Consistency Techniques for Finding an Optimal Relaxation of a Feature Subscription

David Lesaint
BT plc, UK
david.lesaint@bt.com

Deepak Mehta
4C, UCC, Ireland
d.mehta@4c.ucc.ie

Barry O'Sullivan
4C, UCC, Ireland
b.osullivan@4c.ucc.ie

Luis Quesada
4C, UCC, Ireland
l.quesada@4c.ucc.ie

Nic Wilson
4C, UCC, Ireland
n.wilson@4c.ucc.ie

Abstract

Telecommunication services are playing an increasing and potentially disruptive role in our lives. As a result, service providers seek to develop personalisation solutions that put customers in charge of controlling and enriching their services. In this context, the personalisation approach consists of exposing a catalogue of call control features (e.g., call-divert, voice-mail) to end-users and letting them subscribe to a subset of features subject to a set of precedence and exclusion constraints. When a subscription is inconsistent, the problem is to find an optimal relaxation. We present a constraint programming formulation to find an optimal reconfiguration of features. We investigate the performance of maintaining arc consistency within branch and bound search. We also study the impact of maintaining mixed consistency, that is maintaining different levels of consistency on different sets of variables. We further present a global constraint and a set of filtering rules that exploit the structure of our problem. We theoretically and experimentally compare all approaches. Our results demonstrate that the filtering rules of the global constraint outperform all other approaches when a catalogue is dense, and mixed consistency pays off when a catalogue is sparse.

1 Introduction

Information and communication services, from news feeds to internet telephony, are playing an increasing, and potentially disruptive, role in our lives. As a result, service providers seek to develop *personalisation solutions* that put customers in charge of controlling and enriching the behaviour of their telecommunication services. An outcome of this work is the emergence of *features* as fundamental primitives for personalisation. A feature is an increment of functionality which, if activated, modifies the basic service behaviour, e.g., do-not-disturb, multimedia ring-back tones, call-divert-on-busy, credit-card-calling, find-me, etc. In this context, a personalisation approach consists of exposing a *catalogue of features* to end-users and letting them *subscribe* to a subset of features and *sequence* them in the way they prefer. However, not all subscriptions and sequences are acceptable due to the possible occurrence of *feature interactions* [4]. A feature interaction is "some way in which a feature modifies or influences the behavior of another feature in generating the system's overall behavior" [3].

Distributed feature composition provides a comprehensive methodology underpinned by a formal architecture model to address feature interaction [6]. The method consists of constraining the selection and sequencing of features by prescribing constraints that prevent undesirable interactions. These feature interaction resolution constraints are represented in a feature catalogue as precedence or exclusion constraints. A precedence constraint, $f_i \prec f_j$, means that if the features f_i and f_j are part of the same sequence then f_i must precede f_j in the sequence. An exclusion constraint between f_i and f_j means that they cannot occur together in any sequence. Undesirable interactions are then avoided by rejecting any sequence that does not satisfy the catalogue constraints. In addition, a user can also specify precedence constraints between features.

Informally, a *feature subscription* is defined by a set of features, a set of user specified precedence constraints between pairs of features, a set of weights associated with the features and the user specified precedence constraints, and a set of precedence and exclusion constraints from the catalogue. The task is to find a sequence of features that is consistent with the constraints in the catalogue. It may not always be possible to construct a sequence of features that consists of all the user selected features and respect all user specified precedence constraints. In such cases, *the task is to find an optimal relaxation of the feature subscription that is closest to the initial requirements of the user*. This problem is NP-hard [10].

We present a *constraint programming* formulation for finding an optimal reconfiguration of a feature subscription. In particular, we formulate our problem as a Constraint Optimisation Problem (COP). In general, a branch and bound search algorithm that maintains some level of

local consistency is usually used for finding an optimal solution. A preliminary comparison of maintaining arc consistency and two mixed consistencies [8] has been presented in [9]. Note that here mixed consistency means maintaining different levels of consistency on different sets of variables of a given problem.

In this paper, we present a global constraint that takes into account the structure of our problem. Since achieving arc consistency on the proposed global constraint is NP-complete, we present a set of filtering rules that approximate this level of consistency in polynomial time. These rules extend the ones proposed in [1] in three ways: (a) they handle user defined (soft) precedences, (b) they handle non-unary positive weights, and (c) they compute tighter bounds. We present the pruning rules of the global constraint declaratively, which gives clear insights regarding the semantics of the constraint. We also compare all the before mentioned filtering techniques theoretically and experimentally. The experiments are performed on a variety of random catalogues and feature subscriptions. Our results demonstrate that the filtering rules of the global constraint outperform all the other approaches when a catalogue is dense and mixed consistency pays off when a catalogue is sparse.

2 Feature Subscription

In this section we present some notation and formal definitions that are relevant for the understanding of the telecommunication problem considered in this paper.

Let f_i and f_j be features, we write a precedence constraint of f_i before f_j as $\langle f_i, f_j \rangle$, or as $f_i \prec f_j$. An exclusion constraint between f_i and f_j expresses that these features cannot appear together in a sequence of features. We encode this as the pair of precedence constraints $\langle f_i, f_j \rangle$ and $\langle f_j, f_i \rangle$. A **catalogue** is a tuple $\langle \mathcal{F}, \mathcal{H} \rangle$, where $\mathcal{F}$ is a set of features that are available to users and $\mathcal{H}$ is a set of precedence constraints on $\mathcal{F}$.

