based solvers for WCSP. First, we bound the minimum number of iterations of HS thanks to what we call distinguished cores. Then, we show a source of inefficiency by introducing two simple problems where HS is unfeasible. Next, we propose two reformulation methods that merge cost-functions to overcome the problem. We provide a theoretical analysis that quantifies the magnitude of the improvement of each method with respect to the number of iterations of the algorithm. In particular, we show that the reformulations can bring an exponential number of iterations down to a constant number in our working examples. Finally, we complement our theoretical analysis with two sets of experiments. First, we show that our results are aligned with real executions. Second, and most importantly, we conduct experiments on typical benchmark problems and show that cost-function merging may be heuristically applied and it may accelerate HS algorithms by several orders of magnitude. In some cases, it even outperforms state-of-the-art solvers.

Introduction

In the *Weighted Constraint Satisfaction Problem* (WCSP) framework the goal is to optimize the combined cost of local cost functions. It includes important problems such as *Most Probable Explanation* in probabilistic networks (Dechter 2003; Koller and Friedman 2009) and it has many applications in *resource allocation* (Cabon et al. 1999), *bioinformatics* (Viricel et al. 2018), *scheduling* (Bensana, Lemaître, and Verfaillie 1999), *etc*. State-of-the-art WCSP solvers in the last two decades follow a *branch-and-bound* strategy enforcing at each expanded node local consistency properties that prune unfeasible nodes and provide effective bounds (Cooper et al. 2010; Larrosa and Schiex 2004; Allouche et al. 2015).

The MaxSAT problem has much in common with the WCSP framework. A surprisingly very effective technique for MaxSAT solving is the so called *Implicit Hitting Set* (HS) approach (Davies and Bacchus 2013; Berg, Bacchus, and Poole 2020; Bacchus et al. 2018). The HS approach was adapted from MaxSAT to WCSP in (Delisle and Bacchus 2013) and a simple algorithm, HS-WCSP, showed some promising preliminary results. Given the similarity between MaxSAT and WCSP, we believe that it is worth advancing further towards efficient HS-based WCSP solvers. In particular, we believe that efficient HS algorithms for WCSP must take advantage of the more structured language of WCSP (using cost functions) in contrast to the flatter language of MaxSAT (using clauses).

What makes WCSPs difficult in practice is that different cost functions often *disagree* on which assignments are likely to be good. For example, there are many optimization problems where the objective function is made of two conflicting components (*e.g:* risk *vs* benefit in finances, performance *vs* robustness in design,...) so optimal solutions corresponds to the best trade-off. Thus, one can think of WCSP solutions in general as those assignments that make complex trade-offs among many conflicting cost-functions. In HS algorithms, this phenomenon causes the discovery of many *cores* that are identical except for a portion that is different but represents (roughly) a different version of the same trade-off. When this happens, it translates to a large number of iterations. In this paper, and for the sake of a theoretical analysis, we consider the extreme case of this phenomenon that we call *core interchangeability*. We define core interchangeability at two levels: *symbolic* and *numeric*; and show that both types can cause HS-WCSP to iterate an exponential number of times.

Motivated by that observation, we propose a method to overcome the problem. Our approach is to reformulate the problem by conveniently *merging cost functions* aiming at making the space of cores more compact. We propose two different types of merging: i) *symbolic merging*, which treats weights as symbols without being aware of their numerical semantics, and ii) *numeric merging*, that treats weights as numbers. We show that symbolic merging is sufficient for dealing with symbolic interchangeability, but insufficient for dealing with numeric interchangeability, where only numeric merging renders HS-WCSP feasible. These results are summarized in Table 1. We complement our theoretical work empirically. First, we confirm the asymptotic bounds obtained for our two case study problems and evaluate how

	GreaterThan problem	*AllDiff* problem
Orig.	$\Omega(\frac{4^n}{\sqrt{n}})$	$\Omega(2^n)$
Symb.	$\Omega\left(\frac{c^{\sqrt{n}}}{n}\right)$ $c = 13.0019\ldots$	$\Omega(n)$
Num.	$\Omega(1)$	$\Omega(1)$

Table 1: Number of iterations required by HS-WCSP for the *GreaterThan* and *AllDiff* problems with three different encodings (*original* (Orig.), *symbolic* (Symb.) and *numeric* (Num.)). n denotes the number of variables. The bound is tight (there are actual implementations of HS-WCSP for which the $\Omega(\cdot)$ can be replaced by $\Theta(\cdot)$).

reasonable was to restrict our attention to the number of iterations as the main performance measure. Second, we explore how useful our theoretical analysis on our two artificial case study problems may be in practice. For that purpose, we propose a heuristic method to automatically identify pieces of real WCSPs which are likely to have core interchangeability (up to a certain level) and show that both symbolic and numeric merging are useful techniques in practice.

Preliminaries

CSP. A *Constraint Satisfaction Problem* (CSP) is a tuple (X, D, C) where X is a set of *variables*, D is a set of finite *domains* ($d_p \in D$ is the domain of variable $x_p \in X$) and C is a set of *constraints*. Each constraint $c_i \in C$ depends on a subset of variables $Y_i \subseteq X$, called the *scope*. The set D_{Y_i} denotes the Cartesian product of the domains of the variables in Y_i. Thus, $t \in D_{Y_i}$ is a *tuple* over Y_i and $t[S]$ with $S \subseteq Y_i$ denotes the projection of t over the variables of S. Constraints are boolean functions $c_i : D_{Y_i} \longrightarrow \{true, false\}$. A *solution* to the CSP is a tuple $t \in D_X$ that satisfies all the constraints ($c_i(t[Y_1]) = true$ for all $c_i \in C$). A CSP is *satisfiable* if it has at least one solution.

