Solving a Telecommunications Feature Subscription Configuration Problem

David Lesaint[1], Deepak Mehta[2], Barry O'Sullivan[2], Luis Quesada[2], and Nic Wilson[2]

[1] Intelligent Systems Research Centre, British Telecom, UK
david.lesaint@bt.com
[2] Cork Constraint Computation Centre, University College Cork, Ireland
{d.mehta,b.osullivan,l.quesada,n.wilson}@4c.ucc.ie

Abstract. Call control features (e.g., call-divert, voice-mail) are primitive options to which users can subscribe off-line to personalise their service. The configuration of a feature subscription involves choosing and sequencing features from a catalogue and is subject to constraints that prevent undesirable feature interactions at run-time. When the subscription requested by a user is inconsistent, one problem is to find an optimal relaxation. In this paper, we show that this problem is NP-hard and we present a constraint programming formulation using the variable weighted constraint satisfaction problem framework. We also present simple formulations using partial weighted maximum satisfiability and integer linear programming. We experimentally compare our formulations of the different approaches; the results suggest that our constraint programming approach is the best of the three overall.

1 Introduction

Information and communication services, from news feeds to internet telephony, are playing an increasing, and potentially disruptive, role in our daily lives. As a result, providers seek to develop personalisation solutions allowing customers to control and enrich their service. In telephony, for instance, personalisation relies on the provisioning of call control features. A feature is an increment of functionality which, if activated, modifies the basic service behaviour in systematic or non-systematic ways, e.g., do-not-disturb, multi-media ring-back tones, call-divert-on-busy, credit-card-calling, find-me.

Modern service delivery platforms provide the ability to implement features as modular applications and compose them on demand when setting up live sessions, that is, consistently with the feature subscriptions preconfigured by participants. In this context, a personalisation approach consists of exposing feature catalogues to subscribers and letting them select and sequence the features of their choice.

Not all sequences of features are acceptable though due to the possible occurrence of feature interactions. A feature interaction is "some way in which a feature modifies or influences the behaviour of another feature in generating the system's overall behaviour" [1]. For instance, a do-not-disturb feature will block any incoming call and cancel the effect of any subsequent feature subscribed by the callee. This is an undesirable interaction: as shown in Figure 1, the call originating from X will never reach

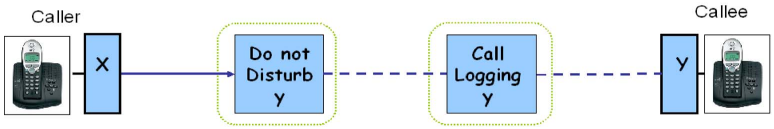

Fig. 1. An example of an undesirable feature interaction

call-logging. However, if call-logging is placed before do-not-disturb then both features will play their role.

Distributed Feature Composition (DFC) provides a method and a formal architecture model to address feature interactions [1–3]. 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 be together in any sequence. Undesirable interactions are then avoided by rejecting any sequence that does not satisfy the catalogue constraints.

A *feature subscription* is defined by a set of features, a set of user specified precedence constraints and a set of feature interaction constraints from the catalogue. The main 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 a relaxation of the feature subscription that is closest to the initial requirements of the user*.

In this paper, we shall show that checking the consistency of a feature subscription is polynomial in time, but finding an optimal relaxation of a feature subscription, when inconsistent, is NP-hard. We shall then present the formulation of finding an optimal relaxation using *constraint programming*. In particular, we shall use the variable weighted constraint satisfaction problem framework. In this framework, a branch and bound algorithm that maintains some level of consistency is usually used for finding an optimal solution. We shall investigate the impact of maintaining three different levels of consistency. The first one is Generalised Arc Consistency (GAC) [4], which is commonly used. The others are mixed consistencies. Here, mixed consistency means maintaining different levels of consistency on different sets of variables of a given problem. The first (second) mixed consistency enforces (a restricted version of) singleton GAC on some variables and GAC on the remaining variables of the problem.

We shall also consider *partial weighted maximum satisfiability*, an artificial intelligence technique, and *integer linear programming*, an operations research approach. We shall present the formulations using these approaches and shall discuss their differences with respect to the constraint programming formulation.

We have conducted experiments to compare the different approaches. The experiments are performed on a variety of random catalogues and random feature subscriptions. We shall present empirical results that demonstrate the superiority of maintaining

mixed consistency on the generalised arc consistency. For hard problems, we see a difference of up to three orders of magnitude in terms of search nodes and one order of magnitude in terms of time. Our results suggest that, when singleton generalised arc consistency is used, the constraint programming approach considerably outperforms our integer linear programming and partial weighted maximum satisfiability formulations. We highlight the factors that deteriorate the scalability of the latter approaches.

The rest of the paper is organised as follows. Section 2 provides an overview of the DFC architecture, its composition style and subscription configuration method. Section 3 presents the relevant definitions and theorems. Section 4 describes the constraint programming formulation for finding an optimal relaxation and discusses branch and bound algorithms that maintain different levels of consistency. The integer linear programming and partial weighted maximum satisfiability formulations of the problem are described in Section 5. The empirical evaluation of these approaches is shown in Section 6. Finally our conclusions are presented in Section 7.

