A BDD Approach to the Feature Subscription Problem

T. Hadzic\(^1\) and D. Lesaint\(^2\) and D. Mehta\(^3\) and B. O’Sullivan\(^4\) and L. Quesada\(^5\) and N. Wilson\(^6\)

Abstract. Modern feature-rich telecommunications services offer significant opportunities to human users. To make these services more usable, facilitating personalisation is very important since it enhances the users’ experience considerably. However, regardless how service providers organise their catalogues of features, they cannot achieve complete configurability due to the existence of feature interactions. Distributed Feature Composition (DFC) provides a comprehensive methodology, underpinned by a formal architecture model to address this issue. In this paper we present an approach based on using Binary Decision Diagrams (BDD) to find optimal reconfigurations of features when a user’s preferences violate the technical constraints defined by a set of DFC rules. In particular, we propose hybridizing constraint programming and standard BDD compilation techniques in order to scale the construction of a BDD for larger size catalogues. Our approach outperforms the standard BDD techniques by reducing the memory requirements by as much as five orders-of-magnitude and compiles the catalogues for which the standard techniques ran out of memory.

1 Introduction

Information and communication services, from news feeds to internet telephony, are playing a more and more 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 [Int93, Int97]. A feature is an increment of functionality which, if activated, modifies the basic service and generates or modifies the behaviour of other features. An optimal relaxation is a most preferred subset of the catalogue of features. In this paper, we formalise some notions that are relevant for the feature subscription problem in Section 3. The necessary background for binary decision diagrams is provided in Section 4. In Section 5 we describe our BDD-based solution approach to computing optimal subscriptions given that a BDD approach easily constructs a BDD for catalogues consisting of 25 features without any memory problems, while standard techniques run out of memory.

In the remainder of this paper we first describe the Distributed Feature Composition architecture in Section 2. We formalise some notions that are relevant for the feature subscription problem in Section 3. The necessary background for binary decision diagrams is provided in Section 4. In Section 5 we describe our BDD-based solution approach to computing optimal subscriptions given that a BDD is already compiled. In Section 6 we describe four techniques for

![Figure 1. An example of an undesirable feature interaction.](image)
compiling catalogues into BDDs. We report experimental results in Section 7 comparing each technique, and finally, we conclude.

2 Distributed Feature Composition

This section provides an overview of the DFC architecture, its routing method and the terminology relevant to the understanding of the feature subscription problem [Les07].

In DFC each feature is implemented by one or more modules called feature box types (FBT). 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 endpoints is set up by chaining feature boxes, i.e., instances of FBTs. 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 features 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 [ZGS04]. 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.

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 appearing in zones or their ordering. They simply fetch this information from the feature subscriptions that are pre-configured 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. 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—TCS. Configuring a subscription involves selecting, parameterising and sequencing features in each region consistently with the catalogue constraints and other integrity rules [JZ03]. 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 Configuring Feature Subscriptions

A catalogue is a pair \(\langle F, P \rangle\), where \(F\) is a set of features and \(P\) is a set of precedence constraints on \(F\). Let \(f_1\) and \(f_2\) be features, we write a precedence constraint of \(f_1\) before \(f_2\) as \((f_1, f_2)\), or alternatively, \(f_1 \rightarrow f_2\). Note that an exclusion constraint between \(f_1\) and \(f_2\) can be encoded as the pair of precedence constraints \((f_1, f_2)\) and \((f_2, f_1)\).

