Inferring Variable Conflicts for Local Search*

Magnus Ågren, Pierre Flener, and Justin Pearson

Department of Information Technology, Uppsala University, Sweden
{agren, pierref, justin}@it.uu.se

Abstract. For efficiency reasons, neighbourhoods in local search are often shrunk by only considering moves modifying variables that actually contribute to the overall penalty. These are known as conflicting variables. We propose a new definition for measuring the conflict of a variable in a model and apply it to the set variables of models expressed in existential second-order logic extended with counting ($\exists$SOL$^+$). Such a variable conflict can be automatically and incrementally evaluated. Furthermore, 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 an $\exists$SOL$^+$ version thereof, while still obtaining competitive results.

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. See [4,6,8], for example.

We address the inference of variable conflicts from a formulation of a constraint. After giving necessary background information in Section 2, we propose in Section 3 a new definition for measuring the conflict of a variable and apply it to the *set* variables of models expressed in existential second-order logic extended with counting ($\exists$SOL$^+$) [5]. Such a variable conflict can be automatically and incrementally evaluated. The calculated value is lower-bounded by an intuitive target value, namely the maximum penalty decrease of the model that may be achieved by only changing the given variable, and upper-bounded by the penalty of the model. We demonstrate the usefulness of the approach in Section 4 by replacing a built-in constraint by an $\exists$SOL$^+$ version, while still obtaining competitive results.

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 being defined on a subset of $\mathcal{X}$ and specifying their valid combinations of values.

* This research was partially funded by EuroControl project C/1.246/HQ/JC/04.

F. Benhamou (Ed.): CP 2006, LNCS 4204, pp. 665–669, 2006.

A variable $S \in \mathcal{X}$ is a *set variable* if its corresponding domain D_S is $2^{\mathcal{U}}$, where $\mathcal{U}$ is a common finite set of values of some type, called the *universe*.

Local search iteratively makes a small change to a current assignment of values to *all* variables (configuration), upon examining the merits of many such changes, until a solution is found or allocated resources have been exhausted. The configurations examined constitute the neighbourhood of the current configuration, crucial guidance being provided by penalties and variable conflicts.

Definition 1. *Let $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ be a CSP. A* configuration *for P (or $\mathcal{X}$) is a total function $k : \mathcal{X} \to \bigcup_{D \in \mathcal{D}} D$. We use $\mathcal{K}$ to denote the* set of all configurations *for a given CSP or set of variables, depending on the context. A* neighbourhood function *for P is a function $n : \mathcal{K} \to 2^{\mathcal{K}}$. The* neighbourhood *of P with respect to (w.r.t.) a configuration $k \in \mathcal{K}$ and n is the set $n(k)$. The* variable neighbourhood *for $x \in \mathcal{X}$ w.r.t. k is the subset of $\mathcal{K}$ reachable from k by changing $k(x)$ only: $n_x(k) = \{\ell \in \mathcal{K} \mid \forall y \in \mathcal{X} : y \neq x \to k(y) = \ell(y)\}$. A* penalty function *of a constraint $c \in \mathcal{C}$ is a function $penalty(c) : \mathcal{K} \to \mathbb{N}$ such that (s.t.) $penalty(c)(k) = 0$ if and only if (iff) c is satisfied w.r.t. k. The* penalty *of c w.r.t. k is $penalty(c)(k)$. A* conflict function *of c is a function $conflict(c) : \mathcal{X} \times \mathcal{K} \to \mathbb{N}$ s.t. if $conflict(c)(x, k) = 0$ then $\forall \ell \in n_x(k) : penalty(c)(k) \leq penalty(c)(\ell)$. The* conflict *of x w.r.t. c and k is $conflict(c)(x, k)$.*