2 Configuring Feature Subscriptions in DFC

In DFC each feature is implemented by one or more modules called *feature box types* (FBT) and each FBT has many run-time instances called *feature boxes*. We assume in this paper that each feature is implemented by a single FBT and we associate features with FBTs. As shown in Figure 2, a call session between two end-points is set up by chaining feature boxes. The routing method decomposes the connection path into a source and a target region and each region into *zones*. A source (target) zone is a sequence of feature boxes that execute for the same source (target) address.

The first source zone is associated with the source address encapsulated in the initial setup request, e.g., zone of X in Figure 2. A change of source address in the source region, caused for instance by an identification feature, triggers the creation of a new source zone [5]. If no such change occurs in a source zone and the zone cannot be expanded further, routers switch to the target region. Likewise, a change of target address in the target region, as performed by Time-Dependent-Routing (TDR) in Figure 2, triggers the creation of a new target zone. If no such change occurs in a target zone and the zone cannot be expanded further (as for Z in Figure 2), the request is sent to the final box identified by the encapsulated target address.

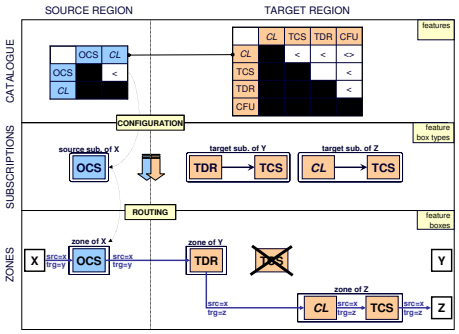

Fig. 2. DFC: Catalogues, subscriptions and sessions.

DFC routers are only concerned with locating feature boxes and assembling zones into regions. They do not make decisions as to the type of feature boxes (the FBTs)

appearing in zones or their ordering. They simply fetch this information from the *feature subscriptions* that are preconfigured for each address in each region based on the *catalogue* published by the service provider.

A catalogue is a set of features subject to precedence and exclusion constraints. Features fall into three classes: *source*, *target* and *reversible*, i.e., a subset of features that are both source and target. Constraints are formulated by designers on pairs of source features and pairs of target features to prevent undesirable feature interactions in each zone [6]. Specifically, a precedence constraint imposes a routing order between two features, as for the case of Terminating-Call-Screening (TCS) and Call-Logging (CL) in Figure 2. An exclusion constraint makes two features mutually exclusive, as for the case of CL and Call-Forwarding-Unconditional (CFU) in Figure 2.

A subscription is a subset of catalogue features and a set of user precedence constraints between features in each region. For instance, the subscription of Y in the target region includes the user precedence TDR$\prec$TCS. Configuring a subscription involves selecting, parameterising and sequencing features in each region consistently with the catalogue constraints and other integrity rules [3]. In particular, the source and target regions of a subscription must include the same reversible features in inverse order, i.e. source and target regions are not configured independently.

3 Formal Definitions

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 $\langle f_j, f_i \rangle$.

Definition 1 (Feature Catalogue). *A catalogue is a tuple $\langle F, P \rangle$, where F is a set of features that are available to users and P is a set of precedence constraints on F.*

The *transpose* of a catalogue $\langle F, P \rangle$ is the catalogue $\langle F, P^T \rangle$ such that $\forall \langle f_i, f_j \rangle \in F^2 : \langle f_i, f_j \rangle \in P \Leftrightarrow \langle f_j, f_i \rangle \in P^T$. In DFC the precedence constraints between the features in the source (target) catalogue are specified with respect to the direction of the call. For the purpose of configuration, we combine the source catalogue $\langle F_s, P_s \rangle$ and the target catalogue $\langle F_t, P_t \rangle$ into a single catalogue $\langle F_c, P_c \rangle \equiv \langle F_s \cup F_t, P_s \cup {P_t}^T \rangle$.

Definition 2 (Feature Subscription). *A feature subscription S of catalogue $\langle F_c, P_c \rangle$ is a tuple $\langle F, C, U, W_F, W_U \rangle$, where $F \subseteq F_c$, C is the projection of P_c on F, i.e., $P_c{\downarrow_F} = \{f_i \prec f_j \in P_c : \{f_i, f_j\} \subseteq F\}$, U is a set of (user defined) precedence constraints on F, $W_F : F \to \mathbb{N}$ is a function that assigns weights to features and $W_U : U \to \mathbb{N}$ is a function that assigns weights to user precedence constraints. The value of S is defined by* $\mathrm{Value}(S) = \sum_{f \in F} W_F(f) + \sum_{p \in U} W_U(p)$.

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.

Definition 3 (Consistency). *A feature subscription $\langle F, C, U, W_F, W_U \rangle$ of some catalogue is defined to be consistent if and only if the directed graph $\langle F, C \cup U \rangle$ is acyclic.*

Due to the composition of the source and target catalogues into a single catalogue, a feature subscription S is consistent if and only if both source and target regions are consistent in the DFC sense.