A **feature subscription** $\mathcal{S}$ of a catalogue $\langle \mathcal{F}, \mathcal{H} \rangle$ is a tuple $\langle F, H, P, w_f, w_p \rangle$, where $F \subseteq \mathcal{F}$, H is the projection of $\mathcal{H}$ on F, i.e., $\mathcal{H}{\downarrow_F} = \{f_i \prec f_j \in \mathcal{H} : \{f_i, f_j\} \subseteq F\}$, P is a set of (user defined) precedence constraints on F, $w_f : F \rightarrow \mathbb{N}$ is a function that assigns weights to features and $w_p : P \rightarrow \mathbb{N}$ is a function that assigns weights to user precedence constraints. The value of $\mathcal{S}$ is defined by $\mathrm{Value}(\mathcal{S}) = \sum_{f \in F} w_f(f) + \sum_{p \in P} w_p(p)$. Note that a weight associated with a feature signifies its importance for the user. These weights could be elicited directly, or using data mining or analysis of user interactions.

A feature subscription $\mathcal{S} = \langle F, H, P, w_f, w_p \rangle$ is defined to be **consistent** if and only if the directed graph $\langle F, H \cup P \rangle$ is acyclic. Determining whether a feature subscription $\mathcal{S}$ is consistent or not can be checked in $\mathcal{O}(|F|+|H \cup P|)$ time by using a topological sort [9]. Note that if a feature subscription is consistent then it implies that there exists at least one sequence of features that does not have undesirable feature interactions. If an input feature subscription is not consistent then the task is to relax the input feature subscription by dropping one or more features or user precedence constraints to generate a consistent feature subscription with maximum value.

A **relaxation** of a feature subscription $\langle F, H, P, w_f, w_p \rangle$ is a subscription $\langle F', H', P', {w_f}', {w_p}' \rangle$ such that $F' \subseteq F$, $H' = \mathcal{H}{\downarrow_{F'}}$, $P' \subseteq P{\downarrow_{F'}}$, ${w_f}'$ is w_f restricted to F', and ${w_p}'$ is w_p restricted to P'. Let $R_{\mathcal{S}}$ be the set of all consistent relaxations of a feature subscription $\mathcal{S}$. We say that $\mathcal{S}_i \in R_{\mathcal{S}}$ is an **optimal relaxation** of $\mathcal{S}$ if it has maximum value among all relaxations, i.e., if and only if there does not exist $\mathcal{S}_j \in R_{\mathcal{S}}$ such that $\mathrm{Value}(\mathcal{S}_j) > \mathrm{Value}(\mathcal{S}_i)$. Finding an optimal relaxation of a subscription is NP-hard [10].

3 Constraint Programming

Constraint Programming has been successfully used in many applications such as planning, scheduling, resource allocation, routing, and bio-informatics [12]. Here problems are primarily stated as a Constraint Satisfaction Problems (CSP), that is a finite set of variables, together with a finite set of constraints. A solution to a CSP is an assignment of a value to each variable such that all constraints are satisfied simultaneously. The basic approach to solving a CSP instance is to use a backtracking search algorithm that interleaves two processes: *constraint propagation* and *labeling*. Constraint propagation helps in pruning values that do not lead to a solution of the problem. Labeling involves assigning values to variables that may lead to a solution.

A CSP is said to be *Arc Consistent* (AC) if it has non-empty domains and for any assignment of a variable each constraint in which that variable is involved can be satisfied. A CSP is said to be *Singleton Arc Consistent* [2] if it has non-empty domains and for any assignment of a variable, the resulting subproblem can be made AC. Mixed consistency means maintaining different levels of consistency on different variables of a problem. In [8] it has been shown that maintaining SAC on some variables and AC on the remaining variables can reduce the solution time.

Various generalisations of the CSP have been developed to find a solution that is optimal with respect to certain criteria such as costs, preferences or priorities. One of the most significant is the Constraint Optimisation Problem (COP). Here the goal to find an optimal solution that maximises (minimises) an objective function. The simplest COP formulation retains the CSP limitation of allowing only hard Boolean-valued constraints but adds an objective function over the variables.

A depth-first branch and bound algorithm (BB) is generally used to find a solution of a COP having an optimal

value. In the case of maximisation, BB keeps the current optimal value of the solution while traversing the search tree. This value is a lower bound (v^-) on the objective function. At each node of the search tree BB computes an overestimation of the global value. This value is an upper bound (v^+) on the best solution that can be found as long as the current search path is maintained. If $v^- \geq v^+$, then a solution of a greater value than the current optimal value cannot be found below the current node, so the current branch is pruned and the algorithm backtracks.

4 A COP Formulation for Finding an Optimal Relaxation of a Feature Subscription

In this section we model the problem of finding an optimal relaxation of a feature subscription $\langle F, H, P, w_f, w_p \rangle$ as a COP.

