Inferring Variable Conflicts for Local Search from High-Level Models

Magnus Ågren, Pierre Flener, and Justin Pearson
Department of Information Technology, Uppsala University
Box 337, SE – 751 05 Uppsala, Sweden
{agren,pierref,justin}@it.uu.se

February 24, 2006

Abstract

For efficiency reasons, neighbourhoods in local search algorithms are often shrunk by only considering moves modifying variables that actually contribute to the overall penalty. These are known as conflicting variables. This is a well-known technique for speeding up search. State-of-the-art solutions to, e.g., the progressive party problem exploit this with great success. We propose a way of automatically and incrementally measuring the conflict of a variable in a local search model and apply this to the set variables of models expressed in existential second-order logic extended with counting ($\exists$SOL$^+$). Furthermore, we show that this measure is lower-bounded by an intuitive conflict measure, and upper-bounded by the penalty of the model. We also demonstrate the usefulness of the approach by replacing a built-in global constraint by a modelled $\exists$SOL$^+$ version thereof, while still obtaining competitive results. This is especially attractive when a particular (global) constraint is not built in.

1 Introduction

In local search, it is often important to limit the size of the neighbourhood by only considering moves modifying conflicting variables, i.e., variables that actually contribute to the overall penalty. This concentrates the search to moves that may actually decrease this overall penalty. See, e.g., [3] (where the terminology critical variables is used) or [5].

In local search, it may happen that a suitable (possibly combinatorial, a.k.a. global) constraint for formulating the problem at hand is not available in the modelling language.

We aim at making modelling languages extensible, so as to overcome such a situation. The user should not have to design a suitable constraint, including incremental algorithms for calculating and maintaining its constraint penalties and variable conflicts, nor incur too much performance loss with respect to (with respect to) such a handcrafted constraint.

In this paper, we address the inference of variable conflicts from a high-level formulation of a non built-in constraint. Our key contributions are as follows:

- We propose a new way of automatically and incrementally measuring the conflict of a variable in a constraint model. We here apply the proposed definition to the *set* variables of constraint models expressed in a very expressive language, namely existential second-order logic extended with counting ($\exists$SOL$^+$). Our definition extends and differs significantly from the one for scalar variables in [7], but its principle can also be applied to scalar variables.
- We show that the proposed definition of the conflict of a variable x is lower-bounded by the intuitive target value, namely the maximum penalty decrease of the model that may be achieved by changing the value of x.

- We demonstrate the efficiency and usefulness of the approach by replacing a built-in global constraint of our local-search framework by a modelled $\exists\text{SOL}^+$ version thereof, while still obtaining competitive results in terms of run-time and quality of the solutions.

In the next section, we provide necessary background information about local search and $\exists\text{SOL}^+$. Then we present our definition of the conflict of a variable with respect to a constraint in $\exists\text{SOL}^+$ and prove that this value is lower-bounded by an intuitive value. After that, we demonstrate the usefulness of our approach by applying it to a real-life problem. Finally, we summarise our results and discuss related work.

2 Preliminaries

As usual, a *constraint satisfaction problem* (*CSP*) is a triple $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{X}$ is a finite set of variables, $\mathcal{D}$ is a finite set of domains, each $D_x \in \mathcal{D}$ containing the set of possible values for $x \in \mathcal{X}$, and $\mathcal{C}$ is a finite set of constraints, each $c \in \mathcal{C}$ being defined on a subset of $\mathcal{X}$ and specifying their valid combinations of values.

2.1 Set Variables and Local Search

We illustrate our results for CSPs where all variables are set variables [2]:

Definition 1 (Set Variable and its Universe) *Let $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ be a CSP. A variable $S \in \mathcal{X}$ is a* set variable *if its corresponding domain D_S is $2^{\mathcal{U}_S}$, where $\mathcal{U}_S$ is a finite set of values of some type, called the* universe *of S.*

In the context of this paper and without loss of generality, the set variables of a CSP all share a common universe $\mathcal{U}$.

In local search, an initial, possibly arbitrary, assignment of values to *all* the variables is maintained:

Definition 2 (Configuration) *Let $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ be a CSP. A* configuration *for P (or $\mathcal{X}$) is a total function $k : \mathcal{X} \to \bigcup_{x \in \mathcal{X}} D_x$.*

We will use $\mathcal{K}$ to denote the *set of all configurations* for a given CSP or set of variables, depending on the context.

Example 1 *Consider a CSP $P = \langle \{S, T\}, \{D_S, D_T\}, \{S \subset T\} \rangle$ where $D_S = D_T = 2^{\mathcal{U}}$ and $\mathcal{U} = \{a, b, c\}$. A configuration for P is given by $k(S) = \{a, b\}$ and $k(T) = \emptyset$, or equivalently by $k = \{S \mapsto \{a, b\}, T \mapsto \emptyset\}$.*

Local search iteratively makes a small change to the current configuration, upon examining the merits of many such changes until a solution is found or allocated resources have been exhausted. The configurations thus examined constitute the neighbourhood of the current configuration:

Definition 3 (Neighbourhood) *Let $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ be a CSP and let $k \in \mathcal{K}$. A* neighbourhood function *for P is a function $n : \mathcal{K} \to 2^{\mathcal{K}}$. The* neighbourhood *of P with respect to k and n is the set $n(k)$.*

The variable neighbourhood *for $x \in \mathcal{X}$ with respect to k is the subset of $\mathcal{K}$ reachable from k by only changing $k(x)$:*

$$n_x(k) = \{\ell \in \mathcal{K} \mid \forall y \in \mathcal{X} : y \neq x \to k(y) = \ell(y)\}$$

Note that the size of $n_x(k)$ is equal to the size of the domain of x. If x is a set variable, this is exponential in the size of $\mathcal{U}_x$. However, we use variable neighbourhoods to define concepts only, and do not enumerate in practice.

Example 2 *Consider P and k of Example 1 except that $\mathcal{U} = \{a, b\}$. The neighbourhood of P with respect to k and the neighbourhood function for P that moves an element from S to T is the set $\{k_a = \{S \mapsto \{b\}, T \mapsto \{a\}\}, k_b = \{S \mapsto \{a\}, T \mapsto \{b\}\}$. The variable neighbourhood for S with respect to k is the set $n_S(k) = \{k, k_1 = \{S \mapsto \emptyset, T \mapsto \emptyset\}, k_2 = \{S \mapsto \{a\}, T \mapsto \emptyset\}, k_3 = \{S \mapsto \{b\}, T \mapsto \emptyset\}\}$.*

Intuitively, the penalty of a constraint c is an estimate on how much it is violated with respect to the current configuration. Similarly, we say that a variable x is conflicting with respect to c and the current configuration if we may decrease the penalty of c by only changing the value of x.