The transpose of a catalogue \(\langle F, P \rangle\) is \(\langle F, P^T \rangle\) such that \(\forall(f_1, f_2) \in P \Rightarrow (f_2, f_1) \in P^T\). 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 compose the source catalogue \(\langle F_s, P_s \rangle\) and the target catalogue \(\langle F_t, P_t \rangle\) into a single catalogue \(\langle F_s, P_s \cup \langle f_1 \rightarrow f_2 \rangle, P_t \cup \langle f_1 \rightarrow f_2 \rangle^T \rangle\). A catalogue \(\langle F_s, P_s \rangle\) can also be seen as a directed graph by mapping the features in \(F_s\) to vertices and the precedence constraints in \(P_s\) to the edges. A maximal set (with respect to inclusion) of features of the catalogue that one can subscribe to is a set \(F'\) such that (i) \(F' \subseteq F_s\), (ii) the directed graph \(\langle F', P_s \cup \langle f_1 \rightarrow f_2 \rangle \rangle\) is acyclic and (iii) \(\forall f \in F_s \setminus F'\), the directed graph \((F', \cup \langle f \rangle, \langle f_1 \rightarrow f_2 \rangle)^T\) is cyclic.

A feature subscription \(\langle F_s, P_s \rangle\) of catalogue \(\langle F, P \rangle\) is a tuple \(\langle F, C, W_F \rangle\), where \(F \subseteq F_s\), \(C\) is the projection of \(P_s\) on \(F_s\), i.e., \(P_{C \downarrow F}\), \(C = \{f_1 \rightarrow f_2\} \subseteq F\) and \(W_F : F \rightarrow N\) is a function that assigns weights to features. The value of \(S\) is defined by \(Value(S) = \sum_{f \in F} W_F(f)\). Note that a weight associated with a feature signifies its importance for the user.

A feature subscription \(\langle F, C, W_F \rangle\) is defined to be consistent if and only if the directed graph \(\langle F, C\rangle\) is acyclic. Determining whether a feature subscription \(\langle F, C, W_F \rangle\) is consistent or not can be checked in \(O(|F| + |C|)\) time using Topological Sort [CLR90].

A relaxation of a subscription \(\langle F, C, W_F \rangle\) is a subscription \(\langle F', C', W'_F \rangle\) such that \(F' \subseteq F\), \(C' = P_{C \downarrow F}\), and \(W_{F'} : F' \rightarrow N\). Let \(S = \langle F, C, W_F \rangle\) be an inconsistent feature subscription. A relaxation \(S' = \langle F', C', W'_F \rangle\) of \(S\) is maximal if \(S'\) is consistent and each relaxation \(S'' = \langle F'', C'', W''_F \rangle\) where \(S'' \neq S'\), \(F'' \supseteq F'\), \(C'' = P_{C \downarrow F'}\), and \(W''_F = W_{F'} \downarrow F'\) is inconsistent. We call \(F'\) a maximal set of features of the subscription \(S\). A single maximal relaxation can be found by traversing the features in \(F\) and checking at each time that the current feature/precedence can be added. If the feature can be added then it is considered part of the relaxation and the next check is performed on this basis. If not, the feature is simply discarded. As \(F\) checks are performed, the overall complexity is \(O((|F| + |C|))\). However, finding the set of all maximal relaxations is exponential [BS05].

Let \(R_S\) be the set of all maximal relaxations of subscription \(S\). We say that \(S' \in R_S\) is an optimal relaxation of \(S\) if it has maximum value amongst all maximal relaxations, i.e., if and only if there does
not exist $S'' \in R_S$ such that $\text{Value}(S'') > \text{Value}(S')$. Finding an optimal relaxation is NP-Hard [LMO+07]. We shall focus on this problem in this paper.

### 4 Binary Decision Diagrams

A binary decision diagram (BDD) [Bry86] is a rooted directed acyclic graph, with vertices $V$ and edges $E \subseteq V \times V$, that encodes a constraint set over some set of linearly ordered Boolean variables. It has two terminal nodes labeled with $T_0$ and $T_1$. All other nodes $u \in V \setminus \{T_0, T_1\}$ are labeled with a variable $\text{var}(u) \in \{1, \ldots, n\}$ and have exactly two outgoing edges: a low edge ending in node $\text{low}(u)$ and a high edge ending in $\text{high}(u)$. The BDD is ordered if variable labels along all the paths from the root to either $T_0$ or $T_1$ respect the ordering of the variables. Given an assignment to the variables, whether the constraint is satisfied is determined by following a path starting at the root node and recursively following the high edge, if the associated variable is assigned 1, and the low edge, if the associated variable is assigned 0. The constraint set is satisfied if we reach terminal node $T_1$; otherwise it is violated.