Variables and Domains. We associate each feature $f_i \in F$ with two variables: a *Boolean variable* bf_i and an *integer variable* pf_i. A Boolean variable bf_i is instantiated to 1 or 0 depending on whether f_i is included in the computed subscription or not, respectively. The domain of each integer variable pf_i is $\{1, \ldots, |F|\}$. Assuming that the computed subscription is consistent, an integer variable pf_i corresponds to the position of the feature f_i in a sequence. We associate each user precedence constraint $p_{ij} \equiv (f_i \prec f_j) \in P$ with a *Boolean variable* bp_{ij}. A Boolean variable bp_{ij} is instantiated to 1 or 0 depending on whether p_{ij} is respected in the computed subscription or not, respectively. We associate a variable v with the objective function, the initial lower domain of which is 0 and the initial upper bound is the sum of the weights of all the features and user precedences.

Constraints. A catalogue precedence constraint $p_{ij} \in H$ that feature f_i should be before feature f_j can be expressed as $bf_i \wedge bf_j \Rightarrow (pf_i < pf_j)$. Note that the constraint is trivially satisfied if either of the selection variables bf_i and bf_j is instantiated to 0.

A user precedence constraint $p_{ij} \in P$ that f_i should be placed before f_j in their subscription can be expressed as $bp_{ij} \Rightarrow (bf_i \wedge bf_j \wedge (pf_i < pf_j))$. Note that if a user precedence constraint holds then the features f_i and f_j are included in the subscription and also the feature f_i is placed before f_j, that is, the selection variables bf_i and bf_j are instantiated to 1 and $pf_i < pf_j$ is true.

Objective Function. The objective is to maximise the value of v, where

$$v = \sum_{f_i \in F} bf_i \times w_f(f_i) + \sum_{p_{ij} \in P} bp_{ij} \times w_p(p_{ij}).$$

We have investigated the impact of maintaining three different levels of consistency within branch and bound search. The first is arc consistency and the rest are mixed consistencies. In the following sections, we shall describe these consistency techniques and present their worst-case time complexities when enforced on any instance of feature subscription, if formulated as described before.

4.1 Arc Consistency

The worst-case time complexity of enforcing AC is $\mathcal{O}(e\,d+n^2)$, where e is the total number of user precedences and catalogue precedences, n is the number of Boolean variables associated with the features and the user precedences, and d is the number of features. Note that the worst-case time complexity of making all precedence constraints arc consistent is $\mathcal{O}(e\,d)$, since there are e precedence constraints, each of them can be made arc consistent at most d times, and the complexity of making each of them arc-consistent is constant. In addition, arc consistency is also enforced on the constraint $v^- < \sum_{f_i \in F} bf_i \times w_f(f_i) + \sum_{p_{ij} \in P} bp_{ij} \times w_p(p_{ij})$, the complexity of which is linear with respect to the number of Boolean variables. Whenever a Boolean variable is instantiated the constraint is revised and since there are at most $2n$ variable-value pairs, it can be made arc-consistent at most $2n$ times. Therefore, the worst case time complexity of enforcing arc consistency on this constraint is $\mathcal{O}(n^2)$. Also notice that enforcing arc consistency after initialising all the Boolean variables guarantees that the problem is globally consistent. Therefore, our decision variables are only the Boolean variables.

4.2 Singleton Arc Consistency

Maintaining a higher level of consistency can be expensive in terms of time. However, if more values can be removed from the domains of the variables, the search effort can be reduced and this may save time. We shall investigate the effect of maintaining Singleton Arc Consistency (SAC) on the Boolean variables and AC on the remaining variables and denote it by SAC$_b$. We have used the SAC-1 [5] algorithm for enforcing SAC on the Boolean variables. Enforcing SAC on the Boolean variables in a SAC-1 manner works by traversing a list of of $2n$ variable-value pairs. For each instantiation of a Boolean variable x to each value $0/1$, if there is a domain wipeout while enforcing AC then the value is removed from the corresponding domain and AC is enforced. Each time a value is removed, the list is traversed again. Since there are $2n$ variable-value pairs, the number of calls to the underlying arc consistency algorithm is at most $4n^2$. Thus the worst-case time complexity of SAC$_b$ is $\mathcal{O}(n^2\,(e\,d + n^2))$.

4.3 Restricted Singleton Arc Consistency

The main problem with SAC-1 is that deleting a single value triggers the loop again. The Restricted SAC (RSAC) avoids this by considering each variable-value pair only once [11]. We also investigate the effect of maintaining Restricted SAC (RSAC) on the Boolean variables and AC on the remaining variables, and denote it by RSAC_b. The worst-case time complexity of RSAC_b is $\mathcal{O}(n\,(e\,d + n^2))$.

5 The SoftPrec Global Constraint

Another way to prune more values is to use a global constraint that takes into account the structure of our problem. For instance, if a user has selected features f_1, f_2, f_3 and f_4 and if these features are constrained by the catalogue precedences $f_1 \prec f_2$, $f_2 \prec f_1$, $f_3 \prec f_4$ and $f_4 \prec f_3$, and if three features are required to be included then we would like to fail without doing any search. This is possible by inferring cycles from the precedence constraints and using them to prune the bounds of the objective function. We propose a global constraint that ensures that the selected features and user precedences are consistent with the catalogue precedences and subsequently prunes the bounds of the objective function based on the cycles inferred between pairs of features that are not yet selected.