Example 1. Let $P = \langle \{S, T\}, \{D_S, D_T\}, \{S \subset T\} \rangle$ 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 $k = \{S \mapsto \{a, b\}, T \mapsto \emptyset\}$. The neighbourhood of P w.r.t. 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 w.r.t. k is the set $n_S(k) = \{k, k_1 = \{S \mapsto \emptyset, T \mapsto \emptyset\}, k_2 = \{S \mapsto \{a\}, T \mapsto \emptyset\}, k_3 = \{S \mapsto \{b\}, T \mapsto \emptyset\}, k_4 = \{S \mapsto \{c\}, T \mapsto \emptyset\}, k_5 = \{S \mapsto \{a, c\}, T \mapsto \emptyset\}, k_6 = \{S \mapsto \{b, c\}, T \mapsto \emptyset\}, k_7 = \{S \mapsto \{a, b, c\}, T \mapsto \emptyset\}\}$. Let the penalty and conflict functions of $S \subset T$ be defined by:

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

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

We have that $penalty(S \subset T)(k) = 3$. Indeed, we may satisfy P w.r.t. k by, e.g., adding the three values a, b, and c to T. We also have that $conflict(S \subset T)(S, k) = 2$ and $conflict(S \subset T)(T, k) = 3$. Indeed, by removing the values a and b from S, we may decrease the penalty of P by two. Similarly, by adding the values a, b, and c to T, we may decrease the penalty of P by three.

We use existential second-order logic extended with counting ($\exists$SOL$^+$) for modelling set constraints [1]. In the BNF below, the non-terminal symbol $\langle S \rangle$ denotes an identifier for a bound set variable S such that $S \subseteq \mathcal{U}$, 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:

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

As a running example, consider the constraint $S \subset T$ of Ex. 1. This may be expressed in $\exists\mathrm{SOL}^+$ by $\Omega = \exists S \exists T((\forall x(x \notin S \vee x \in T)) \wedge (\exists x(x \in T \wedge x \notin S)))$.

We proposed a penalty function for $\exists\mathrm{SOL}^+$ formulas in [1], which was inspired by [9]. For example, the penalty of a literal $x \in S$ w.r.t. a configuration k is 0 if $k(x) \in k(S)$ and 1, otherwise. The penalty of a conjunction (disjunction) is the sum (minimum) of the penalties of its conjuncts (disjuncts). The penalty of a universal (existential) quantification is the sum (minimum) of the penalties of the quantified formula where the occurrences of the bound variable are replaced by each value in the universe.

Example 2. Recall $k = \{S \mapsto \{a, b\}, T \mapsto \emptyset\}$ of Ex. 1. Then $\mathit{penalty}(\Omega)(k) = 3$.

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

The notion of abstract conflict measures the maximum possible penalty decrease obtainable by only changing the value of the given variable. It is uniquely determined by the chosen penalty function:

Definition 2. *Let $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ be a CSP and let $c \in \mathcal{C}$. The* abstract conflict function *of c w.r.t. $\mathit{penalty}(c)$ is the function $\mathit{abstractConflict}(c) : \mathcal{X} \times \mathcal{K} \to \mathbb{N}$ s.t. $\mathit{abstractConflict}(c)(x, k) = \max\{\mathit{penalty}(c)(k) - \mathit{penalty}(c)(\ell) \mid \ell \in n_x(k)\}$. The* abstract conflict *of $x \in \mathcal{X}$ w.r.t. c and $k \in \mathcal{K}$ is $\mathit{abstractConflict}(c)(x, k)$.*

Example 3. The function $\mathit{conflict}(S \subset T)$ of Ex. 1 gives abstract conflicts.

Similarly to our penalty function in [1], it is important to stress that the calculation of the variable conflict defined next is automatable and feasible incrementally [3], as it is based only on the syntax of the formula and the semantics of the quantifiers, connectives, and relational operators of $\exists\mathrm{SOL}^+$, but not on the intended semantics of the formula.