Theorem 1 (Complexity of Consistency Checking). *Determining whether a feature subscription $\langle F, C, U, W_F, W_U\rangle$ is consistent or not can be checked in $\mathcal{O}(|F| + |C| + |U|)$.*

Proof. We use *Topological Sort* [7]. In Topological Sort we are interested in ordering the nodes of a directed graph such that if the edge $\langle i, j\rangle$ is in the set of edges of the graph then node i is less than node j in the order. In order to use *Topological Sort* for detecting whether a feature subscription is consistent, we associate nodes with features and edges with precedence constraints. Then, the subscription is consistent iff for all edges $\langle i, j\rangle$ in the graph associated with the subscription we have that $i \prec j$ in the order computed by *Topological Sort*. As the complexity of *Topological Sort* is linear with respect to the size of the graph, detecting whether a feature subscription is consistent is $\mathcal{O}(|F| + |C| + |U|)$. □

If an input feature subscription is not consistent then the task is to relax the given feature subscription by dropping one or more features or user precedence constraints to generate a consistent feature subscription with maximum value.

Definition 4 (Relaxation). *A* relaxation *of a feature subscription $\langle F, C, U, W_F, W_U\rangle$ is a subscription $\langle F', C', U', W'_F, W'_U\rangle$ such that $F' \subseteq F$, $C' = P_c{\downarrow_{F'}}$, $U' \subseteq U{\downarrow_{F'}}$, $W_{F'}$ is W_F restricted to F', and $W_{U'}$ is W_U restricted to U'.*

Definition 5 (Optimal Relaxation). *Let R_S be the set of all consistent relaxations of a feature subscription S. We say that $S_i \in R_S$ is an optimal relaxation of S if it has maximum value among all relaxations, i.e., if and only if there does not exist $S_j \in R_S$ such that* $\mathrm{Value}(S_j) > \mathrm{Value}(S_i)$.

Theorem 2 (Complexity of Finding an Optimal Relaxation). *Finding an optimal relaxation of a feature subscription is NP-hard.*

Proof. Given a directed graph $\langle V, E\rangle$, the *Feedback Vertex Set Problem* is to find a smallest $V' \subseteq V$ whose deletion makes the graph acyclic. This problem is known to be NP-hard [8]. We prove that finding an optimal relaxation is NP-hard by reducing the feedback vertex set problem to the latter. Given a feature subscription $S = \langle F, C, U, W_F, W_U\rangle$, the feedback vertex set problem can be reduced to our problem by associating the nodes of the directed graph V with features F, the edges E with catalogue precedence constraints C, the empty set $\emptyset$ with U, and the constant function that maps every element of its domain to 1 ($\lambda x.1$) with both W_F and W_U. Notice that, as $U = \emptyset$, the only way of finding an optimal relaxation of S is by removing a set of features from F. Assuming that an optimal relaxation is $S' = \langle F', C', U', W'_F, W'_U\rangle$, the set of features $F - F'$ corresponds to the smallest set of nodes V' whose deletion makes the directed graph acyclic. Thus, we can conclude that finding an optimal relaxation S' is NP-hard. □

4 A Constraint Programming Approach

Constraint programming has been successfully used in many applications such as planning, scheduling, resource allocation, routing, and bio-informatics [9]. 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.

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) the 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.

4.1 Formulation

In this section we model the problem of finding an optimal relaxation of a feature subscription $\langle F, C, U, W_F, W_U \rangle$ 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 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 $p_{ij} \equiv (f_i \prec f_j) \in U$ 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.

Constraints. A catalogue precedence constraint $p_{ij} \in C$ that feature f_i should be before feature f_j can be expressed as follows:

$$bf_i \wedge bf_j \;\Rightarrow\; (pf_i < pf_j).$$

Note that the constraint is activated only if the selection variables bf_i and bf_j are instantiated to 1. A user precedence constraint $p_{ij} \in U$ that f_i should be placed before f_j in their subscription can be expressed as follows:

$$bp_{ij} \;\Rightarrow\; (bf_i \wedge bf_j \wedge (pf_i < pf_j)).$$

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 of finding an optimal relaxation of a feature subscription can be expressed as follows:

$$\text{Maximise} \sum_{f_i \in F} bf_i \times W_F(f_i) + \sum_{p_{ij} \in U} bp_{ij} \times W_U(p_{ij}).$$

4.2 Solution Technique

A depth-first branch and bound algorithm (BB) is generally used to find an optimal solution. In case of maximisation, BB keeps the current optimal value of the solution while traversing the search tree. This value is a lower bound (lb) of the objective function. At each node of the search tree BB computes an overestimation of the global value. This value is an upper bound (ub) of the best solution that can be found as long as the current search path is maintained. If $ub \leq lb$, 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.

Enforcing local consistency enables the computation of $ub_{(i,a)}$, which is a specialisation of ub for a value a of an unassigned variable i. If $ub_{(i,a)} \leq lb$, then value a can be removed because it will not be present in any solution better than the current one. Removed values are restored when BB backtracks above the node where they were eliminated. The quality of the upper bound can be improved by increasing the level of local consistency that is maintained at each node of the search tree. The different levels of local consistencies that we have considered are *generalised Arc Consistency* (GAC) [4] and *mixed consistency* [10].