Definition 4 (Penalty and Conflict) *Let $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ be a CSP and let $c \in \mathcal{C}$. A* penalty function *of c is a function $penalty(c) : \mathcal{K} \to \mathbb{N}$ such that (such that) $penalty(c)(k) = 0$ if and only if c is satisfied with respect to k. The* penalty *of c with respect to k is $penalty(c)(k)$. The* penalty *of P with respect to k is the sum $\sum_{c \in \mathcal{C}} penalty(c)(k)$.*

A conflict function *of c is a function $conflict(c) : \mathcal{X} \times \mathcal{K} \to \mathbb{N}$ such that if $conflict(c)(x, k) = 0$ then $\forall \ell \in n_x(k) : penalty(c)(k) \leq penalty(c)(\ell)$. The* conflict *of x with respect to c and k is $conflict(c)(x, k)$. The* conflict *of x with respect to P and k is the sum $\sum_{c \in \mathcal{C}} conflict(c)(x, k)$.*

Example 3 *Consider once again P from Example 1 and let the penalty and conflict functions of $S \subset T$ be defined by:*

$$penalty(S \subset T)(k) = |k(S) \setminus k(T)| + \begin{cases} 1, & \text{if } k(T) \subseteq k(S) \\ 0, & \text{otherwise} \end{cases}$$

and

$$conflict(S \subset T)(x, k) = |k(S) \setminus k(T)| + \begin{cases} 1, & \text{if } x = T \text{ and } k(T) \subseteq k(S) \\ 0, & \text{otherwise} \end{cases}$$

respectively. Now, the penalties of P with respect to k of Example 1 and k_a of Example 2 are respectively $penalty(S \subset T)(k) = 3$ and $penalty(S \subset T)(k_a) = 1$. Indeed, we may satisfy P with respect to k by, e.g., adding the three values a, b, and c to T and with respect to k_a by, e.g., removing the single value b from S.

The conflicts of S and T with respect to P and k are $conflict(S \subset T)(S, k) = 2$ and $conflict(S \subset T)(T, k) = 3$, respectively. Indeed, by changing the value of S, we may decrease the penalty of P by two (by removing the values a and b). Similarly, by changing the value of T, we may decrease the penalty of P by three (by adding the values a, b, and c).

We are usually interested in the *maximum* amount of penalty decrease possible to be able to rank different variables based on their conflict. This is naturally expressed by the notion of abstract conflict, defined as follows.

Definition 5 (Abstract Conflict) *Let c be a constraint defined on the variables $\mathcal{X}$. The* abstract conflict function *of c is a function $abstractConflict(c) : \mathcal{X} \times \mathcal{K} \to \mathbb{N}$ such that:*

$$abstractConflict(c)(x, k) = \max\{penalty(c)(k) - penalty(c)(\ell) \mid \ell \in n_x(k)\}$$

The abstract conflict *of $x \in \mathcal{X}$ with respect to c and a configuration $k \in \mathcal{K}$ is $abstractConflict(c)(x, k)$.*

Note that the abstract conflict of a variable is never negative.

Example 4 *The conflict function of $S \subset T$ defined in Example 3 gives the abstract conflicts of variables.*

We now prove that the abstract conflict function is correct with respect to Definition 4.

Proposition 1 *Let c be a constraint. Then $abstractConflict(c)$ is a conflict function.*

Proof. Let c be a constraint defined on the set of variables $\mathcal{X}$, let k be a configuration for $\mathcal{X}$, and let $x \in \mathcal{X}$. Assume that $abstractConflict(c)(x, k) = 0$. Then $\max\{penalty(c)(k) - penalty(c)(\ell) \mid \ell \in n_x(k)\} = 0$ and hence $\forall \ell \in n_x(k) : penalty(c)(k) \leq penalty(c)(\ell)$. □

$$
\begin{array}{l}
\langle \mathit{Constraint} \rangle ::= (\underline{\exists}\ \langle S \rangle)^+\ \langle \mathit{Formula} \rangle \\
\\
\langle \mathit{Formula} \rangle ::= \underline{(}\langle \mathit{Formula} \rangle\underline{)} \\
\qquad\qquad \mid\ (\underline{\forall} \mid \underline{\exists})\langle x \rangle\ \langle \mathit{Formula} \rangle \\
\qquad\qquad \mid\ \langle \mathit{Formula} \rangle\ (\underline{\wedge} \mid \underline{\vee})\ \langle \mathit{Formula} \rangle \\
\qquad\qquad \mid\ \langle \mathit{Literal} \rangle \\
\\
\langle \mathit{Literal} \rangle ::= \langle x \rangle\ (\underline{\in} \mid \underline{\notin})\ \langle S \rangle \\
\qquad\qquad \mid\ \langle x \rangle\ (\underline{<} \mid \underline{\leq} \mid \underline{=} \mid \underline{\neq} \mid \underline{\geq} \mid \underline{>})\ \langle y \rangle \\
\qquad\qquad \mid\ \underline{|}\langle S \rangle\underline{|}\ (\underline{<} \mid \underline{\leq} \mid \underline{=} \mid \underline{\neq} \mid \underline{\geq} \mid \underline{>})\ \langle a \rangle
\end{array}
$$

Figure 1: The BNF grammar for $\exists\mathrm{SOL}^+$ where terminal symbols are underlined. The non-terminal symbol $\langle S \rangle$ denotes an identifier for a bound set variable S such that $S \subseteq \mathcal{U}$, where $\mathcal{U}$ is the common universe, while $\langle x \rangle$ and $\langle y \rangle$ denote identifiers for bound variables x and y such that $x, y \in \mathcal{U}$, and $\langle a \rangle$ denotes a natural number constant a.

2.2 Existential Second-Order Logic

We use existential second-order logic extended with counting for modelling set constraints [1]. This language (referred to by $\exists\mathrm{SOL}^+$ and shown in BNF in Figure 1) is very expressive as it captures at least the complexity class NP [4]. A constraint in $\exists\mathrm{SOL}^+$ is of the form $\exists S_1 \cdots \exists S_n \phi$, i.e., a sequence of existentially quantified set variables, ranging over the power set of an implicit common universe $\mathcal{U}$, and constrained by a logical formula ϕ.