Definition 3. *Let $\mathcal{F} \in \exists\mathrm{SOL}^+$, let $S \in \mathit{vars}(\mathcal{F})$, and let k be a configuration for $\mathit{vars}(\mathcal{F})$. The* conflict *of S w.r.t. $\mathcal{F}$ and k is defined by:*

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

We only show cases for subformulas of the form $|S| \Diamond c$ *and* $x \triangle S$ *where* $\Diamond \in \{\leq\}$ *and* $\triangle \in \{\in\}$*. The other cases are defined similarly.*

Example 4. Recall once again $k = \{S \mapsto \{a, b\}, T \mapsto \emptyset\}$ of Ex. 1. According to Def. 3, we have that $\mathit{conflict}(\Omega)(S, k) = 2$ and $\mathit{conflict}(\Omega)(T, k) = 3$, i.e., the same values as obtained by the handcrafted $\mathit{conflict}(S \subset T)$ of Ex. 1.

The novelty of Def. 3 compared to the one in [8] lies in rules (c) and (e) for disjunctive formulas, due to the different abstract conflict that we target (see [3] for more details). The following example clarifies these rules in terms of (e).

Example 5. Consider $\mathcal{F} = (|S| = 5 \vee (|T| = 3 \wedge |S| = 6))$ and let k_1 be a configuration s.t. $|k_1(S)| = 6$ and $|k_1(T)| = 4$. Then $\mathit{penalty}(\mathcal{F})(k_1) = 1$ and we have $\mathit{conflict}(|S| = 5)(S, k_1) = 1$ and $\mathit{conflict}(|T| = 3 \ \wedge \ |S| = 6)(S, k_1) = 0$. Rule (e) applies for calculating $\mathit{conflict}(\mathcal{F})(S, k_1)$, which, for each disjunct, gives the *maximum possible penalty decrease* one may obtain by changing $k_1(S)$. This is 1 for the first disjunct since we may decrease $\mathit{penalty}(\mathcal{F})(k_1)$ by 1 by changing $k_1(S)$ as witnessed by $\mathit{penalty}(\mathcal{F})(k_1) - (\mathit{penalty}(|S| = 5)(k_1) - \mathit{conflict}(|S| = 5)(S, k_1)) = 1 - (1 - 1) = 1$. It is 0 for the second disjunct since we cannot decrease $\mathit{penalty}(\mathcal{F})(k_1)$ by changing $k_1(S)$ as witnessed by $\mathit{penalty}(\mathcal{F})(k_1) - (\mathit{penalty}(|T| = 3 \wedge |S| = 6)(k_1) - \mathit{conflict}(|T| = 3 \wedge |S| = 6)(S, k_1) = 1 - (1 - 0) = 0$. The maximum value of these is 1 and hence $\mathit{conflict}(\mathcal{F})(S, k_1) = 1$.

Consider now k_2 s.t. $|k_2(S)| = 4$ and $|k_2(T)| = 4$. Then $\mathit{penalty}(\mathcal{F})(k_2) = 1$ and $\mathit{conflict}(|T| = 3 \ \wedge \ |S| = 6)(T, k_2) = 1$. The maximum possible penalty decrease one may obtain by changing $k_2(T)$ in the only disjunct for T is -1 as witnessed by $\mathit{penalty}(\mathcal{F})(k_2) - (\mathit{penalty}(|T| = 3 \wedge |S| = 6)(k_2) - \mathit{conflict}(|T| = 3 \wedge |S| = 6)(T, k_2) = 1 - (3 - 1) = -1$. But we may not have a negative conflict, hence the union with $\{0\}$ in (e). Indeed, we cannot decrease $\mathit{penalty}(\mathcal{F})(k)$ by changing $k_2(T)$ since even if we satisfy $|k_2(T) = 3|$, the conjunct $|k_2(S) = 6|$ implies a penalty larger than 1 which is the minimum penalty of the two disjuncts.

We now state some properties of variable conflicts compared to the abstract conflict of Def. 2 and the formula penalty [1]. The proofs can be found in [3].