WCSP. A *Weighted CSP* (WCSP) is a CSP augmented with a set of *cost functions* F (i.e., (X, D, C, F)). A cost function $f_i \in F$ is a mapping $f_i : D_{Y_i} \longrightarrow \mathbb{N}$. A *solution* is a tuple $t \in D_X$ that satisfies all the constraints in C and we will assume that there is at least one such tuple. The *cost of a solution* t is $\sum_i f_i(t[Y_i])$. The WCSP problem consists of computing a minimum-cost solution $opt(P)$.

The following two WCSPs will be used in our analysis.

***GreaterThan* Problem.** The *GreaterThan* problem (noted $P_{GT}(n)$) is a WCSP with n variables $X = \{x_1, \ldots, x_n\}$ with $d_i = \{0, 1, \ldots, n-1\}$. It has a cost function $f_i(x_i) = x_i$ for each $1 \leq i \leq n$ and a single constraint $\sum_{i=1}^n x_i \geq n$. Clearly, an optimal solution is any assignment such that $\sum_{i=1}^n x_i = n$ and the optimal cost is $opt(P_{GT}) = n$.

***AllDiff* Problem.** The *AllDiff* problem (noted $P_{AD}(n)$) is a WCSP with n variables $X = \{x_1, \ldots, x_n\}$ with $d_i = \{0, 1, \ldots, n-1\}$. It has a cost function $f_i(x_i) = x_i$ for each $1 \leq i \leq n$ and one constraint $x_i \neq x_j$ for every pair of variables. Clearly, all solutions are optimal and correspond to assignments that assign a different value to each variable (e.g., $(x_1 \leftarrow 0, x_2 \leftarrow 1, \ldots, x_n \leftarrow n-1)$), and the optimal cost is $opt(P_{AD}) = \frac{n(n-1)}{2}$.

HS for Weighted CSPs

Next we review (and largely rephrase in an arguably simplified way) the HS approach for WCSP introduced in (Delisle and Bacchus 2013). The following definitions assume an arbitrary WCSP $P = (X, D, C, F)$ with m cost functions $F = \{f_1, f_2, \ldots, f_m\}$.

Weight Vector. A *weight vector* (or, simply, a *vector*) is $\vec{v} = (v_1, v_2, \ldots, v_m)$ with each component v_i being associated to cost function f_i. The value of v_i must be a weight occurring in f_i (i.e, $v_i \in \{f_i(t) \mid t \in D_{Y_i}\}$). The *cost* of a vector $\vec{v}$ is $cost(\vec{v}) = \sum_{i=1}^m v_i$.

Partial Order. Recall that the usual (partial) order among vectors, $\vec{v} \leq \vec{u}$, holds if for each component i we have that $v_i \leq u_i$. When $\vec{v} \leq \vec{u}$ we say that $\vec{u}$ *dominates* $\vec{v}$. When $\vec{v} \not\leq \vec{u}$ we say that $\vec{v}$ *hits* $\vec{u}$.

Induced CSP. A vector $\vec{v}$ *induces* a CSP $P(\vec{v})$ where every cost function f_i is replaced by constraint $(f_i \leq v_i)$.

Core. A vector $\vec{k}$ is a *core* if its induced CSP $P(\vec{k})$ is unsatisfiable. $\mathcal{C}$ will denote the set of cores. A core is *maximal* if it is not dominated by any other core or, in other words, if increasing any of its non-maximal components produces a satisfiable induced CSP. $\mathcal{MC}$ will denote the set of maximal cores. Note that this definition of core is similar but not equivalent to the usual concept of core in constraint programming and SAT (Gupta, Genc, and O'Sullivan 2021), since in our definition a core requires an underlying WCSP.

Minimum Cost Hitting Vector (MHV). A vector $\vec{h}$ *hits* a set of cores $\mathcal{K}$ if $\vec{h}$ hits each vector $\vec{k} \in \mathcal{K}$ (or, in other words, $\vec{h}$ is not dominated by any $\vec{k} \in \mathcal{K}$). The *minimum cost hitting vector* MHV of $\mathcal{K}$, noted MHV($\mathcal{K}$), is a hitting vector $\vec{h}$ with minimum cost. The MHV problem is a generalization of the Hitting Set Problem, so it is easy to see that it is NP-hard.

Core Extracting Solver (CES). A *core-extracting Solver*, noted CES, is a function that receives as input a WCSP P and a vector $\vec{h}$. If $\vec{h}$ is a core, it returns a core $\vec{k}$ such that $\vec{h} \leq \vec{k}$. Otherwise (i.e., if $\vec{h}$ is not a core), it returns $NULL$. Since a CES needs to solve a CSP and there is no special requirement about the core (so $\vec{h}$ itself could do the job), it is easy to see that core-extraction is no easier (and can be made no harder) than solving an NP-complete problem.

Algorithm HS-WCSP. The HS approach for WCSPs is based on the following. Let $\mathcal{K}$ be a set of cores of a WCSP P. Let vector $\vec{h}$ be a MHV of $\mathcal{K}$. Then,

- $cost(\vec{h})$ is a lower bound of $opt(P)$.
- If the induced CSP $P(\vec{h})$ is satisfiable, then any solution of $P(\vec{h})$ is an optimal solution of P with optimal cost $cost(\vec{h})$.