As a running example, consider the constraint $S \subset T$ of Example 1. This may be expressed by the $\exists\mathrm{SOL}^+$ formula:

$$\exists S \exists T((\forall x(x \notin S \vee x \in T)) \wedge (\exists x(x \in T \wedge x \notin S))) \tag{1}$$

Note that some of the usual connectives (such as $\neg$, $\rightarrow$, and $\leftrightarrow$) are not part of the language. This is only due to the way we define the penalty and conflict functions for $\exists\mathrm{SOL}^+$ below and does not pose any limitations on the expressiveness of the language: Any formula including those connectives may be normalised into a formula without them by standard transformations. In fact, a user would probably express the sub-formula $x \notin S \vee x \in T$ of (1) by $x \in S \rightarrow x \in T$, which would then (internally) be transformed.

Given $\mathcal{F} \in \exists\mathrm{SOL}^+$, we use $\mathit{vars}(\mathcal{F})$ to denote the set variables in $\mathcal{F}$. For example, if $\mathcal{F}$ is (1) above, then $\mathit{vars}(\mathcal{F}) = \{S, T\}$.

In order to use $\exists\mathrm{SOL}^+$ constraints with local search, a penalty function must be defined. This was done in [1], as repeated below.

Definition 6 (Penalty of a Formula) *Let $\mathcal{F} \in \exists\mathrm{SOL}^+$ and let k be a configuration for $\mathit{vars}(\mathcal{F})$. The* penalty *of $\mathcal{F}$ with respect to k is defined by:*

(a) $\mathit{penalty}(\exists S_1 \cdots \exists S_n \phi)(k) = \mathit{penalty}(\phi)(k)$
(b) $\mathit{penalty}(\forall x \phi)(k) = \sum_{u \in \mathcal{U}} \mathit{penalty}(\phi)(k \cup \{x \mapsto u\})$
(c) $\mathit{penalty}(\exists x \phi)(k) = \min\{\mathit{penalty}(\phi)(k \cup \{x \mapsto u\} \mid u \in \mathcal{U}\})$
(d) $\mathit{penalty}(\phi \wedge \psi)(k) = \mathit{penalty}(\phi)(k) + \mathit{penalty}(\psi)(k)$
(e) $\mathit{penalty}(\phi \vee \psi)(k) = \min\{\mathit{penalty}(\phi)(k), \mathit{penalty}(\psi)(k)\}$
(f) $\mathit{penalty}(|S| \leq c)(k) = \begin{cases} 0, \text{ if } |k(S)| \leq c \\ |k(S)| - c, \text{ otherwise} \end{cases}$
(g) $\mathit{penalty}(x \in S)(k) = \begin{cases} 0, \text{ if } k(x) \in k(S) \\ 1, \text{ otherwise} \end{cases}$
(h) $\mathit{penalty}(x \leq y)(k) = \begin{cases} 0, \text{ if } k(x) \leq k(y) \\ 1, \text{ otherwise} \end{cases}$

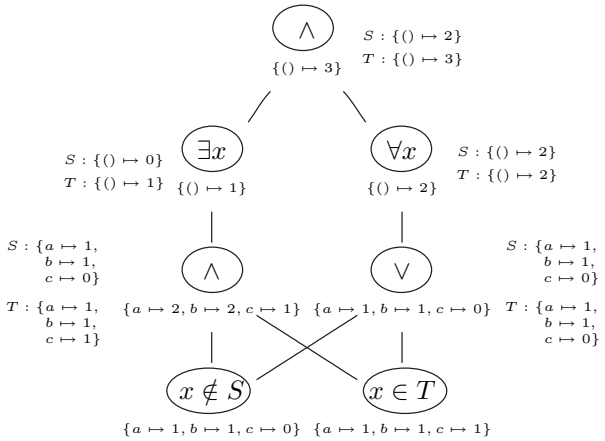

Figure 2: Penalty and conflict tree of (1).

In the definition above, for subformulas of the form $x \Diamond y$, $|S| \Diamond c$, and $x \triangle S$, only the cases where $\Diamond \in \{\leq\}$ and $\triangle \in \{\in\}$ are shown. The other cases are defined similarly. Note that the penalty of a formula is never negative.

Example 5 *Recall $k = \{S \mapsto \{a, b\}, T \mapsto \emptyset\}$ of Example 1 and let $\mathcal{F}$ be (1). According to Definition 6, penalty$(\mathcal{F})(k) = 3$, which is meaningful since we may satisfy $\mathcal{F}$ by, e.g., adding the three values a, b, and c to $k(T)$.*

Figure 2 shows the *penalty tree* of (1) with respect to k of Example 5. Each node in the tree is associated with the penalty of the corresponding sub-formula in (1) with respect to k (shown below the nodes in the tree; the reader should ignore the annotations beside the nodes until the next section). Hence, the root node is associated with the penalty of (1) with respect to k. Note that, for sub-formulas containing bound first-order variables, the corresponding nodes are associated with a penalty with respect to each possible combination of values for those variables. Figure 2 also shows an important concept in our implementation of the $\exists\text{SOL}^+$ framework. In the penalty tree of a formula, each node is represented at most once. For example, if two sub-formulas ϕ and ψ of $\mathcal{F}$ both contain a sub-formula γ, then there will be only one occurrence of the penalty tree of γ in the penalty tree of $\mathcal{F}$. This is illustrated in Figure 2 by, e.g., the sub-formula $x \notin S$ only occurring once.

3 Variable Conflicts of an $\exists\text{SOL}^+$ Formula

We now define the conflict function of $\exists\text{SOL}^+$ formulas. Similarly to the penalty function of Definition 6, it is important to stress that this calculation is totally generic and automatable, as it is based only on the syntax of the formula and the semantics of the quantifiers, connectives, and relational operators of $\exists\text{SOL}^+$, but not on the intended semantics of the formula.

Defining the conflict of a variable with respect to an $\exists\text{SOL}^+$ constraint c and a configuration k is not a trivial exercise that reduces to applying exactly the same rules as for the penalty of c with respect to k. Matters are complicated by the fact that not every variable occurs in every sub-formula of c. The following example illustrates some of these pitfalls on scalar variables; they generalise to set variables by replacing its $z = a$ constraints by $|S| = a$ constraints.