A problem is said to be *generalised arc consistent* 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 problem is said to be *singleton generalised arc consistent* [11] if it has non-empty domains and for any assignment of a variable, the resulting subproblem can be made GAC. Enforcing Singleton generalised Arc Consistency (SGAC) in a SAC-1 manner [12] works by having an outer loop consisting of variable-value pairs of the form (x, a). For each a in the domain of x if there is a domain wipeout while enforcing arc consistency then a is removed from the domain of x and arc consistency is enforced. The main problem with SAC-1 is that deleting a single value triggers the outer loop again. The Restricted SAC (RSAC) algorithm avoids this triggering by considering each variable-value pair only once [13].

Mixed consistency means maintaining different levels of consistency on different variables of a problem. In [14] it has been shown that maintaining mixed consistency, in particular maintaining SAC on some variables and maintaining arc consistency on some variables, can reduce the solution time for some CSPs. In this paper we shall study the effect of maintaining different levels of consistency on different sets of variables within a branch and bound search. We shall investigate the effect of Maintaining generalised Singleton Arc Consistency (MGSAC) on the Boolean variables and Maintaining generalised Arc Consistency (MGAC) on the remaining variables of the problem. We shall also investigate the effect of Maintaining Restricted Singleton generalised Arc Consistency (MRSGAC) on the Boolean variables and MGAC on the remaining variables. The former shall be denoted by MSGAC_b and the latter by MRGSAC_b. Results presented in

Section 6 suggest that maintaining singleton generalised arc consistency on the Boolean variables of the random instances of the feature subscription configuration problem reduces the search space and time of the branch and bound algorithm significantly.

5 Other Approaches

We present a partial weighted maximum Boolean satisfiability and an integer linear programming formulation for finding an optimal relaxation of a feature subscription.

5.1 Partial Weighted Maximum Boolean Satisfiability

The Boolean Satisfiability Problem (SAT) is a decision problem whose instance is an expression in propositional logic written using only $\wedge$, $\vee$, $\neg$, variables and parentheses. The problem is to decide whether there is an assignment of *true* and *false* values to the variables that will make the expression *true*. The expression is normally written in conjunctive normal form. The Partial Weighted Maximum Boolean Satisfiability Problem (PWMSAT) is an extension of SAT that includes the notions of hard and soft clauses. Any solution should respect the hard clauses. Soft clauses are associated with weights. The goal is to find an assignment that maximises the sum of the weights of the satisfied clauses. The PWMSAT formulation of finding an optimal relaxation of a feature subscription $\langle F, C, U, W_F, W_U \rangle$ is outlined below.

Variables. Let *PrecDom* be the set of possible precedence constraints that can be defined on F, i.e., $\{f_i \prec f_j : \{f_i, f_j\} \subseteq F \wedge f_i \neq f_j\}$). For each feature $f_i \in F$ there is a Boolean variable bf_i, which is true or false depending on whether feature f_i is included or not in the computed subscription. For each precedence constraint p_{ij} there is a Boolean variable bp_{ij}, which is true or false depending on whether the precedence constraint $f_i \prec f_j$ holds or not in the computed subscription.

Clauses. In our model, clauses are represented with a tuple $\langle w, c \rangle$, where w is the weight of clause and c is the clause itself. Note that the hard clauses are associated with weight $\top$, which represents an infinite penalty for not satisfying the clause. Each precedence constraint $p_{ij} \in C$ must be satisfied if the features f_i and f_j are included in the computed subscription. We model this by adding the following clause

$$\langle \top, (\neg bf_i \vee \neg bf_j \vee bp_{ij}) \rangle.$$

The precedence relation should be transitive and asymmetric in order to ensure that the subscription graph is acyclic. In order to ensure this, for every $\{p_{ij}, p_{jk}\} \subseteq \textit{PrecDom}$, we add the following clause:

$$\langle \top, (\neg bp_{ij} \vee \neg bp_{jk} \vee bp_{ik}) \rangle. \tag{1}$$

Note that Rule (1) need only be applied to $\langle i, j, k \rangle$ such that $i \neq k$ because of Rule (2) below. In our model, both bp_{ij} and bp_{ji} can be false. However, if one of them is true

the other one is false. As this should be the case for any precedence relation, we add the following clause for every $p_{ij} \in \mathit{PrecDom}$:

$$\langle \top, (\neg \mathit{bp}_{ij} \vee \neg \mathit{bp}_{ji}) \rangle. \tag{2}$$

We make sure that each precedence constraint $p_{ij} \in \mathit{PrecDom}$ is only satisfied when its features are included by considering the following clauses:

$$\langle \top, (\mathit{bf}_i \vee \neg \mathit{bp}_{ij}) \rangle \quad \langle \top, (\mathit{bf}_j \vee \neg \mathit{bp}_{ij}) \rangle.$$

We need to penalise any solution that does not include a feature $f_i \in F$ or a user precedence constraint $p_{ij} \in U$. This is done by adding the following clauses:

$$\langle \mathit{wf}_i, (\mathit{bf}_i) \rangle \quad \langle \mathit{wp}_{ij}, (\mathit{bp}_{ij}) \rangle,$$

where $\mathit{wf}_i = W_F(f_i)$ and $\mathit{wp}_{ij} = W_U(\langle f_i, f_j \rangle)$. The cost of violating these clauses is the weight of the feature f_i and the weight of the precedence constraint p_{ij} respectively.