Let $\langle F, H, P, w_f, w_p\rangle$ be a feature subscription. Let *bf* be a vector of Boolean variables associated with features F. We say that f_i is included if *bf*$(i) = 1$, and f_i is excluded if *bf*$(i) = 0$. We abuse the notation by using *bf*(i) to mean *bf*$(i) = 1$ and $\neg$*bf*(i) to mean *bf*$(i) = 0$. A similar convention is adopted for the other Boolean variables. Let *bp* be a matrix of Boolean variables associated with pairs of features. Here *bp*(i, j) means that f_i is included, f_j is included, and f_i precedes f_j. *bp* is intended to represent a partial ordering between the chosen features F' which is compatible with the catalogue constraints restricted to F'. Note that the definition of *bp* is different than the one used in Section 4. We say that *bp* is consistent with *bf* if *bp*$(i, j) \Rightarrow$ *bf*$(i) \wedge$ *bf*(j). We say that *bp* represents a *strict partial order* on *bf*, if for all the included features i, j and k, (i) *bp*$(i, j) \wedge$ *bp*$(j, k) \Rightarrow$ *bp*(i, k) (transitive) and (ii) *bp*$(i, j) \Rightarrow \neg$*bp*(j, i) (asymmetric).

To aid propagation, we also introduce another matrix *prec* of Boolean variables associated with pairs of features. Here *prec*(i, j) means, roughly speaking, that if f_i and f_j are included then f_i must precede f_j. We say that *prec* extends *bp* if for all i, j, *bp*$(i, j) \Rightarrow$ *prec*(i, j). We say that *prec* extends the catalogue precedence constraints H if for all i and j, $(f_i \prec f_j) \in H \Rightarrow$ *prec*(i, j). We say that *prec* is *bf*-transitive if for all i, j and k, *bf*$(j) \wedge$ *prec*$(i, j) \wedge$ *prec*$(j, k) \Rightarrow$ *prec*(i, k).

We say that *prec*, *bf* and *bp* are compatible with each other if for all i and j, *prec*$(i, j) \wedge$ *bf*$(i) \wedge$ *bf*$(j) \Leftrightarrow$ *bp*(i, j). Notice that this compatibility implies that *bp* is consistent with *bf*, *prec* extends *bp*, and *prec* is asymmetric on the included features.

Definition 1 (Semantics of *SoftPrec*) *Given that prec, bf and bp are vectors of Boolean variables, v is an integer variable, and $S = \langle F, H, P, w_f, w_p\rangle$ is a feature subscription, the global constraint SoftPrec(bf, bp, prec, S, v) holds if and only if*

(i) bp represents a strict partial order on bf;

(ii) prec extends the catalogue H;

(iii) prec is bf-transitive;

(iv) prec, bf and bp are compatible with each other;

(v) $v = \sum_{i \in F}$ *bf*$(i) \times w_f(i) + \sum_{\langle i,j\rangle \in P}$ *bp*$(i, j) \times w_p(i, j)$.

As it is possible to express the feedback vertex problem in terms of *SoftPrec*, achieving generalised arc consistency is NP-complete.

5.1 Pruning Rules

In this section we present a set of pruning rules that follows from the semantics of *SoftPrec*. Each pruning rule has the following form: $\dfrac{\textit{Precondition}}{\textit{Postcondition}}$.

Here *Precondition* defines what should be true for the rule to be triggered and *Postcondition* defines what should be true after executing the pruning rule. The pruning rules presented here extends the ones presented in [1]. by considering user precedence constraints, non-unary costs for both features and precedences and computing tighter bounds. We have divided the pruning rules into two categories: those that maintain transitive and asymmetric properties of the precedence relation, and those that prune the values based on the computed bounds. Note that consistency with the catalogue can be ensured by pruning the domain of *prec* based on the following implication: $(f_i \prec f_j) \in H \Rightarrow$ *prec*(i, j).

5.1.1 Pruning Rules for Maintaining Transitive and Asymmetric Properties

SoftPrec infers incompatibilities between the features by maintaining the transitive closure of the precedence relation. As features are optional, transitivity can only be enforced when the connecting features are included:

$$\frac{\textit{bf}(j) \wedge \textit{prec}(i, j) \wedge \textit{prec}(j, k)}{\textit{prec}(i, k)} \qquad (1)$$

$bp(i,j)$ is satisfied if and only if the corresponding features are included and $prec(i,j)$ holds:

$$\frac{bp(i,j)}{bf(i) \wedge bf(j) \wedge prec(i,j)} \quad \frac{bf(i) \wedge bf(j) \wedge prec(i,j)}{bp(i,j)} \quad (2)$$

$$\frac{(\neg bf(i) \vee \neg bf(j) \vee \neg prec(i,j))}{\neg bp(i,j)} \quad (3)$$

As *bp* is asymmetric and *prec* is equivalent to *bp* on the included features, *prec* is asymmetric on the included features, i.e., $(prec(i,j) \wedge prec(j,i)) \Rightarrow \neg(bf(i) \wedge bf(j))$:

$$\frac{bf(i) \wedge bf(j) \wedge prec(i,j)}{\neg prec(j,i)} \quad (4)$$

$$\frac{bf(i) \wedge prec(i,j) \wedge prec(j,i)}{\neg bf(j)} \quad (5)$$

$$\frac{prec(j,i)}{\neg bp(i,j)} \quad (6)$$

Notice that Rule 1 is a direct consequence of (iii) of the definition of *SoftPrec*, Rules 2 and 3 immediately follow from (iv), and Rules 4, 5 and 6 follow from (i) and (iv).