Algorithm 1: HS-WCSP receives as input a WCSP P and returns its optimal cost.

```
Input: WCSP $P$
Output:
 1: $\mathcal{K} := \emptyset; \vec{h} := \vec{0}$;
 2: while $true$ do
 3:     $\vec{k}$ :=CES($P, \vec{h}$)
 4:     if $\vec{k} = NULL$ then
 5:         return $cost(\vec{h})$
 6:     else
 7:         $\mathcal{K} := \mathcal{K} \cup \{\vec{k}\}$
 8:         $\vec{h}$ :=MHV($\mathcal{K}$)
 9:     end if
10: end while
```

- If the induced CSP $P(\vec{h})$ is unsatisfiable, then there is at least one core $\vec{k} \notin \mathcal{K}$ such that $\vec{h} \leq \vec{k}$.

Algorithm 1 shows HS-WCSP, a simple HS algorithm for WCSPs. It maintains a growing set of cores $\mathcal{K}$ and a minimum hitting vector $\vec{h}$ of $\mathcal{K}$. At each iteration, a core-extraction solver is invoked with the induced CSP $P(\vec{h})$ (line 3). If it is satisfiable, the cost of $\vec{h}$ is the optimal solution of the WCSP (line 5). Otherwise, core $\vec{k}$ is added to $\mathcal{K}$ (line 7), the MHV vector $\vec{h}$ is updated (line 8) and the algorithm continues. Note that the cost of $\vec{h}$ is a non-decreasing lower bound of the optimum solution.

In (Delisle and Bacchus 2013) the MHV problem is modeled as an Integer Program and CES is modeled as a SAT formula with assumptions, so both problems are solved with off-the-shelf solvers. They also propose some practical improvements of HS-WCSP. For the sake of simplicity in our analysis, we will not consider them until the empirical evaluation in Section .

Bounding Below the Number of Iterations

In this Section we study the performance of HS-WCSP in terms of the number of iterations. Note that iterations correspond to the number of cores that have to be extracted before obtaining the optimal value.

The number of iterations of an arbitrary implementation of HS-WCSP is non-deterministic since it depends on the cores computed by the CES function and the minimum hitting vectors computed by MHV and they may vary from implementation to implementation. Thus, we give a lower bound on that number which is independent from the particular implementation. For that purpose, we first identify a subset of maximal cores that are associated to necessary cores for obtaining the optimum cost. Consider a maximal core $\vec{k} \in \mathcal{MC}$ and let $U(\vec{k})$ be the set containing those cores dominated by $\vec{k}$ and not dominated by any other maximal core. That is,

$$U(\vec{k}) = \{\vec{k'} \in \mathcal{C} \mid \vec{k'} \leq \vec{k}, \forall_{\vec{k''} \in \mathcal{MC}, \text{s.t. } \vec{k'} \neq \vec{k''}} \vec{k'} \not\leq \vec{k''}\}$$

Definition 1 (distinguished core)**.** *A maximal core $\vec{k} \in \mathcal{MC}$ is distinguished if there is some $\vec{k'} \in U(\vec{k})$ such that $cost(\vec{k'}) < opt(P)$.*

Lemma 1. *Let $\vec{k} \in \mathcal{MC}$ be a distinguished maximal core, and let $\vec{h}$ be the MHV($\mathcal{C} - U(\vec{k})$)). Then, $cost(\vec{h}) < opt(P)$.*

The following theorem follows directly from the lemma.

Theorem 1 (lower bound)**.** *Let $\mathcal{DC}$ be the set of distinguished maximal cores. Then, HS-WCSP(P) iterates at least $|\mathcal{DC}|$ times.*

Definition 2 (HS-WCSP-max)**.** *HS-WCSP-max is the specific version of HS-WCSP such that the CES function always returns a maximal core when the induced CSP is unsatisfiable (i.e., $\vec{k} \in \mathcal{MC}$ in line3 of Algorithm 1).*

The following theorem gives an upper bound on the number of iterations of HS-WCSP-max. This bound is useful because it shows that the previous lower bound is sometimes tight, that is, it is not an underestimation since there are actual implementations of HS-WCSP for which it is attained.

Theorem 2 (upper bound)**.** *Let $\mathcal{MC}$ be the set of maximal cores. Then, HS-WCSP-max(P) iterates at most $|\mathcal{MC}|$ times.*

Now, we use the previous two results to show that HS-WCSP is unfeasible for the *GreaterThan* ($P_{GT}(n)$) and *AllDiff* ($P_{AD}(n)$) problems. Note that both of them have the same set of cost functions. Therefore, their set of vectors is $\{(v_1, v_2, \dots v_n) \mid 0 \leq v_i < n\}$.

We start showing that *GreaterThan* requires at least an exponential number of iterations. Besides, this best-case bound is tight in the sense that HS-WCSP-max requires exactly that number of iterations.

Lemma 2. *Consider $P_{GT}(n)$:*

- *$\mathcal{MC}$ is the set of vectors with cost $n-1$.*
- *$|\mathcal{MC}| = \binom{2n-2}{n-1} \sim \frac{4^{(n-1)}}{\sqrt{\pi n}}$.*
- *All maximal cores are distinguished.*

Theorem 3.

- *HS-WCSP($P_{GT}(n)$) iterates $\Omega(\frac{4^{(n-1)}}{\sqrt{\pi n}})$ times.*
- *HS-WCSP-max($P_{GT}(n)$) iterates $\Theta(\frac{4^{(n-1)}}{\sqrt{\pi n}})$ times.*

Proof. Follows directly from Theorem 1, Theorem 2 and Lemma 2. □

Next we do a similar analysis for the *AllDiff* problem.