Even though BDDs are worst-case exponential in the size of an input constraint model, they are compact for many important classes of constraints. This is because we can use reduced ordered BDDs, where isomorphic nodes are merged and redundant nodes are eliminated. Two distinct nodes $u$ and $u'$ are isomorphic if they are labeled with the same variable $\text{var}(u) = \text{var}(u')$ and have the same child nodes: $\text{low}(u) = \text{low}(u')$ and $\text{high}(u) = \text{high}(u')$. Merging corresponds to deleting one of the nodes (say $u'$) and redirecting all incoming edges of $u'$ to $u$. A node $u$ is redundant if both child nodes are the same: $\text{low}(u) = \text{high}(u)$. Eliminating $u$ corresponds to deleting it, and redirecting all incoming edges to the child node. Eliminating $u$ introduces a long-edge that skips the variable $\text{var}(u)$ indicating that all assignments to skipped variables are allowed. A reduced ordered BDD for the conjunction of two Boolean constraints $\{x_1 \neq x_2, x_3 \neq x_4\}$ is shown in Figure 3. Notice how an assignment $x_1 = 1, x_2 = 1$ leads to a long-edge ending in $T_0$ and skipping $x_3$ and $x_4$. This means that no matter what is assigned to $x_3$ and $x_4$, when $x_1 = 1$ and $x_2 = 1$, the constraint is violated.

![Figure 3. Reduced OBDD for $x_1 \neq x_2 \land x_3 \neq x_4$](image)

**Optimisation Using BDDs.** An additive objective function $\sum c_j(x_j)$ can be minimised subject to a constraint set by finding a shortest path from the root to $T_1$ in the corresponding BDD. If node $u$ and $u'$ have labels $x_k$ and $x_l$, respectively, then an edge from $u$ to $u'$ has length:

$$c^*[u, u'] = c_k(v) + \sum_{j=k+1}^{l-1} \min\{c_j(1), c_j(0)\}$$

where $v = 1$ if $u' = \text{high}(u)$ and $v = 0$ if $u' = \text{low}(u)$. If this edge is part of a shortest path, it induces assignments to corresponding variables $\{x_k, x_{k+1}, \ldots, x_{l-1}\}$ such that $x_k = v$, and for all skipped variables $x_j = v_j$, where $v_j = 1$ if $c_j(1) < c_j(0)$, $v_j = 0$ if $c_j(1) > c_j(0)$ and $v_j \in \{0, 1\}$ otherwise.

### 5 A BDD-Based Solution Approach

Given a catalogue $\langle F_1, P_c \rangle$ with $n$ features and $m$ precedence relationships, we generate a Constraint Satisfaction Problem where for each feature $f_i \in F_1$, we introduce two variables: a Boolean variable $s_i$ indicating whether a feature $f_i$ is selected, and an integer variable $p_i \in \{1, \ldots, n\}$ indicating the position of a feature in a sequence of selected features. For every catalogue precedence $\langle f_i, f_j \rangle \in P_c$, we introduce the constraint $s_i \land s_j \Rightarrow p_i < p_j$.

Our solution approach is based on dividing the computational effort between two phases. In the offline phase (prior to user interaction), we compile the catalogue into a BDD $B$ (see Section 6) that represents the conjunction of all the precedence constraints:

$$B = \bigwedge_{\langle f_i, f_j \rangle \in P_c} (s_i \land s_j) \Rightarrow p_i < p_j. \quad (1)$$

In the online phase, when the user-selected features $F \subseteq F_1$ are known, we compute optimal relaxations using efficient shortest path algorithms, for example, Dijkstra’s shortest-path algorithm [CLR90].