5.1.2 Pruning Rules for Updating Bounds

Let F^+ and F^- be the sets of included features and excluded features respectively. Let P^+ and P^- be the sets of included user precedences and excluded user precedences, respectively. Let $mv = \sum_i w_f(i) + \sum_{\langle j,k \rangle} w_p(j,k)$ be the maximum value of the subscription, when all the features and the user precedences are included. Let $cv = \sum_{i \in F^+} w_f(i) + \sum_{\langle j,k \rangle \in P^+} w_p(j,k)$ be the current value of the subscription. Let $cc = \sum_{i \in F^-} w_f(i) + \sum_{\langle j,k \rangle \in P^-} w_p(j,k)$ be the current cost of the subscription.

We say that two features i and j are mutually exclusive when they precede each other, i.e., $prec(i,j) \wedge prec(j,i)$. Let ex be the set of mutually exclusive pairs of features. Let $fex_f(i)$ be the set of undecided[1] features j such that $\{i,j\} \in ex$. Let $pex_f(i)$ be the set of undecided user precedences $\langle j,k \rangle$ such that either $j \in fex_f(i)$ or $k \in fex_f(i)$. Note that $fex_f(i)$ and $pex_f(i)$ are the sets of those features and user precedences, respectively, whose exclusion is an immediate consequence of the inclusion of feature i. Let $c_f^+(i) = \sum_{j \in fex_f(i)} w_f(j) + \sum_{\rho \in pex_f(i)} w_p(\rho)$ be the cost of including feature i in the subscription. Let $c_f^-(i)$ be the cost of excluding feature i from the subscription, which is the sum of the weight of the feature and the weights of the undecided user precedences that involve feature i.

Let $lb_f^+(i)$ and $lb_f^-(i)$ be the lower bounds of v, when f_i is included and excluded, respectively. Let $ub_f^+(i)$ and $ub_f^-(i)$ be the upper bounds, when f_i is included and excluded, respectively. These bounds are computed as follows:

$$\begin{aligned} lb_f^+(i) &= cv + w_f(i) \\ lb_f^-(i) &= cv \\ ub_f^+(i) &= mv - (cc + c_f^+(i)) \\ ub_f^-(i) &= mv - (cc + c_f^-(i)). \end{aligned}$$

If the bounds obtained when excluding/including a feature is not within the bounds of v then the feature must be included/excluded, respectively. The following are the pruning rules defined in terms of these bounds.

$$\frac{(lb_f^-(i) > v^+) \vee (ub_f^-(i) < v^-)}{bf(i)} \quad (7)$$

$$\frac{(lb_f^+(i) > v^+) \vee (ub_f^+(i) < v^-)}{\neg bf(i)} \quad (8)$$

The upper (lower) bound of the value of the subscription cannot be greater (less) than the maximum (minimum) of the upper (lower) bound of the value resulting from the inclusion and exclusion of a feature:

$$\frac{v^+ > \max(ub_f^+(i), ub_f^-(i))}{v^+ = \max(ub_f^+(i), ub_f^-(i))} \quad (9)$$

$$\frac{v^- < \min(lb_f^+(i), lb_f^-(i))}{v^- = \min(lb_f^+(i), lb_f^-(i))} \quad (10)$$

Let $iw_f(i)$ be the increment in the current value obtained by including feature i, i.e., $iw_f(i) = w_f(i)$ if the inclusion of the feature is undecided, otherwise $iw_f(i) = 0$. Let $fex_p(\langle i,j \rangle)$ be the set of undecided features k such that either $\{i,k\} \in ex$ or $\{j,k\} \in ex$. Let $pex_p(\rho)$ be the set of undecided user precedences $\langle k,l \rangle$ such that either $k \in fex_p(\rho)$ or $l \in fex_p(\rho)$. In other words, $fex_p(\rho)$ $pex_p(\rho)$ are the sets of features and user precedences that are excluded immediately as a consequence of the inclusion of the user precedence ρ. Let $c_p^+(\rho) = \sum_{i \in fex_p(\rho)} w_f(i) + \sum_{\rho' \in pex_p(\rho)} w_p(\rho')$ be the cost of including precedence ρ

Let $lb_p^+(\rho)$ and $lb_p^-(\rho)$ be the lower bounds and let $ub_p^+(\rho)$ and $ub_p^-(\rho)$ be the upper bounds when a user precedence ρ is included and excluded, respectively. These bounds are computed as follows:

$$\begin{aligned} lb_p^+(\langle i,j \rangle) &= cv + w_p(\langle i,j \rangle) + iw_f(i) + iw_f(j) \\ lb_p^-(\rho) &= cv \\ ub_p^+(\rho) &= mv - (cc + c_p^+(\rho)) \\ ub_p^-(\rho) &= mv - (cc + w_p(\rho)). \end{aligned}$$

The pruning rules associated with these bounds are obtained by replacing $lb_f^+(a)$, $lb_f^-(a)$, $ub_f^+(a)$, $ub_f^-(a)$ and $bf(a)$ with $lb_p^+(\rho)$, $lb_p^-(\rho)$, $ub_p^+(\rho)$, $ub_p^-(\rho)$ and $bp(\rho)$ in Equations 7–10, respectively.