Lemma 3. *Consider $P_{AD}(n)$:*

- *$\mathcal{MC}$ is the set of permutations of*

$$\begin{aligned}\{&(0, 0, (n-1), (n-1), \dots, (n-1)),\\ &(1, 1, 1, (n-1), (n-1), \dots, (n-1)),\\ &(2, 2, 2, 2, (n-1), (n-1), \dots, (n-1)),\\ &\dots,\\ &((n-2), \dots, (n-2))\}\end{aligned}$$

- *$|\mathcal{MC}| = 2^n - n$.*
- *All maximal cores are distinguished.*

Theorem 4.

- *HS-WCSP($P_{AD}(n)$) iterates $\Omega(2^n)$ times.*
- *HS-WCSP-max($P_{AD}(n)$) iterates exactly $2^n - n$ times.*

Proof. Follows directly from Theorem 1, Theorem 2 and Lemma 3. □

Core Interchangeability

The $P_{GT}(n)$ and $P_{AD}(n)$ problems are unfeasible for HS-WCSP because both of them have a symmetrical structure that produces an exponentially large number of necessary cores. In the following, we characterize the two forms of symmetry exhibited by these two problems.

Consider two m-vectors $\vec{v}$ and $\vec{u}$ and a subset of its components $I \subseteq \{1, 2, \ldots, m\}$. Then,

- $\vec{v}$ and $\vec{u}$ are *symbolic-equivalent* in I iff they are a permutation of each other in components in I and identical in components not in I.
- $\vec{v}$ and $\vec{u}$ are *numerical-equivalent* in I iff $cost(\vec{u}) = cost(\vec{v})$ and they are identical in components not in I.

For example, consider vectors $\vec{v} = (3, 7, 4)$, $\vec{u} = (3, 4, 7)$ and $\vec{w} = (3, 8, 3)$, and $I = \{2, 3\}$. They are all numeric-equivalent in I, because they are identical out of I and their cost in I is 11. Further, only $\vec{v}$ and $\vec{u}$ are symbolic-equivalent because $(7, 4)$ is a permutation of $(4, 7)$.

Definition 3 (symbolic core interchangeability)**.** *A WCSP P is s-interchangeable in I if for every pair of symbolic-equivalent vectors $\{\vec{v}, \vec{u}\}$ in I, $\vec{v}$ is a core iff $\vec{u}$ is a core.*

Definition 4 (numerical core interchangeability)**.** *A WCSP P is n-interchangeable in I if for every pair of numeric-equivalent vectors $\{\vec{v}, \vec{u}\}$ in I, $\vec{v}$ is a core iff $\vec{u}$ is a core.*

For example, the set of cores of $P_{GT}(n)$ is the set of vectors with cost less than n. Since this condition only depends on the cost, $P_{GT}(n)$ is both s-interchangeable and n-interchangeable in its full set of cost functions $I = \{1, \ldots, n\}$. On the other hand, the set of cores of $P_{AD}(n)$ is a set of permutations of $n-2$ canonical vectors. Since this condition is preserved under any permutation, $P_{AD}(n)$ is s-interchangeable but it is not n-interchangeable in $I = \{1, \ldots, n\}$ (for instance, $\vec{v} = (0, 1, 2, 3, 4, \ldots, n-1)$ is not a core but $\vec{u} = (0, 0, 3, 3, 4, \ldots, n-1)$ has the same cost and is a core).

Note that n-interchangeability is stronger than s-interchangeability. Consequently, it causes a larger number of core redundancy and therefore requires a more complex solving approach.

Cost Function Merging

s-interchangeability and n-interchangeability are well-defined examples of cost-functions *disagreeing* to each other as we discussed in the Introduction. They may be too strong to happen in practice, but we will use them to motivate and quantify the theoretical advantage of our approaches.

In the following, we present two methods aiming at reducing interchangeable cores. The first method, called *symbolic merging*, may be enough for problems with symbolic interchangeability. However, problems with numeric interchangeability need a stronger form of merging, that we call *numeric merging*.

Symbolic Merging

The idea behind symbolic merging is the following. Let $\vec{k} = (3, 2, 7, 5)$ be a core of some problem P. If the set of cores of P is symbolic-equivalent in $I = \{1, 2, 3\}$, it means that vectors $(2, 3, 7, 5), (2, 7, 3, 5), \ldots$ are also cores. These permutations of cores can be more compactly represented by just counting the number of times each value appears in indexes in I. That is, vectors having a 2, a 3 and a 7 in components in I are cores.

Formally, let $P = (X, D, C, F)$ be a WCSP and $G \subseteq F$ a subset of its functions. The symbolic merging of G is a new WCSP $P^S = (X \cup Y, D \cup D_Y, C \cup C_Y, F \cup H_Y - G)$ where Y is a set of auxiliary variables taking values in $\{0, \ldots, |G|\}$. There is a variable $y_w \in Y$ for every weight $w > 0$ appearing in any of the functions in G. Variable y_w counts the number of cost functions in G assigning weight w which is enforced by constraint $y_w = \sum_{f_i \in G} [f_i(Y_i) = w]$ in C_Y. Finally, the set of functions G is replaced by a cost function $f_w(y_w) = w \cdot y_w$ for every auxiliary variable y_w. It is easy to see that the reformulation preserves the optimal cost.