The number of Boolean variables in the PWMSAT model (approximately $|F|^2$) is greater than the number of Boolean variables in the CP model ($|F| + |U|$). These extra variables are used by Rule (1) and (2) to avoid cycles in the final subscription graph. We remark that the subscription contains a cycle if and only if the transitive closure of $C \cup U$ contains a cycle. Therefore, it is sufficient to associate Boolean variables only with the precedence constraints in the transitive closure of $C \cup U$. Reducing these variables will also reduce the transitive clauses, especially when the input subscription graph is not dense. Otherwise, Rule (1) will generate $|F| \times (|F| - 1) \times (|F| - 2)$ transitivity clauses. For example, for the subscription $\langle F, C, U, W_F, W_U \rangle$ with $F = \{f_1, f_2, f_3, f_4, f_5, f_6\}$, $C = \{p_{12}, p_{21}, p_{34}, p_{43}, p_{56}, p_{65}\}$, and $U = \emptyset$, Rule (1) will generate 120 transitive clauses. Since any relaxation of the given subscription respecting the clauses generated by Rule (2) is acyclic, the 120 transitive clauses are useless. Thus, if *PrecDom* is instead set to be the transitive closure of $C \cup U$, then Rule (1) would not generate any clause for the mentioned example. Another way to reduce the number of transitive clauses is by not considering the ones where $\{p_{ji}, p_{kj}, p_{ik}\} \cap C \neq \emptyset$, especially when the input subscription graph is not sparse. The reason is that these transitive clauses are always entailed due to the enforcement of the catalogue precedence constraints.

Note that the two techniques described before for reducing the number of transitive clauses complement each other. This reduction in the number of clauses might have an impact on the runtime of the PWMSAT approach, since less memory might be needed. Even though it is sufficient to associate a Boolean variable with each precedence constraint in the transitive closure of $C \cup U$, it is still greater than $|F| + |U|$. Another way of reducing the number of variables is to associate a feature with a finite domain variable representing its position (as done in the CP model), log-encode the finite domain variables, and express the precedence constraints using a lexicographical comparator [15]. This approach indeed uses fewer variables than the implemented approach since only $|F| \times \log |F|$ variables are needed for encoding the position variables. However, it is not so straightforward to automatically translate the resulting Boolean formula into its corresponding conjunctive normal form.

5.2 Integer Linear Programming

In Linear Programming the goal is to optimise an objective function subject to linear equality and inequality constraints. When all the variables are forced to be integer-valued, the problem is an Integer Linear Programming (ILP) problem. The standard way of expressing these problems is by presenting the function to be optimised, the linear constraints to be respected and the domain of the variables involved. Both the CP and the PWMSAT formulations for finding an optimal relaxation of a feature subscription $\langle F, C, U, W_F, W_U \rangle$ can be modeled in ILP. The translation of the PWMSAT formulation into ILP formulation is straightforward. For this particular model, we observed that CPLEX was not able to solve even simple problems within a time limit of 4 hours. Due to the lack of space we shall describe neither the formulation nor its corresponding results. The ILP formulation that is equivalent to the CP formulation is outlined below.

Variables. For each $f_i \in F$, we use a binary variable bf_i and an integer variable pf_i. A binary variable bf_i is equal to 1 or 0 depending on whether feature f_i is included or not. An integer variable pf_i is the position of feature f_i in the final subscription. For each user precedence constraint $p_{ij} \in U$, we use a binary variable bp_{ij}. It is instantiated to 1 or 0 depending on whether the precedence constraint $f_i \prec f_j$ holds or not.

Linear Inequalities. If the features f_i and f_j are included in the computed subscription and if $p_{ij} \in C$ then the position of feature f_i must be less than the position of feature f_j. To this effect, we need to translate the underlying implication $(bf_i \wedge bf_j \Rightarrow (pf_i < pf_j))$ into the following linear inequality:

$$pf_i - pf_j + n * bf_i + n * bf_j \leq 2n - 1\ . \tag{3}$$

Here, n is a constant that is used to refer to the number of features $|F|$ selected by the user. When both bf_i and bf_j are 1, Inequality (3) will force $(pf_i < pf_j)$. Note that this is not required for any user precedence constraint $p_{ij} \in U$, since it can be violated.

A user precedence $p_{ij} \in U$ is equivalent to the implication $bp_{ij} \Rightarrow pf_i < pf_j \wedge bf_i \wedge bf_j$, which in turn is equivalent to the conjunction of the three implications $(bp_{ij} \Rightarrow (pf_i < pf_j))$, $(bp_{ij} \Rightarrow bf_i)$ and $(bp_{ij} \Rightarrow bf_j)$. These implications can be translated into the following inequalities:

$$pf_i - pf_j + n * bp_{ij} \leq n - 1 \tag{4}$$

$$bp_{ij} - bf_i \leq 0 \tag{5}$$

$$bp_{ij} - bf_j \leq 0\ . \tag{6}$$

Inequality (4) means that $bp_{ij} = 1$ forces $pf_i < pf_j$ to be true. Also, if $bp_{ij} = 1$ then both bf_i and bf_j are equal to 1 from Inequalities (5) and (6) respectively.