[1] Features and user precedences associated with undetermined Boolean variables are said to be undecided.

5.2 Time Complexity

In this section we describe the complexities of the pruning rules. Let α be the number of features, let β be the number of user precedences. It is recalled that n is the sum of the number of features and user precedences, i.e., $n = \alpha + \beta$.

If Rule 1 is activated due to $bf(j)$, then at most α^2 precedences can be included. As there are α features in total, the complexity is $\mathcal{O}(\alpha^3)$. If the rule is activated due to $prec(i, j)$ or $prec(j, k)$ we may infer $prec(i, k)$ for at most $\mathcal{O}(\alpha)$ precedences. As there are α^2 possible precedences, the complexity is $\mathcal{O}(\alpha^3)$. Thus, the overall complexity of this pruning rule is $\mathcal{O}(\alpha^3)$. The rules in Equation 2 can be activated at most α^2 times, and the complexity of the work done at each activation is constant. Therefore, the overall complexity of these pruning rules is $\mathcal{O}(\alpha^2)$. The same is true for Rules 3, 4, 5 and 6.

For all the rules described in Section 5.1.2 the complexity is determined by the number of times the cost of the Boolean variables bf and bp are computed times the complexity of computing the cost. The complexity of computing the cost of including/excluding a feature/precedence is linear with respect to n. For each Boolean variable the cost can be computed at most $\mathcal{O}(n)$ times and since there are n Boolean variables the overall complexity is $\mathcal{O}(n^3)$.

5.3 Tighter Upper Bounds

In Section 5.1.2 the upper bound of including/excluding a feature i is deduced based on the current cost of the partially built subscription and the cost of including/excluding f_i. Remember that the cost of including f_i is computed based on the sets of undecided features j and undecided user precedences involving those features j such that $\{f_i, f_j\} \in ex$. Remember also that the cost of excluding f_i is computed based on the set of the undecided user precedences involving f_i and the feature i itself. In other words we can say that when computing the cost of including or excluding f_i, a set of mutually exclusive pairs of features $\{f_j, f_k\} \in ex$ is used such that f_j and f_k are undecided and the exclusion of any of them is the immediate consequence of including or excluding f_i. Note that if f_i and f_j are undecided and if $\{f_i, f_j\} \in ex$ then the inclusion of f_i results in the exclusion of f_j. In this section we show that a tighter upper bound of including/excluding f_i can be deduced by computing an additional cost from those pairs $\{f_j, f_k\} \in ex$ such that both f_j and f_k remain undecided even after including/excluding f_i. We shall call it a *forward cost* (*fc*). The following steps are required to compute it:

- Construct a set G of all f_i such that $\{f_i, f_j\} \in ex$ and both f_i and f_j are undecided.
- Partition G into sets $C_1, \ldots, C_k$ such that the features involved in any two intersecting pairs of features in ex are in the same set, e.g., if f_i, f_j, and f_k are in G, $\{f_i, f_j\} \in ex$ and $\{f_j, f_k\} \in ex$ then f_i, f_j and f_k must be in the same set. Note that at least one feature will be excluded from each set C_i.
- For each feature f_a of each set C_i, compute the sum of the weight of f_a and the weight of user precedences defined over f_a and f_b such that either f_b is in C_i or f_b is not in any set C_j. We call this value the *future cost* of excluding $f_a \in C_i$.
- Compute the forward cost by aggregating the least future cost of the features of each set C_i.

As the forward cost depends on the set ex we denote it as *fc(ex)*. For example, if $ex = \{\{f_1, f_2\}, \{f_2, f_3\}, \{f_3, f_4\}, \{f_5, f_6\}\}$, and if all the features are undecided then the forward cost of including f_2 depends on the pair $\{f_5, f_6\}$ of ex, since both f_5 and f_6 remain undecided immediately after including f_2. Similarly, the forward cost of excluding f_2 depends on the pairs $\{f_3, f_4\}$ and $\{f_5, f_6\}$ of ex. Let $ex^+_f(i)$ be the set of the pairs of features in ex based on which the forward cost of including a feature i is computed. Formally, $ex^+_f(i) = ex - \{\{j, k\} \in ex : k = i \vee \{j, i\} \in ex\}$. Let $ex^-_f(a)$ be the set of pairs of features in ex based on which the forward cost of excluding a feature i is computed. Formally, $ex^-_f(i) = ex - \{\{j, k\} \in ex : k = i\}$. Based on the forward cost with respect to the sets $ex^+_f(i)$ and $ex^-_f(a)$, the tighter upper bounds can be defined as follows:

$$\begin{aligned} ub^+_f(a) &= mv - (cc + c^+_f(a) + fc(ex^+_f(i))) \\ ub^-_f(a) &= mv - (cc + c^-_f(a) + fc(ex^-_f(i))). \end{aligned}$$

Similar bounds can be defined for $ub^+_p(\rho)$ and $ub^-_p(\rho)$.

6 Experimental Results

In this section, we empirically evaluate the performance of all the consistency techniques discussed in the previous sections, when used for finding an optimal relaxation of feature subscriptions within branch and bound search.