Let us now analyse the effect of symbolic merging on our two case studies. Let $G = F$. Since $P_{GT}(n)$ and $P_{AD}(n)$ have the same set of cost functions F, their reformulation will be identical (except for their original set of constraints C). For every weight $0 < w < n$ there is one auxiliary variable y_w, one constraint $y_w = \sum_{i=1}^{n} [x_i = w]$ and one cost function $f_w(y_w) = w \cdot y_w$. Original cost functions are removed. With this new encoding, the set of vectors in both problems is $\{\vec{v} = (v_1, v_2, \ldots v_{n-1}) \mid v_i = i \cdot \alpha_i,\ 0 \leq \alpha_i \leq n\}$.

Let $P^S_{GT}(n)$ denote the symbolic merging reformulation of $P_{GT}(n)$. We show next that symbolic merging reduces the number of iterations of HS-WCSP from exponential to subexponential (albeit superpolynomial). Although the reduction is significant, the algorithm remains unfeasible in practice for this problem because the minimum number of iterations cannot be bounded by any polynomial.

Lemma 4. *Consider $P^S_{GT}(n)$,*

- *$\mathcal{MC}$ is the set of vectors with cost $n-1$.*
- $|\mathcal{MC}| \sim \frac{1}{4n\sqrt{3}} \exp\left(\pi\sqrt{\frac{2n}{3}}\right) = \Theta\left(\frac{(13.0019\ldots)^{\sqrt{n}}}{n}\right)$
- *All maximal cores are distinguished.*

Theorem 5.

- *HS-WCSP($P^S_{GT}(n)$) iterates $\Omega\left(\frac{(13.0019\ldots)^{\sqrt{n}}}{n}\right)$ times.*
- *HS-WCSP-max($P^S_{GT}(n)$) iterates $\Theta\left(\frac{(13.0019\ldots)^{\sqrt{n}}}{n}\right)$ times.*

Proof. Follows directly from Theorem 1, Theorem 2 and Lemma 4. □

Let $P^S_{AD}(n)$ denote the symbolic merging reformulation of $P_{AD}(n)$. We show next that symbolic merging reduces the number of iterations of HS-WCSP from exponential to linear. Consequently, symbolic merging is enough for this problem.

Lemma 5. *Consider $P^S_{AD}(n)$,*

- *$\mathcal{MC}$ is*

$$\{(0, 2n, 3n, \dots, (n-2)n, (n-1)n),\\ (n, 0, 3n, \dots, (n-2)n, (n-1)n),\\ \dots,\\ (n, 2n, 3n, \dots, 0, (n-1)n),\\ (n, 2n, 3n, \dots, (n-2)n, 0)\}$$

- *$|\mathcal{MC}| = n$.*
- *All maximal cores are distinguished.*

Theorem 6.

- *HS-WCSP($P^S_{AD}(n)$) iterates $\Omega(n)$ times.*
- *HS-WCSP-max($P^S_{AD}(n)$) iterates n times.*

Proof. Follows directly from Theorem 1, Theorem 2 and Lemma 5. □

Numeric Merging

Numeric merging is the strongest form of merging. The idea is as follows. Let $\vec{k} = (3, 2, 7, 5)$ be a core of some problem P. If the set of cores of P is numerical-equivalent on indexes $I = \{1, 2, 3\}$, it means that vectors $(2, 2, 8, 5), (2, 3, 7, 5), (2, 7, 3, 5)$, ... are also cores. This set of cores can be more compactly represented by just recording the sum of the components in I. That is, vectors having a total cost of 12 in components in I are cores.

Formally, let $P = (X, D, C, F)$ be a WCSP and $G \subseteq F$ be a subset of its functions. The numeric merging of G is a new WCSP $P^N = (X, D, C, F \cup \{g\} - G)$ where G is replaced by cost function $g(X) = \sum_{f_i \in G} f_i(Y_i)$. The reformulation preserves the optimal cost.

Let us now analyse the effect of numerical merging on our two case studies. Let $G = F$. Since $P_{GT}(n)$ and $P_{AD}(n)$ have the same set of cost functions F, their reformulation will be identical (except for their original set of constraints C). All their unary cost-funcitons are replaced by a single n-ary cost function $g(X) = \sum_{i=1}^n x_i$. Consequently, with this new encoding, vectors become 1-dimensional and the set of vectors corresponds to $\vec{v} = (v_1)$ with $0 \le v_1 \le (n-1)^2$.

Let $P^N_{GT}(n)$ and $P^N_{AD}(n)$ denote the numerical merging reformulation of $P_{GT}(n)$ and $P_{AD}(n)$, respectively. Then,

Lemma 6. *Consider $P^N_{GT}(n)$,*

- *$\mathcal{MC}$ is $\{(n-1)\}$.*
- *Vector $(n-1)$ is distinguished.*

Theorem 7.

- *HS-WCSP($P^N_{GT}(n)$) iterates at least once.*
- *HS-WCSP-max($P^N_{GT}(n)$) iterates once.*

Proof. Follows directly from Theorem 1, Theorem 2 and Lemma 6. □

The numerical merging of all functions in the *GreaterThan* problem reduces the number of iterations from exponential to a constant. Of course, it begs the question of whether the single call to the CES function with the reformulated problem pays off in practice.

Although symbolic merging was already a feasible approach for the *AllDiff* problem, for completeness, we also report the effect of numeric merging on this problem.