Objective Function. The objective is to find an optimal relaxation of a feature subscription configuration problem $\langle F, C, U, W_F, W_U \rangle$ that maximises the sum of the weights of the features and the user precedence constraints that are selected:

$$\text{Maximise} \sum_{f_i \in F} wf_i\, bf_i + \sum_{p_{ij} \in U} wp_{ij}\, bp_{ij}.$$

6 Experimental Results

In this section, we shall describe the empirical evaluation of finding an optimal relaxation of randomly generated feature subscriptions using constraint programming, partial weighted maximum Boolean satisfiability and integer linear programming.

6.1 Problem Generation and Solvers

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 $\langle f_c, B_c, T_c \rangle$. Here, f_c is the number of features, B_c is the number of binary constraints and $T_c \subseteq \{<, >, <>\}$ is a set of types of constraints. Note that $f_i <> f_j$ means that in any given subscription both f_i and f_j cannot exist together. A random catalogue is generated by selecting B_c pairs of features randomly from $f_c(f_c - 1)/2$ 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 $\langle f_u, p_u, w \rangle$. Here, f_u is the number of features that are selected randomly from f_c features, p_u is the number of user precedence constraints between the pairs of features that are selected randomly from $f_u(f_u - 1)/2$ 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: $\langle 50, 250, \{<, >\} \rangle$, $\langle 50, 500, \{<, >, <>\} \rangle$ and $\langle 50, 750, \{<, >\} \rangle$. For each random catalogue, we generated classes of feature subscriptions of the following forms: $\langle 10, 5, 4 \rangle$, $\langle 15, 20, 4 \rangle$, $\langle 20, 10, 4 \rangle$, $\langle 25, 40, 4 \rangle$, $\langle 30, 20, 4 \rangle$, $\langle 35, 35, 4 \rangle$, $\langle 40, 40, 4 \rangle$, $\langle 45, 90, 4 \rangle$ and $\langle 50, 5, 4 \rangle$. Note that $\langle 50, 250, \{<, >\} \rangle$ is the default catalogue by and the value of w is 4 by default, unless stated otherwise. For the catalogue $\langle 50, 250, \{<, >\} \rangle$ we also generated $\langle 5, 0, 1 \rangle$, $\langle 10, 0, 1 \rangle, \ldots, \langle 50, 0, 1 \rangle$ and $\langle 5, 5, 1 \rangle$, $\langle 10, 10, 1 \rangle, \ldots, \langle 50, 50, 1 \rangle$ classes of random feature subscriptions. For each class 10 instances were generated and their mean results are reported in this paper.

The CP model was implemented and solved using CHOCO [16], a Java library for constraint programming systems. The PWMSAT model of the problem was implemented and solved using SAT4J [17], an efficient library of SAT solvers in Java. The ILP model of the problem was solved using ILOG CPLEX [18]. 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 (#nodes) and runtime in milliseconds (time). We used the time limit of 4 hours to cut the search.

6.2 Maintaining Different Levels of Consistency in CP

For the CP model, we first investigated the effect of Maintaining generalised Arc Consistency (MGAC) during branch and bound search. We then studied the effect of maintaining different levels of consistency on different sets of variables within a problem. In particular we investigated, (1) maintaining generalised singleton arc consistency on the Boolean variables and MGAC on the remaining variables, and (2) maintaining restricted

singleton generalised arc consistency on the Boolean variables and MGAC on the remaining variables; the former is denoted by MSGAC$_b$ and the latter by MRSGAC$_b$. The results are presented in Table 1 for these three branch and bound search algorithms.

Table 1 clearly shows that maintaining (R)SGAC on the Boolean variables and GAC on the integer variables dominates maintaining GAC on all the variables. 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. We also see that there is no difference in the number of nodes visited by MRSGAC$_b$ and MSGAC$_b$ for the first two classes of feature subscriptions. However, as the problem size increases the difference in terms of the number of nodes also increases significantly. Note that in the remainder of the paper the results that correspond to the CP approach are obtained by using MSGAC$_b$ algorithm.

	MGAC		MRSGAC$_b$		MSGAC$_b$	
$\langle f, p \rangle$	time	#nodes	time	#nodes	time	#nodes
$\langle 10, 5 \rangle$	17	21	23	16	26	16
$\langle 15, 20 \rangle$	92	726	34	41	42	41
$\langle 20, 10 \rangle$	203	1,694	39	47	50	46
$\langle 25, 40 \rangle$	14,985	88,407	595	187	678	169
$\langle 30, 20 \rangle$	6,073	29,211	653	184	768	161
$\langle 35, 35 \rangle$	124,220	481,364	7,431	1,279	8,379	1,093
$\langle 40, 40 \rangle$	1,644,624	5,311,838	67,798	9,838	76,667	8,475

Table 1. Average results of MGAC, MRSGAC$_b$ and MSGAC$_b$.

6.3 Comparison between the Alternative Approaches

The performances of using constraint programming (CP), partial weighted maximum satisfiability (PWMSAT) and integer linear programming (CPLEX) approaches are presented in Tables 2 and 3. If any approach failed to find and prove an optimal relaxation within a time limit of 4 hours then that time limit is used as the runtime of the algorithm and the number of nodes visited in that time limit is used as the number of nodes of the algorithm in order to compute the average runtime and average search nodes of a given problem class. In the tables, the column labelled as #us is used to denote the number of instances for which the time limit was exceeded. If this column is not present for any approach then it means that all the instances of all the problem classes were solved within the time limit. In general finding an optimal relaxation is NP-hard. Therefore, we need to investigate which approach can do it in reasonable time.