6.1 Problem Generation and Solvers

In order to compare the different consistency techniques we generated and experimented with a variety of *random catalogues* and many classes of *random feature subscriptions*. All the random selections below are performed with uniform distributions. A random catalogue is defined by a tuple $\langle n_c, m_c, T_c \rangle$. Here, n_c is the number of features, m_c is the number of precedence constraints and $T_c \subseteq \{\prec, \succ, \prec\!\succ\}$ is a set of types of constraints. Note that $f_i \prec\!\succ f_j$

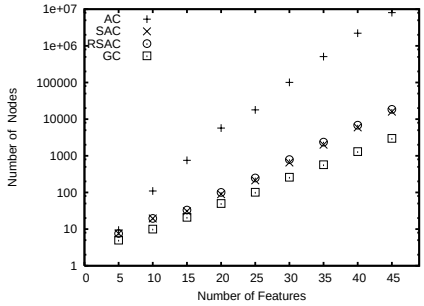

Figure 1. Results for subscriptions when the catalogue is $\langle 50, 750, \{\prec, \succ\}\rangle$.

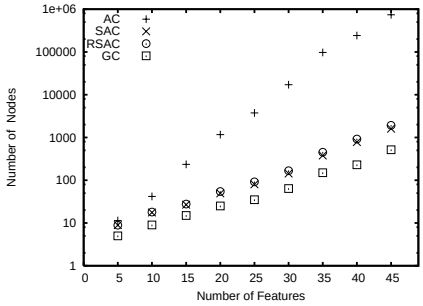

Figure 2. Results for subscriptions when the catalogue is $\langle 50, 500, \{\prec, \succ, \prec\succ\}\rangle$.

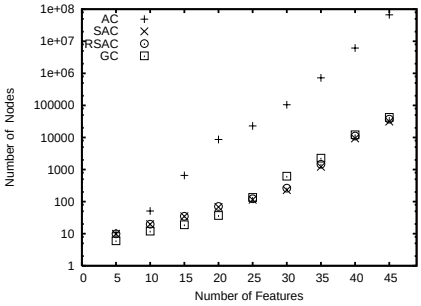

Figure 3. Results for subscriptions when the catalogue is $\langle 50, 250, \{\prec, \succ\}\rangle$.

means that in any given subscription both f_i and f_j cannot exist together. A random catalogue is generated by selecting m_c pairs of features randomly from $n_c(n_c-1)/2$ pairs of features. Each selected pair of features is then associated with a type of constraint that is selected randomly from T_c. A random feature subscription is defined by a tuple $\langle n_u, m_u, w\rangle$. Here, n_u is the number of features that are selected randomly from n_c features, m_u is the number of user precedence constraints between the pairs of features that are selected randomly from $n_u(n_u-1)/2$ pairs of features, and w is an integer greater than 0. Each feature and each user precedence constraint is associated with an integer weight that is selected randomly between 1 and w inclusive.

We generated catalogues of the following forms: $\langle 50, 250, \{\prec, \succ\}\rangle$, $\langle 50, 500, \{\prec, \succ, \prec\succ\}\rangle$ and $\langle 50, 750, \{\prec, \succ\}\rangle$. For each random catalogue, we generated $\langle 5, 5, 1\rangle$, $\langle 10, 10, 1\rangle, \ldots, \langle 45, 45, 1\rangle$ classes of random feature subscriptions. Note that the value of w is 4 by default, unless stated otherwise. For each class 10 instances were generated and their mean results are reported in this paper.

All the algorithms were implemented using Choco [7], a Java library for constraint programming systems. All the branch and bound search algorithms were equipped with a static version of the *dom/deg* variable ordering heuristic, where *dom* is the domain size and *deg* is the original degree of a variable. All the experiments were performed on a PC Pentium 4 (CPU 1.8 GHz and 768MB of RAM) processor. The performances of all the approaches are measured in terms of search nodes and runtime in seconds.

6.2 Empirical Comparison

We first investigated the effect of maintaining arc consistency, the results of which are indicated by AC. We then investigated, (1) maintaining singleton arc consistency on the Boolean variables and AC on the remaining variables, and (2) maintaining restricted singleton arc consistency on the Boolean variables and AC on the remaining variables; the former is denoted by SAC and the latter by RSAC. We also investigated the impact of using the filtering rules presented in Section 5.1.1 and 5.1.2 of the global constraint *SoftPrec*. The results obtained by them are denoted by GC.

The results shown in Figures 1–3 and Table 1 suggest that AC is inferior to the other approaches by several orders of magnitude. The results also suggest that maintaining (restricted) singleton arc consistency on the Boolean variables reduces the number of nodes and time significantly. To the best of our knowledge this is the first time that such a significant improvement has been observed by maintaining a partial form of singleton arc consistency. Note that RSAC usually visits more nodes than those visited by SAC, but the difference between them is not that significant. This suggest that the level of consistency enforced by RSAC on the instances of feature subscription problem is tantamount to SAC. Despite visiting more nodes, RSAC usually require less time than SAC.

For the catalogue $\langle 50, 250, \{\prec, \succ\}\rangle$, there is not much difference between (R)SAC and GC in terms of search nodes. However, as the number of the catalogue precedences increases, GC starts outperforming all the approaches in terms of the nodes. The more precedences in the catalogue the higher the chances of inferring new precedences. This may result in inferring more mutually exclusive pairs of features which in turn may help in computing tighter bounds and the result is fewer visited nodes. For instance, for the catalogue $\langle 50, 750, \{\prec, \succ\}\rangle$, when the number of features is 45, RSAC visits $18,469$ nodes and spends 149 seconds, on average, while GC visits $2,991$ nodes and spends 49 seconds, on average.