Lemma 7. *Consider $P^N_{AD}(n)$,*

- *$\mathcal{MC}$ is $\{((n(n-1)/2) - 1)\}$.*
- *Vector $((n(n-1)/2) - 1)$ is distinguished.*

Theorem 8. *HS-WCSP($P^N_{AD}(n)$) iterates 1 time.*

Proof. Follows directly from Theorem 1, Theorem 2 and Lemma 7. □

Experimental Results

We implemented the HS-WCSP in C++ as follows[1]. The core-extraction solver (CES, line 3 in Algorithm 1) uses the assumption-based SAT solver *CaDiCal* (Biere et al. 2020), and the minimum hitting vector solver (MHV, line 8 in Algorithm 1) is implemented as a 0-1 integer program solved to optimality with CPLEX. Our encodings are similar to (Delisle and Bacchus 2013)). The merged cost-functions for both symbolic and numeric merging are encoded into SAT using the totalizer encoding (Joshi, Martins, and Manquinho 2015). All experiments were run on a Linux machine with 2.90 GHz CPU, 128GB RAM memory and 16 cores.

GreaterThan and *AllDiff* Problems

We have provided tight bounds on the number of iterations for two combinatorial problems with three different encodings when the CES function computes maximal cores (i.e, HS-WCSP-max). However, our theoretical analysis only gives partial information about the cost of executing HS-WCSP-max on them. In particular, some limitations of our analysis are: i) It is asymptotic, so it may not capture what happens with small instances. ii) It focus on the number of iterations but disregards the cost of each iteration which may change significantly with the different encodings.

To assess the effect of these limitations, we report information from real executions of HS-WCSP-max on the two problems with the three different formulations. We report the number of iterations to see the accuracy of our asymptotics; the number of calls to the SAT solver because the CES implementation obtains maximal cores as a sequence of non-maximal core extractions; the CPU time because the encodings are increasingly more sophisticated and it may be worth to see how it affects CPU time.

Figure 1 reports the results for the *GreaterThan* (top row) and *AllDiff* (bottom row) problems. The first column shows the actual number of iterations for the three encodings. As expected, it follows an exponential, sub-exponential (albeit super-polynomial) and constant growing pattern for *GreaterThan*; and an exponential, linear and constant growing pattern for *AllDiff*. We also include the asymptotic bounds of *GreaterThan* (dashed lines) which turn to be very precise for the original encoding (the two lines overlap) and quite accurate for the symbolic merging. The second column shows the number of calls to the SAT solver. In both problems, the number of calls seems to be just a constant away

[1]https://github.com/erollon/MHS-WCSP

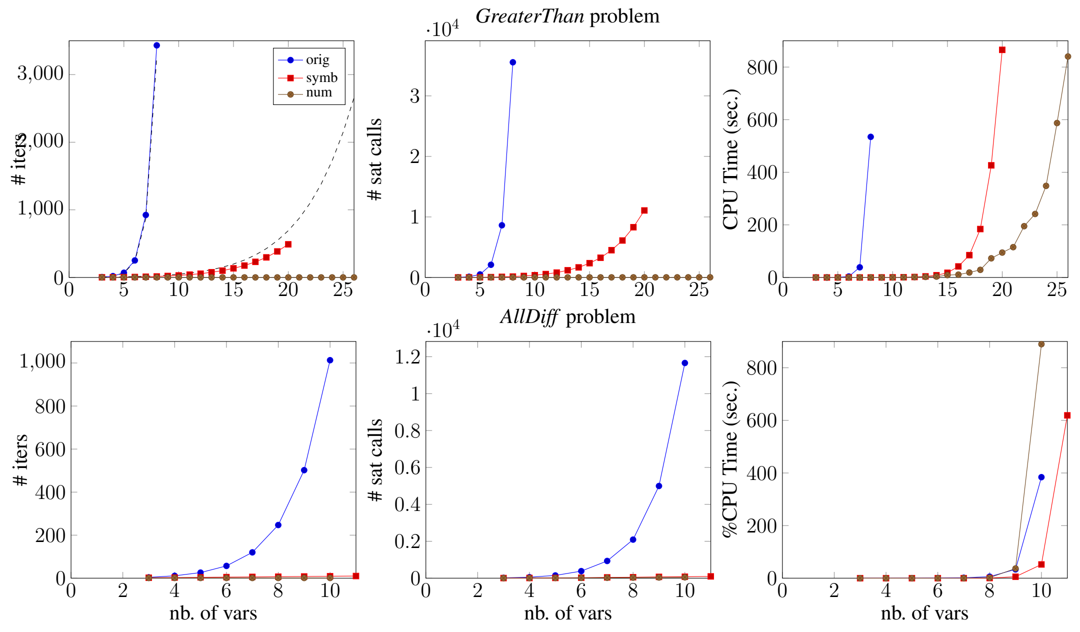

Figure 1: Number of iterations (first column), number of SAT calls (second column) and CPU time in seconds (third column) as a function of the number of variables on the *GreaterThan* problem (top row) and *AllDiff* problem (bottom row) using the original encoding (orig.) symbolic-merging (symb.) and numeric merging (num.). Time limit is 900 seconds.

from the number of iterations (around 10 times). Finally, the third column shows the CPU time. It can be seen that merging clearly pays-off in both problems and, consistently with the theoretical results, numeric merging is the best option for *GreaterThan* and symbolic merging is the best option for *AllDiff*. Moreover, we see that if we experiment across encodings, the number of iterations is not always a good measure, at least with the totalizer encoding for summations.

The experiments show that the cost of the CES component (which corresponds to the aggregated cost of SAT solving) seems to grow exponentially with the size of the instances. There are two lessons to be taken from that: merging can only be done in a limited way (i.e, with respect to small subsets of cost functions), and the totalizer encoding has to be replaced by a more sophisticated alternative to augment the applicability of our approach.