An additive cost function $\sum_{i=1}^n c_i(s_i)$ is defined based on user-selected features $F$, such that $c_i(1) = 0, c_i(0) = W_F(f_i)$ if a feature $f_i \in F$ was selected by a user, and $c_i(1) = 0, c_i(0) = 0$ otherwise. To all Boolean variables encoding finite-domain position variables $p_j$ we assign cost 0. A shortest path with respect to this additive cost function induces an assignment to variables $s_i$ that represents a subset of features $F' \subseteq F$. If the induced set of assignments $(s_1 = v_1, \ldots, s_n = v_n)$ involves an assignment $s_j = 1$ for some feature $f_j$ not selected by a user ($f_j \notin F$), we can truncate this assignment, i.e., set $s_j = 0$, since this yields a compatible assignment of the same cost. After truncating all such selections of non-user features, we are left with an assignment representing an optimal subset of user-selected features $F'$. Once we have $F'$, we can efficiently order it in a way that respects all the relevant precedence constraints by using topological sort.

**Other BDD Applications.** Note that once the BDD is computed and an additive cost function is given, we can efficiently implement a range of functionalities supporting a user choosing a desired feature subscription. For example, we can efficiently compute the set of $k$ best relaxations by using the algorithm presented in [NBW06]; we can assist a user to interactively configure relaxations of cost at most $k$ by using an approach from [HA06]; we can perform postoptimality analysis by analysing which relaxations become available as the maximal cost increases [HH06] and we can find a most similar/diverse relaxation using approach from [HOW07].

The main issue, regardless of which user functionality we choose to implement, is whether we can compile the corresponding BDD $B$ and whether the resulting size allows for efficient online processing.
6 Compiling Feature Subscription BDDs

Generating a BDD that represents all consistent subscriptions of a catalogue \( \{F_c, P_c\} \) by using a standard approach to BDD compilation, would involve first constructing a BDD \( B_{ij} \) for each precedence constraint \( \langle f_i, f_j \rangle \in P_c \), and then conjoining all the resulting BDDs using the standard BDD conjunction operator: \( B = \bigwedge_{\langle f_i, f_j \rangle \in P_c} B_{ij} \). Our attempt to use this compilation approach did not scale beyond catalogues involving 15 features as the resulting BDD had excessive memory requirements, measuring in millions of nodes (more details are presented in Section 7).

In the remainder of the section we therefore describe the techniques we used to scale up the BDD compilation. We first enhanced standard compilation through variable elimination of position variables \( p_i \), which reduced the size of the final BDD \( B \) from millions to thousands of nodes. This alone did not allow us to scale beyond catalogues involving more than 20 features since the size of the largest BDD resulting from intermediate conjunctions (memory peak) was still too large. We therefore used a BDD compilation based on constraint programming search which overcame this problem and helped us scale to our designated goal of handling 25 features.

6.1 Variable Elimination Approach

Since our additive cost function does not depend on position variables, it is sufficient to execute a shortest path algorithm over a BDD:

\[
B_s \equiv \exists p_1, \ldots, p_n \left( \bigwedge_{\langle f_i, f_j \rangle \in P_c} (s_i \land s_j \Rightarrow p_i < p_j) \right) \tag{2}
\]

that represents a projection of \( B \) onto the \( s_i \) variables. In order to handle models where generating the original BDD \( B \) is not possible, we cannot compile \( B_s \) by first computing \( B \) and then eliminating the \( p_i \) variables. Instead, we have to eliminate the \( p_i \) variables during the conjunction of BDDs \( B_{ij} \) as soon as it is detected that they do not occur in the remaining constraints that are to be conjoined. We used a particular conjunction heuristic, where for each feature \( f_i \) we conjoined all the BDDs involving \( s_i \) and \( p_i \), and afterwards eliminated \( p_i \) through the standard BDD operation of existential quantification [MT98]. Figure 4 shows the algorithm implementing this conjunction heuristic, and Figure 5 compares variable elimination against the standard approach by showing the size of BDDs in intermediate conjunction steps.