Example 6 *Assume that the penalty of a numeric constraint $z = a$ with respect to a configuration k is the under/overflow of a compared to $k(z)$, that is penalty$(z = a)(k) = |k(z) - a|$. Assume that conflict$(z = a)(z, k) =$ penalty$(z = a)(k)$. Consider the disjunctive constraint $c = (x = 4 \vee y = 3)$.*

First take the configuration $k_1 = \{x \mapsto 3, y \mapsto 3\}$. The second disjunct is satisfied and penalty$(c)(k_1) = 0$ indeed, so the conflicts of both x and y with respect to c and k_1 should be

0 *as well. This suggests that the conflict of* x *with respect to* c *and* k_1 *cannot be obtained by applying the* min *operator to the variable conflicts of* x *with respect to the disjuncts of* c *(where* x *occurs).*

Now take the configuration $k_2 = \{x \mapsto 2, y \mapsto 6\}$. *None of the disjuncts is satisfied and* $penalty(c)(k_2) = 2$ *(the minimum of the under/overflows of* 2 *and* 3 *in the two disjuncts), which is not* 0 *indeed, so the conflicts of* x *and* y *with respect to* c *and* k_2 *should both* not *be* 0 *but upper-bounded by* 2. *This suggests that the conflict of* y *with respect to* c *and* k_2 *cannot be obtained by applying the* min *operator to the variable conflicts of* y *with respect to the disjuncts of* c *(where* y *occurs).*

However, matters are even more complicated than that. The following example argues that, even in the absence of the problem above with non-occurring variables, there is another problem with using the min operator to combine the variable conflicts with respect to the disjuncts to get the corresponding variable conflicts with respect to the disjunction, as for penalties.

Example 7 *Assume the same penalty and conflict functions of an equality constraint as in Example 6 and consider the constraint* $c = (x = 5 \vee (y = 3 \wedge x = 4))$.

Let $k = \{x \mapsto 4, y \mapsto 4\}$. *Then* $penalty(c)(k) = 1$ *and, assuming that* $conflict(\ell \wedge r)(x, k) = \sum\{conflict(d)(x, k) \mid d \in \{\ell, r\}$ *and* x *occurs in* $d\}$, *we have* $conflict(y = 3 \wedge x = 4)(x, k) = 0$. *Observe now that* x *occurs in both of the disjuncts of* c. *Hence, if we obtain the conflict of* x *with respect to* c *and* k *by applying the min operator to the variable conflicts of* x *with respect to the disjuncts of* c *(where* x *occurs), we have that* $conflict(c)(x, k) = \min\{conflict(x = 5)(x, k), conflict(y = 3 \wedge x = 4)(x, k)\} = \min\{1, 0\} = 0$. *This has the meaning that it is recommended to try and satisfy* c *by not changing* x *with respect to* k. *However, we may satisfy* c *by changing* $k(x)$ *to* 5.

We thus propose the following novel definition of the conflict of a variable with respect to a constraint and a configuration.

Definition 7 (Conflict of a Formula) *Let* $\mathcal{F} \in \exists\text{SOL}^+$, *let* $S \in vars(\mathcal{F})$, *and let* k *be a configuration for* $vars(\mathcal{F})$. *The* conflict *of* S *with respect to* $\mathcal{F}$ *and* k *is defined by:*

(a) $conflict(\exists S_1 \cdots \exists S_n \phi)(S, k) = conflict(\phi)(S, k)$
(b) $conflict(\forall x \phi)(S, k) = \sum_{u \in \mathcal{U}} conflict(\phi)(S, k \cup \{x \mapsto u\})$
(c) $conflict(\exists x \phi)(S, k) = \max\{0\} \cup \{penalty(\exists x \phi)(k) - (penalty(\phi)(k \cup \{x \mapsto u\}) - conflict(\phi)(S, k \cup \{x \mapsto u\})) \mid u \in \mathcal{U}\}$
(d) $conflict(\phi \wedge \psi)(S, k) = \sum\{conflict(\gamma)(S, k) \mid \gamma \in \{\phi, \psi\} \wedge S \in vars(\gamma)\}$
(e) $conflict(\phi \vee \psi)(S, k) = \max\{0\} \cup \{penalty(\phi \vee \psi)(k) - (penalty(\gamma)(k) - conflict(\gamma)(S, k)) \mid \gamma \in \{\phi, \psi\} \wedge S \in vars(\gamma)\}$
(f) $conflict(|S| \leq c)(S, k) = penalty(|S| \leq c)(k)$
(g) $conflict(x \in S)(S, k) = penalty(x \in S)(k)$

As in Definition 6, we only show cases for subformulas of the form $|S| \Diamond c$ *and* $x \triangle S$ *where* $\Diamond \in \{\leq\}$ *and* $\triangle \in \{\in\}$.

Note that the conflict of a set variable S with respect to a formula $\mathcal{F}$ and a configuration k is undefined whenever $S \notin vars(\mathcal{F})$ and non-negative otherwise. This definition is specific to set variables and set constraints (modelled in $\exists\text{SOL}^+$), but its principle also applies to scalar variables and constraints. It correctly (with respect to our stated intuition) handles all the tricky cases in Examples 6 and 7.

Example 8 *Recall once again* $k = \{S \mapsto \{a, b\}, T \mapsto \emptyset\}$ *of Example 1 and let* $\mathcal{F}$ *be (1). According to Definition 7,* $conflict(\mathcal{F})(S, k) = 2$ *and* $conflict(\mathcal{F})(T, k) = 3$, *which is meaningful since we may satisfy* $\mathcal{F}$ *by, e.g., adding the* three *values* a, b, *and* c *to* $k(T)$, *or by removing the* two *values* a *and* b *from* $k(S)$ *and adding any value to* $k(T)$.

In the *conflict tree* of Figure 2, each node representing a sub-formula ϕ in (1) is associated with the conflicts of the set variables in $vars(\phi)$ with respect to ϕ and k (shown to the left/right of the nodes in the tree). Hence, the root node is associated with the conflicts of the variables of (1) with respect to k. Similarly to penalties, for sub-formulas containing bound first-order variables, the corresponding nodes are associated with a conflict for their set variables with respect to each possible combination of values for those first-order variables.

Let us now prove that any $\exists\mathrm{SOL}^+$ conflict is lower-bounded by the abstract conflict and upper-bounded by the penalty. In order to simplify the proofs, we assume that a formula of the form $\forall x\phi$ is replaced by the equivalent formula $(\phi_1 \wedge (\phi_2 \wedge \cdots \wedge (\phi_{n-1} \wedge \phi_n) \cdots))$, where ϕ_i denotes the formula ϕ in which any occurrence of x is replaced by u_i and where $\mathcal{U} = \{u_1, \ldots, u_n\}$ with $n \geq 2$. Similarly, a formula of the form $\exists x\phi$ is replaced by the equivalent formula $(\phi_1 \vee (\phi_2 \vee \cdots \vee (\phi_{n-1} \vee \phi_n) \cdots))$.

First of all, we need to prove the following lemma.