Proposition 1. *Let* $\mathcal{F} \in \exists\mathrm{SOL}^+$*, let* k *be a configuration for* $\mathit{vars}(\mathcal{F})$*, and let* $S \in \mathit{vars}(\mathcal{F})$*. Then* $\mathit{abstractConflict}(\mathcal{F})(S, k) \leq \mathit{conflict}(\mathcal{F})(S, k) \leq \mathit{penalty}(\mathcal{F})(k)$*.*

Corollary 1. *The function induced by Def. 3 is a conflict function w.r.t. Def. 1.*

4 Practical Results and Conclusion

The *progressive party problem* [7] 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.

We use the same set-based model and local search algorithm as we did in [2]. The model includes $AllDisjoint(\mathcal{X})(k)$ constraints that hold iff no two distinct set variables in $\mathcal{X} = \{S_1, \ldots, S_n\}$ overlap. Assuming that this global constraint is not built-in, we may use the following $\exists$SOL$^+$ version instead:

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

We have run the same classical instances as we did in [2], on a 2.4GHz/512MB Linux machine. The following table shows the results for the $\exists$SOL$^+$ and built-in versions of the *AllDisjoint* constraint (mean run time in seconds of successful runs out of 100 and the number of unsuccessful runs, if any, in parentheses).

	$\exists$SOL$^+$ *AllDisjoint*					Built-in *AllDisjoint*				
H/periods (fails)	6	7	8	9	10	6	7	8	9	10
1-12,16			1.3	3.5	42.0			1.2	2.3	21.0
1-13			16.5	239.3				7.0	90.5	
1,3-13,19			18.9	273.2 (3)				7.2	128.4 (4)	
3-13,25,26			36.5	405.5 (16)				13.9	170.0 (17)	
1-11,19,21	19.8	186.7				10.3	83.0 (1)			
1-9,16-19	32.2	320.0 (12)				18.2	160.6 (22)			

The run times for the $\exists$SOL$^+$ version are only 2 to 3 times higher, though it must be noted that *efforts such as designing penalty and conflict functions as well as incremental maintenance algorithms for AllDisjoint were not necessary.* Note also that the robustness of the local search algorithm does not degrade for the $\exists$SOL$^+$ version, as witnessed by the number of solved instances.

To conclude, we proposed a new definition for inferring the conflict of a variable in a model and proved that any inferred variable conflict is lower-bounded by the targeted value, and upper-bounded by the inferred penalty. *The search is indeed directed towards interesting neighbourhoods, as a built-in constraint can be replaced without too high losses in run-time, nor any losses in robustness.*

References

1. M. Ågren, P. Flener, and J. Pearson. Incremental algorithms for local search from existential second-order logic. *Proceedings of CP'05*. Springer-Verlag, 2005.
2. M. Ågren, P. Flener, and J. Pearson. Set variables and local search. *Proceedings of CP-AI-OR'05*. Springer-Verlag, 2005.
3. M. Ågren, P. Flener, and J. Pearson. Inferring variable conflicts for local search. Tech. Rep. 2006-005, Dept. of Information Technology, Uppsala University, 2006.
4. P. Galinier and J.-K. Hao. A general approach for constraint solving by local search. *Proceedings of CP-AI-OR'00*, 2000.
5. N. Immerman. *Descriptive Complexity*. Springer-Verlag, 1998.
6. L. Michel and P. Van Hentenryck. A constraint-based architecture for local search. *Proceedings of OOPSLA'02*, 2002.
7. B. M. Smith *et al.* The progressive party problem: Integer linear programming and constraint programming compared. *Constraints*, 1:119–138, 1996.
8. P. Van Hentenryck and L. Michel. *Constraint-Based Local Search*. MIT Press, 2005.
9. P. Van Hentenryck, L. Michel, and L. Liu. Constraint-based combinators for local search. *Proceedings of CP'04*. Springer-Verlag, 2004.