6.2 Constraint Programming Approach

Even though the final BDDs generated using the variable elimination approach were remarkably small, the BDDs resulting from intermediate conjunction steps were too big to allow scaling beyond catalogues involving more than 20 features. We observed, however, that the set of all maximal sets of features, \( F_{\max} \), of the underlying catalogue \( \{F_c, P_c\} \) was no more than a few thousand, i.e. significantly smaller than in the worst case \(|F_{\max}| \ll 2^{\mid F_c \mid} \). We also observed that in order to construct a BDD \( B_s \) that represents all the consistent subsets of \( F_c \) (not necessarily maximal), it is sufficient to add the powerset of \( F_{\max} \) for all \( F_{\max} \in F_{\max} \). Each powerset is represented by \( \bigwedge_{f_i \in F_{\max} - F_{\max}} \neg s_i \), where \( F_c - F_{\max} \) denotes those features that are not part of the maximal set of features of the catalogue.

\[
B_{s}^{\max} \equiv \bigvee_{F_{\max} \in F_{\max}} \left( \bigwedge_{f_i \in F_{\max}} s_i \land \bigwedge_{f_j \in F_{\max} - F_{\max}} \neg s_j \right) \tag{4}
\]

Notice that any maximal set \( F' \) of the feature subscription \( \langle F, C, W_F \rangle \) is a subset of at least one of the maximal set of features \( F_{\max} \in F_{\max} \). Therefore, it is still possible to find an optimal relaxation \( F' \subseteq F \) of user-selected features \( F \subseteq F_c \) by using the same approach as in Section 5: we define the objective function \( \sum_{i=1}^{n} c_i(s_i) \), and an optimal relaxation is obtained using the shortest path by truncating non-user features \( f_j \notin F \).
7 Experimental Evaluation

We compared the performance of different compilation approaches discussed in the previous sections: a standard approach that generates the BDD $B$ from Equation (1), a variable elimination approach that generates projection $B_s$ from Equation (2), a CP approach generating $B_s$ based on Equation (3), and a CP-max approach generating $\overrightarrow{B_{max}}$ based on Equation (4).

We generated and experimented with a variety of random catalogues $(n_c, d_c)$ where $n_c$ is the number of features, $d_c \in [0, 1]$ is the density of the precedence constraints, i.e. it denotes the percentage of the maximum number of constraints that are selected. A random catalogue is generated by selecting $n_c = \lfloor d_c \times n_c \times (n_c - 1) / 2 \rfloor$ pairs of features. For each selected pair $(f_i, f_j)$ we randomly decide with equal probability whether $f_i \prec f_j$ or $f_j \prec f_i$. In our experiments we let $n_c \in \{5, 10, 15, 20, 25\}$ and $d_c \in \{0.1, 0.2, 0.4, 0.6, 0.7\}$. For each combination of $n_c$ and $d_c$ we generated five random catalogues.

For each approach we tried several variable ordering heuristics and selected those with the best performance. For the first two approaches we put variables appearing in the larger number of constraints higher in the ordering. For the CP and CP-max approaches a variable corresponding to a feature that is included in fewer maximal relaxations was put higher in the ordering.

Table 1 summarises the results for a subset of instances with density $d_c = 0.4$. We can see that the CP and CP-max approaches dramatically outperform the standard and variable elimination compilation approach as they reduce the memory peak by several orders of magnitude. Furthermore, the time required by these approaches was not excessive. In our experience it never exceeded one hour of computation time (on a machine with a 1.8 GHz processor and 768 MB of RAM). This was comparable to the time required by the first two approaches.

8 Conclusions

In this paper we presented an approach for solving a feature subscription problem by first compiling all consistent sets of catalogue features into a BDD in the offline phase, which then allows for an efficient computation of the optimal consistent subset of user selected features in the online phase. We addressed the key computational issue of compiling corresponding BDDs by investigating four alternative compilation techniques. In particular, we suggested two compilation approaches based on constraint programming which reduce the memory peak by several orders-of-magnitude. As a result, we easily reached our target of handling catalogues of 25 features.

9 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.

REFERENCES