Lemma 1 *Let $\mathcal{F} \in \exists\mathrm{SOL}^+$ be of the form $\phi \nabla \psi$, where $\nabla \in \{\wedge, \vee\}$, let k be a configuration for $vars(\mathcal{F})$, and let $S \in vars(\mathcal{F})$. If $S \notin vars(\phi)$ then $abstractConflict(\phi \nabla \psi)(S,k) \leq abstractConflict(\psi)(S,k)$.*

Proof. Assume that $\nabla = \wedge$. Then $abstractConflict(\phi \wedge \psi)(S,k) = \max\{penalty(\phi \wedge \psi)(k) - penalty(\phi\wedge\psi)(\ell) \mid \ell \in n_S(k)\} = \max\{penalty(\phi)(k) - penalty(\phi)(\ell) + penalty(\psi)(k) - penalty(\psi)(\ell) \mid \ell \in n_S(k)\} = e$. Since for each $T \neq S$ it holds that $\forall \ell \in n_S(k) : \ell(T) = k(T)$, and since $S \notin vars(\phi)$, we have that $\forall \ell \in n_S(k) : penalty(\phi)(k) = penalty(\phi)(\ell)$. Then the term $penalty(\phi)(k) - penalty(\phi)(\ell)$ of the expression e above is equal to 0 for any $\ell \in n_S(k)$. Hence, $e = \max\{penalty(\psi)(k) - penalty(\psi)(\ell) \mid \ell \in n_S(k)\} = abstractConflict(\psi)(S,k)$ and hence $e \leq abstractConflict(\psi)(S,k)$.

Assume now that $\nabla = \vee$. Then

$$
\begin{aligned}
&abstractConflict(\phi \vee \psi)(S,k) = \\
&\quad \max\{penalty(\phi \vee \psi)(k) - penalty(\phi \vee \psi)(\ell) \mid \ell \in n_S(k)\} = \\
&\quad \max\{\underbrace{\min\{\overbrace{penalty(\phi)(k)}^{\alpha}, \overbrace{penalty(\psi)(k)}^{\beta}\}}_{A} - \\
&\qquad \underbrace{\min\{\overbrace{penalty(\phi)(\ell)}^{\gamma}, \overbrace{penalty(\psi)(\ell)}^{\delta}\}}_{B} \mid \ell \in n_S(k)\} = e.
\end{aligned}
$$

Consider now the case where $A = \alpha$ and $B = \gamma$ above for an $\ell \in n_S(k)$ that maximises $A - B$. Then $e = \max\{penalty(\phi)(k) - penalty(\phi)(\ell) \mid \ell \in n_S(k)\} = 0$ since $\forall \ell \in n_S(k) : penalty(\phi)(k) = penalty(\phi)(\ell)$, as argued above, and hence $e \leq abstractConflict(\psi)(S,k)$.

Consider now the case where $A = \beta$ and $B = \delta$ above for an $\ell \in n_S(k)$ that maximises $A - B$. Then $e = \max\{penalty(\psi)(k) - penalty(\psi)(\ell) \mid \ell \in n_S(k)\} = abstractConflict(\psi)(S,k)$ and hence $e \leq abstractConflict(\psi)(S,k)$.

Consider now the case where $A = \alpha$ (i.e., $penalty(\phi)(k) \leq penalty(\psi)(k)$) and $B = \delta$ above for an $\ell \in n_S(k)$ that maximises $A - B$. Then $e = \max\{penalty(\phi)(k) - penalty(\psi)(\ell) \mid \ell \in n_S(k)\}$. Since $penalty(\phi)(k) \leq penalty(\psi)(k)$ we have that $e \leq \max\{penalty(\psi)(k) - penalty(\psi)(\ell) \mid \ell \in n_S(k)\} = abstractConflict(\psi)(S,k)$.

Consider now the case where $A = \beta$ (i.e., $penalty(\phi)(k) \geq penalty(\psi)(k)$) and $B = \gamma$ above for an $\ell \in n_S(k)$ that maximises $A - B$. Then $e = \max\{penalty(\psi)(k) - penalty(\phi)(\ell) \mid \ell \in n_S(k)\}$. Since $\forall \ell \in n_S(k) : penalty(\phi)(k) = penalty(\phi)(\ell)$, as argued above, we have that $e = penalty(\psi)(k) - penalty(\phi)(k)$. Now since $penalty(\phi)(k) \geq penalty(\psi)(k)$ we have that $e \leq 0 \leq abstractConflict(\psi)(S,k)$. □

Proposition 2 *Let $\mathcal{F} \in \exists\mathrm{SOL}^+$, let k be a configuration for $vars(\mathcal{F})$, and let $S \in vars(\mathcal{F})$. Then $conflict(\mathcal{F})(S,k) \geq abstractConflict(\mathcal{F})(S,k)$.*

Proof. The proof is by structural induction on $\mathcal{F}$. The result holds for the base cases (f) and (g). For case (a), the result follows directly by induction from the definition.

Case $\mathcal{F} = \phi \wedge \psi$. Assume that $S \in \mathit{vars}(\phi) \cap \mathit{vars}(\psi)$. We have that $\mathit{abstractConflict}(\phi \wedge \psi)(S,k) = \max\{(\mathit{penalty}(\phi)(k) + \mathit{penalty}(\psi)(k)) - (\mathit{penalty}(\phi)(\ell) + \mathit{penalty}(\psi)(\ell)) \mid \ell \in n_S(k)\} = \max\{\mathit{penalty}(\phi)(k) - \mathit{penalty}(\phi)(\ell) + \mathit{penalty}(\psi)(k) - \mathit{penalty}(\psi)(\ell) \mid \ell \in n_S(k)\} = e$. Now, to see that $e \leq \max\{\mathit{penalty}(\phi)(k) - \mathit{penalty}(\phi)(\ell') \mid \ell' \in n_S(k)\} + \max\{\mathit{penalty}(\psi)(k) - \mathit{penalty}(\psi)(\ell'') \mid \ell'' \in n_S(k)\} = f$ we pick the $\ell \in n_S(k)$ that maximises

$$\mathit{penalty}(\phi)(k) - \mathit{penalty}(\phi)(\ell) + \mathit{penalty}(\psi)(k) - \mathit{penalty}(\psi)(\ell).$$

For that ℓ we have that either it maximises both $\mathit{penalty}(\phi)(k) - \mathit{penalty}(\phi)(\ell)$ and $\mathit{penalty}(\psi)(k) - \mathit{penalty}(\psi)(\ell)$, or there exist $\ell', \ell'' \in n_S(k)$ that make the sum of those expressions larger than e. Now $f = \mathit{abstractConflict}(\phi)(S,k) + \mathit{abstractConflict}(\psi)(S,k)$. By induction it then follows that $f \leq \mathit{conflict}(\phi)(S,k) + \mathit{conflict}(\psi)(S,k) = \mathit{conflict}(\phi \wedge \psi)(S,k)$.