Benchmark Instances

In the Introduction we argued that difficult real problems have cost functions that *disagree* with each other and that this disagreement turns into an approximation of our notions of core interchangeability. To test our hypothesis, and considering what we have learned in the previous experiments, we need a method to identify clusters of cost functions with that pattern. We conjectured that such clusters would correspond to groups of functions that share many variables in their domain. Accordingly, we implemented a heuristic to partition the set of cost functions of arbitrary WCSPs into clusters. For that, we used the well-known concept of tree-decomposition (Kask et al. 2005). Tree-decompositions are frequently used in WCSP solvers to identify and exploit structural properties. Intuitively, a *tree-decomposition* (TD) of a WCSP is an arrangement of its cost functions into clusters such that the cluster structure is acyclic. The width of a TD is the size of the largest combined scope among its clusters. There are many heuristics for obtaining a TD with small width (Bodlaender and Koster 2010).

Therefore, we computed a TD and merged all the cost-functions that are placed in the same cluster as a pre-process. Table 2 reports the results on the Spot5 benchmark merging functions according to a min-fill TD (Bodlaender and Koster 2010). All instances were made *virtual arc consistent* (VAC) before the execution. For this experiment, we used a more realistic HS-WCSP implementation that does not guarantee that the CES function returns a maximal core. Instead, the algorithm includes two of the improvements proposed in (Delisle and Bacchus 2013): (i) the CES function computes several disjoin cores at each iteration and, (ii) it improves each of them by greedely increasing the lowest k_i value until the induced CSP becomes satisfiable. When it happens it undoes the last increment and returns it as a core. Our version of HS-WCSP produces results similar to what was reported in (Delisle and Bacchus 2013). For reference with respect to state-of-the-art WCSPs solvers, we also report results obtained with Toulbar2 v.1.1.1 with its default options (Hurley et al. 2016).

As can be seen, both forms of cost-function merging clearly outperform the original formulation solving more instances and more efficiently. Moreover, in this particular benchmark, both forms of cost-function merging also outperform Toulbar2 in all instances (except for 2 instances solved in under a second by all formulations). These results

	Original			Symbolic			Numerical			Toulbar2	
Inst.	lb	t	#its	lb	t	#its	lb	t	#its	lb	t
54	37	0.43	35	37	**0.03**	6	37	0.04	9	37	**0.03**
29	8059	1.23	56	8059	0.31	29	8059	0.14	16	8059	**0.03**
404	114	2.87	77	114	0.36	28	114	**0.25**	24	114	30.97
503	11113	1.01	32	11113	0.12	15	11113	**0.11**	13	11079	-
408	6225	-	131	6228	8.04	83	6228	**7.05**	96	4165	-
412	29345	-	69	32381	46.15	148	32381	**39.18**	146	22185	-
414	35979	-	108	38478	170.95	228	38478	**114.05**	318	27687	-
42	147050	-	207	155050	**38.28**	590	155050	75.66	985	96049	-
505	21245	-	100	21253	6.19	74	21253	**4.80**	107	14128	-
507	25870	-	82	27390	**72.62**	160	27390	127.87	329	17246	-
509	33429	-	78	36446	298.09	265	36446	**91.35**	274	28184	-
28	206103	-	35	245107	-	197	**247104**	-	259	150558	-

Table 2: Results on Spot5 instances. All instances are pre-processed to be VAC. CPU time in seconds. Time limit 1800 seconds (indicated as ”-”). When the time limit is not reached lb reports the optimum value. Otherwise, lb reports the obtained lower bound. The best CPU time is highlighted. When all solvers reach the time limit, the best lower bound is highlighted.

suggest that partitioning the set of functions into clusters being guided by tree-decompositions may pay-off. Comparing the two merging alternatives, we see that there is no clear winner. Numerical merging outperforms symbolic merging on almost all instances but the difference is not large. Note that in instance 28, where both approaches timed out, numerical merging also obtains a better lower bound.

Related Work

It is well-known that many combinatorial problems contain different forms of *symmetries* and that not addressing them may render solvers highly inefficient (Gent, Petrie, and Puget 2006). To overcome this problem, several works propose methods to eliminate them. Clearly, our notion of core interchangeability is just another form of problem symmetry and our two merging approaches can be seen as methods to eliminate them in the particular case of HS algorithms. However, it is worth emphasizing that our approach does not require interchangeability. As we showed in the experiments, it is likely to be effective even on clusters of conflicting (i.e, disagreeing) cost functions.

Our two merging methods are closely related to the so-called *abstract cores framework* (Berg, Bacchus, and Poole 2020) proposed in the MaxSAT context. The idea behind abstract cores is to group equally weighted soft clauses (the so-called abstraction sets) and to extract cores over new variables that correspond to sums over the original blocking variables (the so-called abstract cores). Each sum counts how many of the original blocking variables are falsified and, as a consequence, it is modelled as a cardinality constraint. Essentially, this is our symbolic merging. Our numerical merging goes one step further and drops the restriction of only clustering equally weighted clauses. Now each sum constraint counts the total sum of the coefficients (weights) of the original blocking variables and, as a consequence, it is modelled as a pseudo-Boolean constraint. In this sense, our work advances in the theoretical understanding of the differences arising from the usage of pseudo-Boolean constraints in contrast to cardinality constraints in the abstract cores framework.

Many heuristics to determine how to cluster clauses/functions may be appropriate. In (Berg, Bacchus, and Poole 2020), the heuristic dynamically finds meaningful clusters based on the core structure (which is not available up front). In contrast, we propose to use the tree-decomposition structure (which is available a-priori). Since the tree-decomposition does not change, our clustering is static and so our symbolic/numeric reformulation is done as a pre-process.