Tables 2 and 3 suggest that our CP approach performs better than our ILP and PWMSAT approaches. Although in very few cases the CP approach is outperformed by the other two approaches, it performs significantly better in all other cases. Nevertheless,

Table 2. Catalogue $\langle 50, 250, \{<, >\} \rangle$.

	optimal	PWMSAT			CPLEX			CP	
$\langle f, p \rangle$	value	#nodes	time	#us	#nodes	time	#us	#nodes	time
$\langle 10, 5 \rangle$	36	167	345	0	0	11	0	16	23
$\langle 15, 20 \rangle$	69	721	1,039	0	51	61	0	41	34
$\langle 20, 10 \rangle$	62	1,295	1,619	0	50	47	0	47	39
$\langle 25, 40 \rangle$	115	5,039	4,391	0	3,482	1,945	0	187	595
$\langle 30, 20 \rangle$	93	5,415	6,397	0	1,901	1,025	0	184	653
$\langle 35, 35 \rangle$	118	30,135	23,955	0	35,247	22,763	0	1,279	7,431
$\langle 40, 40 \rangle$	123	186,913	282,760	0	299,829	247,140	0	9,838	67,798
$\langle 45, 90 \rangle$	173	6,291,957	12,638,251	8	5,280,594	7,690,899	2	104,729	1,115,515
$\langle 50, 4 \rangle$	96	165,928	195,717	0	1,164,755	1,010,383	0	60,292	413,611

it is also true that a remarkable improvement in our CP approach is due to maintaining (restricted) singleton arc consistency on the Boolean variables. For example, for feature subscription $\langle 40, 40\rangle$ and catalogue $\langle 50, 250, \{<, >\}\rangle$ constraint programming (with MSGAC$_b$), on average, requires approximately only 1 minute whereas MGAC requires approximately half an hour.

The CP approach solved all the instances within the time limit. CPLEX could not solve 2 instances. More precisely, it could not prove their optimality within the time limit. SAT4J exceeded the time limit for 9 instances. This could be a consequence of $\mathcal{O}(n^3)$ transitive clauses, where $n = |F|$. Figure 3 depicts a plot between the number of clauses and the runtime of SAT4J. This plot clearly suggests that the runtime of SAT4J increases as the number of clauses increases. The high number of clauses restricts the scalability of the PWMSAT approach. For large instances SAT4J also runs out of the default memory (64MB). For instance, for catalogue $\langle 50, 250, \{<, >\}\rangle$ and feature subscription $\langle 45, 90\rangle$, SAT4J runs out of memory when solving one of the instances. Note that the results for SAT4J presented in this section correspond to the instances that are generated after reducing the variables and the clauses by applying the techniques described in Section 5.1. The application of these techniques reduces the runtime up to 65%. However, this only enabled one of the previously unsolvable instances to be solved.

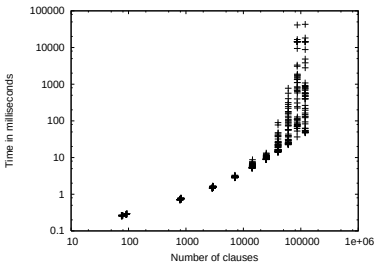

Fig. 3. Clauses vs Time

Figure 4 presents the comparison of the different approaches in terms of their runtimes for the subscriptions, when $U = \emptyset$ and the weight of each feature is 1. The runtimes of the approaches for the instances when $|F| = |U|$ are presented in Figure 5. Overall, the CP approach performs best. Although, the SAT4J solver performs best when $|F| > 35$ and $U = \emptyset$, it would be interesting to find out whether its performance will deteriorate when $|F| > 50$. In Figure 5, when $|F| = 50$, neither the ILP approach nor the PWMSAT approach managed to solve all the instances. This is the reason that their average runtimes, for the case of 50 features, are close to the timeout. If the timeout

Table 3. Results for more dense catalogues.

	Catalogue $\langle 50, 500, \{<, >, <>\}\rangle$						Catalogue $\langle 50, 750, \{<, >\}\rangle$						
	PWMSAT		CPLEX		CP		PWMSAT			CPLEX		CP	
$\langle f, p\rangle$	#nodes	time	#nodes	time	#nodes	time	#nodes	time	#us	#nodes	time	#nodes	time
$\langle 10, 5\rangle$	326	528	0	10	13	3	246	500	0	28	33	16	7
$\langle 15, 20\rangle$	1,066	1,173	4	53	31	28	1,111	985	0	306	261	40	45
$\langle 20, 10\rangle$	2,583	1,981	18	85	49	59	2,484	1,542	0	798	540	82	145
$\langle 25, 40\rangle$	5,753	2,961	76	554	110	250	6,904	3,158	0	7,043	5,741	236	910
$\langle 30, 20\rangle$	9,738	4,092	90	447	158	417	11,841	5,025	0	22,253	18,461	591	2,381
$\langle 35, 35\rangle$	12,584	6,841	300	1,824	461	1,643	31,214	18,278	0	109,472	126,354	2,288	12,879
$\langle 40, 40\rangle$	22,486	11,310	711	3,018	892	3,914	68,112	92,105	0	354,454	514,275	6,363	42,268
$\langle 45, 90\rangle$	60,504	59,267	2,130	17,452	2,286	14,803	602,192	2,443,228	1	1,969,716	3,780,539	19,909	188,826
$\langle 50, 4\rangle$	43,765	21,472	1,500	3,771	4,208	16,921	184,584	319,531	0	1,646,752	3,162,084	51,063	342,492