Assume now that $S \notin \mathit{vars}(\phi)$. Then $\mathit{abstractConflict}(\phi \wedge \psi)(S,k) = \mathit{abstractConflict}(\psi)(S,k)$, by Lemma 1, and $\mathit{conflict}(\phi \wedge \psi)(S,k) = \mathit{conflict}(\psi)(S,k)$, by definition. Hence, the proposition follows by induction.

Case $\mathcal{F} = \phi \vee \psi$. Assume that $S \in \mathit{vars}(\phi) \cap \mathit{vars}(\psi)$. We have that

$$\begin{aligned}&\mathit{abstractConflict}(\phi \vee \psi)(S,k) = \\ &\max\{\underbrace{\min\{\overbrace{\mathit{penalty}(\phi)(k)}^{\alpha}, \overbrace{\mathit{penalty}(\psi)(k)}^{\beta}\}}_{A} - \underbrace{\min\{\overbrace{\mathit{penalty}(\phi)(\ell)}^{\gamma}, \overbrace{\mathit{penalty}(\psi)(\ell)}^{\delta}\}}_{B} \mid \\ &\quad \ell \in n_S(k)\} = e.\end{aligned}$$

Consider now the case where $A = \alpha$ (i.e., $\mathit{penalty}(\phi)(k) \leq \mathit{penalty}(\psi)(k)$) and $B = \gamma$ above for an $\ell \in n_S(k)$ that maximises $A - B$. Then we have that $e = \max\{\mathit{penalty}(\phi)(k) - \mathit{penalty}(\phi)(\ell) \mid \ell \in n_S(k)\} = \mathit{abstractConflict}(\phi)(S,k)$. Now $\mathit{conflict}(\phi \vee \psi)(S,k) = \max\{0, \mathit{penalty}(\phi \vee \psi)(k) - \mathit{penalty}(\phi)(k) + \mathit{conflict}(\phi)(S,k), \mathit{penalty}(\phi \vee \psi)(k) - \mathit{penalty}(\psi)(k) + \mathit{conflict}(\psi)(S,k)\}$ which simplifies into $\max\{0, \mathit{conflict}(\phi)(S,k), \mathit{penalty}(\phi)(k) - \mathit{penalty}(\psi)(k) + \mathit{conflict}(\psi)(S,k)\} = f$. Assume now that $f = \mathit{conflict}(\phi)(S,k)$, then we have that $\mathit{conflict}(\phi)(S,k) \geq \mathit{abstractConflict}(\phi)(S,k) = e$ by induction and we are done. Assume instead that $f = \mathit{penalty}(\phi)(k) - \mathit{penalty}(\psi)(k) + \mathit{conflict}(\psi)(S,k)$ or $f = 0$. Then $f \geq \mathit{conflict}(\phi)(S,k)$ and since

$$\mathit{conflict}(\phi)(S,k) \geq \mathit{abstractConflict}(\phi)(S,k) = e$$

by induction, we are done.

Consider now the case where $A = \beta$ and $B = \delta$ above for an $\ell \in n_S(k)$ that maximises $A - B$. This case holds by being symmetric to the previous one.

Consider now the case where $A = \alpha$ (i.e., $\mathit{penalty}(\phi)(k) \leq \mathit{penalty}(\psi)(k)$) and $B = \delta$ above for an $\ell \in n_S(k)$ that maximises $A - B$. Then we have that $e = \max\{\mathit{penalty}(\phi)(k) - \mathit{penalty}(\psi)(\ell) \mid \ell \in n_S(k)\}$. Now $\mathit{conflict}(\phi \vee \psi)(S,k) = \max\{0, \mathit{penalty}(\phi \vee \psi)(k) - \mathit{penalty}(\phi)(k) + \mathit{conflict}(\phi)(k), \mathit{penalty}(\phi \vee \psi)(k) - \mathit{penalty}(\psi)(k) + \mathit{conflict}(\psi)(k)\}$, which simplifies into $\max\{0, \mathit{conflict}(\phi)(S,k), \mathit{penalty}(\phi)(k) - \mathit{penalty}(\psi)(k) + \mathit{conflict}(\psi)(S,k)\} = f$. By induction $\max\{\mathit{penalty}(\psi)(k) - \mathit{penalty}(\psi)(\ell) \mid \ell \in n_S(k)\} \leq \mathit{conflict}(\psi)(S,k)$, which means that for all $\ell \in n_S(k)$ we have that $\mathit{penalty}(\psi)(k) - \mathit{penalty}(\psi)(\ell) \leq \mathit{conflict}(\psi)(S,k)$. There are three possible values for f: $\mathit{penalty}(\phi)(k) - \mathit{penalty}(\psi)(k) + \mathit{conflict}(\psi)(S,k)$, $\mathit{conflict}(\phi)(S,k)$, and 0, but since $\mathit{conflict}(\phi)(S,k)$ is non-negative we can ignore the $f = 0$ case. In the first case we take the induction hypothesis $\forall \ell \in n_S(k)(\mathit{penalty}(\psi)(k) - \mathit{penalty}(\psi)(\ell) \leq \mathit{conflict}(\psi)(S,k))$ and add to both sides of the inequality the term $\mathit{penalty}(\phi)(k) - \mathit{penalty}(\psi)(k)$, which gives $\forall \ell \in n_S(k)(\mathit{penalty}(\psi)(k) - \mathit{penalty}(\psi)(\ell) + \mathit{penalty}(\phi)(k) - \mathit{penalty}(\psi)(k) \leq \mathit{penalty}(\phi)(k) - \mathit{penalty}(\psi)(k) + \mathit{conflict}(\psi)(S,k))$. Cancelling terms gives $\forall \ell \in n_S(k)(\mathit{penalty}(\phi)(k) - \mathit{penalty}(\psi)(\ell) \leq \mathit{penalty}(\phi)(k) - \mathit{penalty}(\psi)(k) + \mathit{conflict}(\psi)(S,k) = f)$, which implies (by the current assumptions) that $\mathit{abstractConflict}(\phi \vee \psi)(S,k) \leq \mathit{conflict}(\phi \vee \psi)(S,k)$. Last, suppose that $f = \mathit{conflict}(\phi)(S,k)$. Then, since f is the maximum, we have that $\mathit{conflict}(\phi)(S,k) \geq \mathit{penalty}(\phi)(k) - \mathit{penalty}(\psi)(k) + \mathit{conflict}(\psi)(S,k)$, but we have already shown that $\mathit{penalty}(\phi)(k) - \mathit{penalty}(\psi)(k) + \mathit{conflict}(\psi)(S,k) \geq \mathit{abstractConflict}(\phi \vee \psi)(S,k)$.