The difference in terms of time between GC and (R)SAC is not as big as the difference in terms of nodes. However, we believe that the gap will be more significant

Table 1. Comparison of consistency techniques in terms of time.

	$\langle 50, 750, \{\prec, \succ\}\rangle$				$\langle 50, 500, \{\prec, \succ, \prec\succ\}\rangle$				$\langle 50, 250, \{\prec, \succ\}\rangle$			
f_u	AC	SAC	RSAC	GC	AC	SAC	RSAC	GC	AC	SAC	RSAC	GC
15	0.09	0.04	0.03	0.05	0.02	0.02	0.02	0.04	0.05	0.02	0.02	0.04
20	0.82	0.23	0.20	0.19	0.12	0.08	0.07	0.15	0.96	0.09	0.08	0.14
25	3.38	0.94	0.83	0.67	0.50	0.21	0.19	0.38	3.21	0.29	0.27	0.96
30	24.27	3.92	3.52	2.20	2.49	0.54	0.48	0.85	20.34	1.16	1.04	6.65
35	146.72	15.01	13.68	6.68	16.18	1.82	1.69	2.04	183.49	9.13	8.70	34.92
40	779.77	50.62	47.34	17.79	47.73	4.43	4.09	3.94	1,836.39	81.77	76.59	181.51
45	3,468.08	164.83	149.36	49.54	164.89	11.03	10.19	9.43	23,009.72	345.10	320.01	1,047.37

when the pruning rules are implemented efficiently. Due to the the lack of space, we shall not go into the details of the implementation. Note that it also possible to schedule events so that more expensive rules are only activated after less expensive pruning rules. There is also a possibility of grouping events and activating rules based on the group of events. Also notice that in the current implementation the tighter bounds proposed in Section 5.3 are not computed. It would be interesting to investigate the impact of these tighter bounds on the number of nodes and time. Even though the implementation is not the best possible, we already observe some significant improvements for dense catalogues.

7 Conclusions

We studied four techniques for finding an optimal relaxation of feature subscriptions. First we presented a basic constraint programming formulation and investigated the impact of maintaining arc consistency and mixed consistencies. We then proposed a global constraint, called *SoftPrec*, along with filtering rules that exploit the structure of our problem and compute tighter upper bounds. We theoretically and experimentally compared all these techniques. Our empirical results suggest that maintaining arc consistency is inferior to the other techniques by several orders of magnitude. Maintaining (restricted) singleton arc consistency on the Boolean variables reduces the search space and the time significantly. The filtering rules of the global constraint are efficient in terms of pruning. When the catalogue is dense, all the other techniques are outperformed by these rules. In the future, we would like to like to characterise the level of consistency achieved by these rules.

Acknowledgements

This material is based upon works supported by the Science Foundation Ireland under Grant No. 05/IN/I886, and Embark Post Doctoral Fellowships No. CT1080049908 and No. CT1080049909. The authors would also like to thank Hadrien Cambazard for his support in using CHOCO.

References

[1] R. Barták and O. Čepek. Incremental filtering algorithms for precedence and dependency constraints. *International Journal on Artificial Intelligence Tools*, 17(1):205–221, 2008.

[2] C. Bessiere, K. Stergiou, and T. Walsh. Domain filtering consistencies for non-binary constraints. *Artificial Intelligence*, pages 800–822, 2007.

[3] G. W. Bond, E. Cheung, H. Purdy, P. Zave, and C. Ramming. An Open Architecture for Next-Generation Telecommunication Services. *ACM Transactions on Internet Technology*, 4(1):83–123, 2004.

[4] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec. Feature Interaction: A Critical Review and Considered Forecast. *Computer Networks*, 41(1):115–141, January 2003.

[5] R. Debruyne and C. Bessiére. Some practical filtering techniques for the constraint satifaction problem. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 412–417, Nagoya, Japan, 1997.

[6] M. Jackson and P. Zave. Distributed Feature Composition: a Virtual Architecture for Telecommunications Services. *IEEE TSE*, 24(10):831–847, October 1998.

[7] F. Laburthe and N. Jussien. JChoco: A java library for constraint programming.

[8] C. Lecoutre and P. Patrick. Maintaining singleton arc consistency. In *Proceedings of the 3rd International Workshop on Constraint Propagation And Implementation (CPAI'2006) held with CP'2006*, pages 47–61, Nantes, France, September, 2006.

[9] D. Lesaint, D. Mehta, B. O'Sullivan, L. Quesada, and N. Wilson. Personalisation of Telecommunications Services as Combinatorial Optimisation. In *IAAI-08*, pages 1693–1698, Chicago, USA, 2008. AAAI Press.

[10] D. Lesaint, D. Mehta, B. O'Sullivan, L. Quesada, and N. Wilson. Solving a Telecommunications Feature Subscription Configuration Problem. In *Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming*, Sydney, Australia, 2008.

[11] P. Prosser, K. Stergiou, and T. Walsh. Singleton Consistencies. In *CP-2000*, pages 353–368, Sept. 2000.

[12] M. Wallace. Practical applications of constraint programming. *Constraints Journal*, 1(1):139–168, September 1996.