Conclusion and Future Work

Our long-term research aims at making the Hitting Set approach competitive with state-of-the-art alternatives for WCSP solving, as it happens in the very similar MaxSAT problem. In this paper we have shown that a naive application may be unfeasible even for very simple problems. We have characterized two forms of symmetry where HS-WCSP fails and proposed a method based on cost-function merging to overcome each one of them. Our theoretical analysis allowed to clearly identify the limitations of HS-WCSP and quantify the potential advantage of our approach. We claim that these forms of symmetry happen up to a certain level in real problems. Thus, we have introduced a heuristic method to identify those parts of a problem where cost-function merging is likely to be useful. Our method, that uses the notion of tree-decomposition, clearly improves over the basic hitting set approach and, in some cases, may be even competitive with state-of-the-art solvers.

Our work leaves many open lines of work. Just to name a few, we need to explore better implementations of the CES component and this can be done in different ways: improving the SAT encodings of the mergings or moving from SAT to other solving languages. We also need to explore alternatives to the use of tree-decomposition to identify appropriate clusters of cost-functions and, of course, extend our experiments to a wider set of benchmark instances.

Acknowledgements

The work of Javier Larrosa and Emma Rollon was supported by grant PID2021-122830OB-C43, funded by MCIN/AEI/ 10.13039/501100011033 and by "ERDF: A way of making Europe". The work of Conrado Martínez was supported by grant PID2020-112581GB-C21 (MOTION Project), funded by MCIN/AEI/10.13039/501100011033.

References

Allouche, D.; de Givry, S.; Katsirelos, G.; Schiex, T.; and Zytnicki, M. 2015. Anytime Hybrid Best-First Search with Tree Decomposition for Weighted CSP. In Pesant, G., ed., *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, volume 9255 of *Lecture Notes in Computer Science*, 12–29. Springer.

Bacchus, F.; Hyttinen, A.; Järvisalo, M.; and Saikko, P. 2018. Reduced Cost Fixing for Maximum Satisfiability. In Lang, J., ed., *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, 5209–5213. ijcai.org.

Bensana, E.; Lemaître, M.; and Verfaillie, G. 1999. Earth Observation Satellite Management. *Constraints An Int. J.*, 4(3): 293–299.

Berg, J.; Bacchus, F.; and Poole, A. 2020. Abstract Cores in Implicit Hitting Set MaxSat Solving. In Pulina, L.; and Seidl, M., eds., *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, 277–294. Springer.

Biere, A.; Fazekas, K.; Fleury, M.; and Heisinger, M. 2020. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling Entering the SAT Competition 2020. In Balyo, T.; Froleyks, N.; Heule, M.; Iser, M.; Järvisalo, M.; and Suda, M., eds., *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, 51–53. University of Helsinki.

Bodlaender, H. L.; and Koster, A. M. C. A. 2010. Treewidth computations I. Upper bounds. *Inf. Comput.*, 208(3): 259–275.

Cabon, B.; de Givry, S.; Lobjois, L.; Schiex, T.; and Warners, J. P. 1999. Radio Link Frequency Assignment. *Constraints An Int. J.*, 4(1): 79–89.

Cooper, M. C.; de Givry, S.; Sánchez-Fibla, M.; Schiex, T.; Zytnicki, M.; and Werner, T. 2010. Soft arc consistency revisited. *Artif. Intell.*, 174(7-8): 449–478.

Davies, J.; and Bacchus, F. 2013. Postponing Optimization to Speed Up MAXSAT Solving. In Schulte, C., ed., *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, 247–262. Springer.

Dechter, R. 2003. *Constraint processing*. Elsevier Morgan Kaufmann. ISBN 978-1-55860-890-0.

Delisle, E.; and Bacchus, F. 2013. Solving Weighted CSPs by Successive Relaxations. In Schulte, C., ed., *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, 273–281. Springer.

Gent, I. P.; Petrie, K. E.; and Puget, J. 2006. Symmetry in Constraint Programming. In Rossi, F.; van Beek, P.; and Walsh, T., eds., *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, 329–376. Elsevier.

Gupta, S. D.; Genc, B.; and O'Sullivan, B. 2021. Explanation in Constraint Satisfaction: A Survey. In Zhou, Z., ed., *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, 4400–4407. ijcai.org.

Hurley, B.; O'Sullivan, B.; Allouche, D.; Katsirelos, G.; Schiex, T.; Zytnicki, M.; and de Givry, S. 2016. Multi-language evaluation of exact solvers in graphical model discrete optimization. *Constraints An Int. J.*, 21(3): 413–434.

Joshi, S.; Martins, R.; and Manquinho, V. M. 2015. Generalized Totalizer Encoding for Pseudo-Boolean Constraints. In Pesant, G., ed., *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, volume 9255 of *Lecture Notes in Computer Science*, 200–209. Springer.

Kask, K.; Dechter, R.; Larrosa, J.; and Dechter, A. 2005. Unifying tree decompositions for reasoning in graphical models. *Artif. Intell.*, 166(1-2): 165–193.

Koller, D.; and Friedman, N. 2009. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press. ISBN 978-0-262-01319-2.

Larrosa, J.; and Schiex, T. 2004. Solving weighted CSP by maintaining arc consistency. *Artif. Intell.*, 159(1-2): 1–26.

Viricel, C.; de Givry, S.; Schiex, T.; and Barbe, S. 2018. Cost function network-based design of protein-protein interactions: predicting changes in binding affinity. *Bioinform.*, 34(15): 2581–2589.