Consider now the case where $A = \beta$ and $B = \gamma$ above for an $\ell \in n_S(k)$ that maximises $A - B$. This case holds by being symmetric to the previous one.

Finally, as with the $\wedge$ case, Lemma 1 can be applied to consider the case where $S \notin vars(\psi)$. □

Proposition 3 *Let $\mathcal{F} \in \exists\text{SOL}^+$, let k be a configuration for $vars(\mathcal{F})$, and let $S \in vars(\mathcal{F})$. Then $conflict(\mathcal{F})(S,k) \leq penalty(\mathcal{F})(k)$.*

Proof. The proof is by structural induction on $\mathcal{F}$. The result holds for the base cases (f) and (g). For case (a), the result follows directly by induction from the definition.

Case $\mathcal{F} = \phi \wedge \psi$. Assume that $S \in vars(\phi) \cap vars(\psi)$. We have that $conflict(\phi \wedge \psi)(S,k) = conflict(\phi)(S,k) + conflict(\psi)(S,k)$ and that $penalty(\phi \wedge \psi)(k) = penalty(\phi)(k) + penalty(\psi)(k)$ by definition. Since $conflict(\phi)(S,k) \leq penalty(\phi)(k)$ and $conflict(\psi)(S,k) \leq penalty(\psi)(k)$ by induction, we must have that $conflict(\phi)(S,k) + conflict(\psi)(S,k) \leq penalty(\phi)(k) + penalty(\psi)(k)$ and hence $conflict(\phi \wedge \psi)(k) \leq penalty(\phi \wedge \psi)(k)$.

Assume now that $S \notin vars(\phi)$. We then have that $conflict(\phi \wedge \psi)(S,k) = conflict(\psi)(S,k)$ and that $penalty(\phi \wedge \psi)(k) = penalty(\phi)(k) + penalty(\psi)(k)$ by definition. Since $conflict(\psi)(S,k) \leq penalty(\psi)(S,k)$ by induction and since $penalty(\phi)(k) \geq 0$, we have that $conflict(\psi)(S,k) \leq penalty(\phi)(k) + penalty(\psi)(k)$ and hence $conflict(\phi \wedge \psi)(k) \leq penalty(\phi \wedge \psi)(k)$.

Case $\mathcal{F} = \phi \vee \psi$. Assume that $S \in vars(\phi) \cap vars(\psi)$. By definition, we have that

$$\begin{aligned} conflict(\phi \vee \psi)(S,k) = \max\{0, &penalty(\phi \vee \psi)(k) - (penalty(\phi)(k) - conflict(\phi)(S,k)), \\ &penalty(\phi \vee \psi)(k) - (penalty(\psi)(k) - conflict(\psi)(S,k))\}. \end{aligned}$$

The result follows directly when $conflict(\phi \vee \psi)(S,k) = 0$ since $penalty(\phi \vee \psi)(k) \geq 0$. Consider the case when $conflict(\phi \vee \psi)(S,k) = penalty(\phi \vee \psi)(k) - (penalty(\phi)(k) - conflict(\phi)(S,k))$. It is enough to show that $penalty(\phi)(k) - conflict(\phi)(S,k) \geq 0$. Since $conflict(\phi)(S,k) \leq penalty(\phi)(k)$ by induction, this must hold. The case when $conflict(\phi \vee \psi)(S,k) = penalty(\phi \vee \psi)(k) - (penalty(\psi)(k) - conflict(\psi)(S,k))$ follows by similar reasoning.

Assume now that $S \notin vars(\phi)$. Then we have that $conflict(\phi \vee \psi)(S,k) = \max\{0, penalty(\phi \vee \psi)(k) - (penalty(\psi)(S,k) - conflict(\psi)(S,k))\}$ by definition. The result follows directly when $conflict(\phi \vee \psi)(S,k) = 0$ since $penalty(\phi \vee \psi)(k) \geq 0$. Consider the case when $conflict(\phi \vee \psi)(S,k) = penalty(\phi \vee \psi)(k) - (penalty(\psi)(k) - conflict(\psi)(S,k))$. It is enough to show that $penalty(\psi)(k) - conflict(\psi)(S,k) \geq 0$. Since $conflict(\psi)(S,k) \leq penalty(\psi)(k)$ by induction, this must hold. □

Our final result follows from Propositions 1 and 2.

Corollary 1 *The conflict function of Definition 7 is a conflict function according to Definition 4.*

Proof. Assume that $conflict(\mathcal{F})(S,k) = 0$. Then $abstractConflict(\mathcal{F})(S,k) = 0$ by Proposition 2 and hence $\forall \ell \in n_S(k) : penalty(\mathcal{F})(k) \leq penalty(\mathcal{F})(\ell)$ by Proposition 1. □

4 Application: The Progressive Party Problem

The progressive party problem [6] is about timetabling a party at a yacht club, where the crews of certain boats (the guest boats) party at other boats (the host boats) over a number of periods. The crew of a guest boat must party at some host boat in each period. The spare capacity of a host boat is never to be exceeded. The crew of a guest boat may visit a particular host boat at most once. The crews of two distinct guest boats may meet at most once.

4.1 A Set-Based Model

Let H and G be the sets of host boats and guest boats, respectively. Let $c(h)$ and $s(g)$ denote the spare capacity of host boat h and the crew size of guest boat g, respectively. Let P be the set of periods. Let $S_{(h,p)}$ be a set variable denoting the set of guest boats whose crews boat h hosts during period p. The following constraints then model the problem [2]:

$$\forall p \in P\ (\mathit{Partition}(\{S_{(h,p)} \mid h \in H\}, G))$$
$$\forall h \in H\ (\forall p \in P\ (\mathit{MaxWeightedSum}(S_{(h,p)}, s, c(h))))$$
$$\forall h \in H\ (\mathit{AllDisjoint}(\{S_{(h,p)} \mid p \in P\}))$$
$$\mathit{MaxIntersect}(\{S_{(h,p)} \mid h \in H \wedge\ p \in P\}, 1)$$

The constraint $\mathit{Partition}(\mathcal{X}, Q)(k)$ holds if and only if the values with respect to k of the set variables in $\mathcal{X}$ partition the set Q and where the value of a set variable in $\mathcal{X}$ may be the empty set. The constraint $\mathit{MaxWeightedSum}(S, w, m)(k)$ holds if and only if $\sum_{u \in k(S)} w(u) \leq m$. The constraint $\mathit{MaxIntersect}(\mathcal{X}, m)(k)$ holds if and only if the cardinality of the intersection with respect to k between any two distinct set variables in $\mathcal{X}$ is at most m. The constraint $\mathit{AllDisjoint}(\mathcal{X})(k)$ holds if and only if the intersection with respect to k between any two distinct set variables in $\mathcal{X}$ is empty.