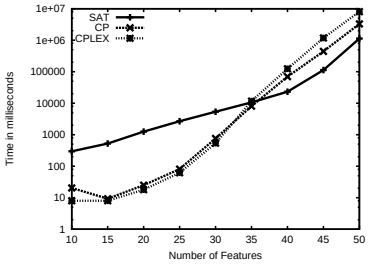

Fig. 4. Results for $\langle f_u, 0, 1\rangle$, where f_u varies from 5 to 50 in steps of 5.

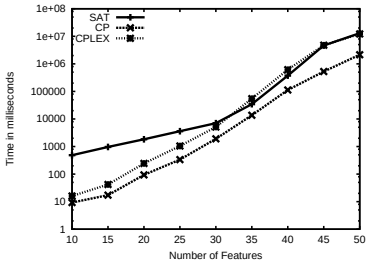

Fig. 5. Results for $\langle f_u, p_u, 1\rangle$, where $f_u = p_u$ and f_u varies from 5 to 50 in steps of 5.

was higher, the gap between the CP approach and the other approaches, for the case of 50 features in Figure 5 would be even more significant.

7 Conclusions

We presented, and evaluated, three optimisation-based approaches to finding optimal reconfigurations of call-control features when the user's preferences violate the technical constraints defined by a set of DFC rules. We proved that finding an optimal relaxation of a feature subscription is NP-hard. For the constraint programming approach, we studied the effect of maintaining generalised arc consistency and two mixed consistencies during branch and bound search. Our experimental results suggest that maintaining (restricted) generalised singleton arc consistency on the Boolean variables and generalised arc consistency on the integer variables outperforms MGAC significantly. Our results also suggest that the CP approach when applied with stronger consistency, is able to scale well compared to the other approaches. Finding an optimal relaxation for a reasonable size catalogue (e.g., [19] refers to a catalogue with up to 25 features) is feasible using constraint programming.

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, Daniel Le Berre and Alan Holland for their support in using CHOCO, SAT4J and CPLEX respectively.

References

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

2. Jackson, M., Zave, P.: Distributed Feature Composition: a Virtual Architecture for Telecommunications Services. IEEE TSE **24**(10) (October 1998) 831–847
3. Jackson, M., Zave, P.: The DFC Manual. AT&T. (November 2003)
4. Bessiere, C.: Constraint propagation. Technical Report 06020, LIRMM, Montpellier,France (March 2006)
5. Zave, P., Goguen, H., Smith, T.M.: Component Coordination: a Telecommunication Case Study. Computer Networks **45**(5) (August 2004) 645–664
6. Zave, P.: An Experiment in Feature Engineering. In McIver, A., Morgan, C., eds.: Programming Methodology. Springer-Verlag (2003) 353–377
7. Cormen, T., Leiserson, C., Rivest, R.: Introduction to Algorithms. The MIT Press (1990)
8. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the The Theory of NP-Completeness. W. H. Freeman and Company (1979)
9. Wallace, M.: Practical applications of constraint programming. Constraints Journal **1**(1) (September 1996) 139–168
10. Dooms, G., Deville, Y., Dupont, P.: CP(Graph): Introducing a graph computation domain in constraint programming. In: CP 2005. (2005)
11. Bessiere, C., Stergiou, K., Walsh, T.: Domain filtering consistencies for non-binary constraints. Artificial Intelligence (2007)
12. Debruyne, R., Bessiére, C.: Some practical filtering techniques for the constraint satifaction problem. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence, Nagoya, Japan (1997) 412–417
13. Prosser, P., Stergiou, K., Walsh, T.: Singleton Consistencies. In Dechter, R., ed.: CP-2000. (September 2000) 353–368
14. Lecoutre, C., Patrick, P.: Maintaining singleton arc consistency. In: Proceedings of the 3rd International Workshop on Constraint Propagation And Implementation (CPAI'2006) held with CP'2006, Nantes, France (September, 2006) 47–61
15. Hawkins, P., Lagoon, V., Stuckey, P.J.: Solving set constraint satisfaction problems using ROBDDs. Journal of Artificial Intelligence Research **24** (2005) 109 – 156
16. Laburthe, F., Jussien, N.: JChoco: A java library for constraint programming.
17. Le Berre, D.: SAT4J: An efficient library of SAT solvers in java.
18. ILOG: CPLEX solver 10.1 http://www.ilog.com/products/cplex/.
19. Bond, G.W., Cheung, E., Goguen, H., Hanson, K.J., Henderson, D., Karam, G.M., Purdy, K.H., Smith, T.M., Zave, P.: Experience with Component-Based Development of a Telecommunication Service. In: CBSE 2005. (2005) 298–305