4.2 Modelling the *AllDisjoint* Constraint

Assume now that the *AllDisjoint* constraint is not available in our local search system. We may then express the constraint in $\exists\mathrm{SOL}^+$ and use that version in the model of the progressive party problem above. Hence, we may experiment with that model without having to come up with and implement a new built-in global constraint. We use the following $\exists\mathrm{SOL}^+$ formula for modelling the $\mathit{AllDisjoint}(\{S_1, \ldots, S_n\})$ constraint:

$$\begin{aligned}\exists S_1 \cdots \exists S_n \forall x((x \notin S_1 \vee (x \notin S_2 \wedge \cdots \wedge x \notin S_n)) \wedge \\ (x \notin S_2 \vee (x \notin S_3 \wedge \cdots \wedge x \notin S_n)) \wedge \cdots \wedge \\ (x \notin S_{n-1} \vee x \notin S_n))\end{aligned}$$

For every value u in the universe we state that: if u is in a set S_i, then u cannot be in any set S_j where $j > i$.

4.3 Performance

We have run the same instances as in [2] on an Intel 2.4 GHz Linux machine with 512 MB memory. Everything in the local search algorithm is the same except for the *AllDisjoint* constraints, which are replaced by suitable instances of the modelled $\exists\mathrm{SOL}^+$ version above. Table 1 shows the results for the modelled and built-in versions of the *AllDisjoint* constraint. The run times for the $\exists\mathrm{SOL}^+$ version are roughly doubled for all the instances, though it must be noted that issues such as designing penalty and conflict functions as well as incremental penalty and conflict maintenance algorithms for the *AllDisjoint* constraint were not necessary. The built-in version may take advantage of global properties of the constraint which, of course, makes it possible to define a faster incremental algorithm.

5 Conclusion

Towards making modelling languages for local search extensible, we have proposed a new way of inferring the conflict of a variable in a constraint from just the syntax of a high-level formulation of that constraint, in a language richer than considered before. We have proven that any inferred variable conflict enjoys the desirable property of being lower-bounded by the actually targeted value, which can usually only be matched by knowing the semantics of the constraint. We have also shown that any inferred variable conflict is upper-bounded by the inferred penalty. In practice, we use a definition of the variable conflicts for n-ary versions of the normally binary connectives $\wedge$ and $\vee$, and we perform an *incremental* maintenance of the variable conflicts, using the ideas of the algorithm for constraint penalties in [2]. We have shown that the search is then indeed directed towards interesting neighbourhoods, as a (presumed) missing built-in constraint can be replaced this way without having to design it from scratch, including incremental algorithms for

Table 1: Run times in seconds for the progressive party problem. Mean run time of successful runs (out of 100) and number of unsuccessful runs (if any) in parentheses.

Results with modelled *AllDisjoint* constraint.

H/periods (fails)	6	7		8	9		10
1-12,16				1.3	3.5		42.0
1-13				16.5	239.3		
1,3-13,19				18.9	273.2	(3)	
3-13,25,26				36.5	405.5	(16)	
1-11,19,21	19.8	186.7					
1-9,16-19	32.2	320.0	(12)				

Results with built-in *AllDisjoint* constraint.

H/periods (fails)	6	7		8	9		10
1-12,16				1.2	2.3		21.0
1-13				7.0	90.5		
1,3-13,19				7.2	128.4	(4)	
3-13,25,26				13.9	170.0	(17)	
1-11,19,21	10.3	83.0	(1)				
1-9,16-19	18.2	160.6	(22)				

maintaining its constraint penalties and variable conflicts, nor incurring too high losses in run-time or quality of the solutions.

The adaptation of the traditional combinators of constraint programming for local search was pioneered for the *Comet* system [8, 7]. The combinators there include logical connectives (such as $\wedge$ and $\vee$), cardinality operators (such as *exactly* and *atmost*), reification, and expressions over variables. We have extended these ideas here to the logical quantifiers, namely $\forall$ and $\exists$. This is not just a matter of simply generalising the arities and the existing definitions [8, 7] of the penalties and conflicts for the $\wedge$ and $\vee$ connectives, respectively, but was made necessary by our handling of set variables over which one would like to iterate, unlike the scalar variables of *Comet*.

Our definition of the conflict of a variable with respect to a disjunctive constraint significantly differs from theirs, as we approximate the *maximum* penalty decrease possible in any of the disjuncts (rules (c) and (e) of Definition 7), whereas they approximate the *minimum* distance for that variable from a value where one of the disjuncts can be satisfied. They thus bias the search towards the disjunct that *seems* easier to satisfy and define variable conflicts *independently* of penalties.

References

[1] M. Ågren, P. Flener, and J. Pearson. Incremental algorithms for local search from existential second-order logic. In P. van Beek, editor, *Proceedings of CP'05*, volume 3709 of *LNCS*, pages 47–61. Springer-Verlag, 2005.

[2] M. Ågren, P. Flener, and J. Pearson. Set variables and local search. In R. Barták and M. Milano, editors, *Proceedings of CP-AI-OR'05*, volume 3524 of *LNCS*, pages 19–33. Springer-Verlag, 2005.

[3] P. Galinier and J.-K. Hao. A general approach for constraint solving by local search. In *Proceedings of CP-AI-OR'00*, 2000.

[4] N. Immerman. *Descriptive Complexity*. Springer-Verlag, 1998.

[5] L. Michel and P. Van Hentenryck. A constraint-based architecture for local search. *ACM SIGPLAN Notices*, 37(11):101–110, 2002. Proceedings of OOPSLA'02.

[6] B. M. Smith, S. C. Brailsford, P. M. Hubbard, and H. P. Williams. The progressive party problem: Integer linear programming and constraint programming compared. *Constraints*, 1:119–138, 1996.

[7] P. Van Hentenryck and L. Michel. *Constraint-Based Local Search*. The MIT Press, 2005.

[8] P. Van Hentenryck, L. Michel, and L. Liu. Constraint-based combinators for local search. In M. Wallace, editor, *Proceedings of CP'04*, volume 3258 of *LNCS*, pages 47–61. Springer-Verlag, 2004.