Filtered Cartesian Flattening: An Approximation Technique for Optimally Selecting Features while Adhering to Resource Constraints

J. White and D. C. Schmidt
Vanderbilt University, EECS Department
Nashville, TN, USA
Email:{jules, schmidt}@dre.vanderbilt.edu

June 22, 2008

Abstract

Software Product-lines (SPLs) use modular software components that can be reconfigured into different variants for different requirements sets. Feature modeling is a common method for capturing the configuration rules for an SPL architecture. A key challenge for developers is determining how to optimally select a set of features while simultaneously honoring resource constraints. For example, optimally selecting a set of features that fit the development budget is an NP problem. Although resource consumption problems have been extensively studied in the Artificial Intelligence and Distributed Computing communities, the algorithms that these communities have developed, such as Multiple-choice Knapsack Problems (MMKP) approximation algorithms, have not been extensively applied to feature selection subject to resource constraints. The paper provides the following contributions to the study of automated feature selection for SPL variants: (1) we present a polynomial time approximation technique called *Filtered Cartesian Flattening* (FCF) for deriving approximately optimal solutions to feature selection problems with resource constraints by transforming the problems into equivalent MMKP problems and using MMKP approximation algorithms, (2) we show that FCF can operate on large feature models that would not be possible with existing algorithmic approaches, and (3) we present empirical results from initial experiments performed using FCF. Our results show that FCF is 93%+ optimal on feature models with 5,000 features.

1 Introduction

Software Product-Lines (SPLs) [5] define reusable software architectures where applications are assembled from modular components. Each time a new version of an SPL application (called a *variant*) is created, the rules governing the composition of the SPLs modular components must be strictly adhered to. *Feature Modeling* [7] is a common modeling technique used to capture an SPL's configuration rules.

A feature model is a tree-like structure where each node in the tree represents a variation or increment in application functionality. Parent-child relationships in the tree indicate the refinement of application functionality. For example, in a feature model for an Magnetic Resonance Imaging system a parent feature would be *Magnet Strength* and the children would be *1.5 Tesla Magnet* or *3 Tesla Magnet*.

Each SPL variant is described as a selection or list of features that are present in the variant. The construction of a variant is bounded by adding constraints on how features are selected. If a feature is

selected, the feature's parent must also be selected. Moreover, features can be required, optional, or involved in XOR/Cardinality relationships with other features to model a wide range of configuration rules.

A key challenge when selecting features for an SPL variant is determining how to select an optimal set of features while simultaneously adhering to one or more resource constraints. For example, in a medical imaging system, if each hardware feature has a cost and value associated with it, selecting a set of features that maximizes the resulting variant's value but also fits within a customer's procurement budget is hard. Proofs that this problem is an NP problem can be built by showing that any instance of the NP Multi-dimensional Multiple-choice Knapsack Problem [1] (MMKP) can be reduced to a feature selection problem with resource constraints, as described in Section 3.1.

Existing techniques for deriving solutions to feature selection problems [4, 8, 3, 13] have utilized exact methods, such as integer programming [12] or SAT solvers [9]. Although these approaches provide guaranteed optimal answers they have exponential algorithmic complexity. As a result, these algorithms do not scale well to large feature selection problems with resource constraints.

Resource constraints have been extensively studied in the Artificial Intelligence and distributed computing communities. Although there are a number of good approximation algorithms that have been developed for problems involving resource constraints [1], these algorithms have so far not been applied to feature modeling. One key obstacle to applying existing MMKP algorithms to feature selection subject to resource constraints is that the hierarchical structure of a feature model does not fit into the MMKP problem paradigm.

To address the lack of approximation algorithms for selecting featues subject to resource constriants, we have created a polynomial-time approximation technique, called *Filtered Cartesian Flattening* (FCF), for selecting approximately optimal feature sets while adhering to multi-dimensional resource constraints. This paper provides several contributions to the automated construction of SPL variants. First, we present the polynomial time FCF technique for selecting approximately optimal feature sets while respecting resource constraints. We then show how FCF can be combined with different MMKP approximation algorithms to provide different levels of optimality and time complexity. Finally, we present initial empirical data showing that FCF provides roughly 93%+ optimality on feature models with 5,000 features.

The remainder of this paper is organized as follows: Section 2 describes the challenge of optimally selecting a set of features subject to a set of resource constraints; Section 3 presents our FCF approximation technique for selecting nearly-optimal feature sets subject to resource constraints; Section 4 presents empirical results showing that our algorithm averages 93%+ optimality on feature models of 5,000 features; Section 5 compares our work to related research; and Section 6 presents concluding remarks and lessons learned.

2 Challenge: Optimally Selecting Features Subject to Resource Constraints

A common goal in selecting features is to maximize the perceived value or quality of the variant produced. In the context of medical imaging systems, for instance, a key goal is to maximize the accuracy of the images produced. The problem is that there are usually additional constraints on the feature selection that are not captured in the feature model and that make the optimization process hard.

For example, features often have costs associated with them, such as the cost of different strength magnets for a Magnetic Resonance Imaging (MRI) machine. A producer of an MRI machine cannot force its customers to buy an MRI variant that exceeds the customer's budget. When a customer requests an MRI machine, therefore, the producer must select a set of features for which the sum is less than the customer's budget, which also maximizes the accuracy of the machine.

We call this problem *optimal feature selection subject to resource constraints*. As shown in Section 3.1,

any instance of another NP problem (the MMKP problem) can be reduced to an instance of this problem. Optimal feature selection subject to resource constraints is thus also in NP.

Large-scale industrial feature models, such as those for the automation software in continuous casting steel plants, can contain on the order of 30,000 features [11]. Existing techniques [4, 8, 3, 13] that use exact but exponential algorithms do not scale well to these large problem sizes. Other existing NP approximation algorithms, such as those for MMKP problems cannot be directly applied to optimal feature selection subject to resource constraints. Since exact algorithms do not scale well for these problems and existing approximation algorithms cannot be applied, it is hard to automate feature models for large-scale applications.

3 Filtered Cartesian Flattening

This section presents the *Filtered Cartesian Flattening* (FCF) approximation technique for optimal feature selection subject to resource constraints. FCF transforms an optimal feature selection problem with resource constraints into an equivalent MMKP problem and then solves it using an MMKP approximation algorithm.

3.1 MMKP Problems

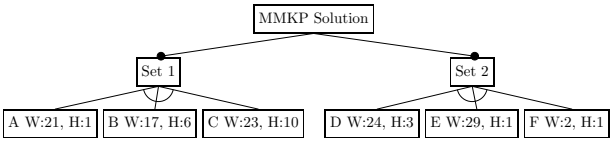

Figure 1: A Feature Model of an MMKP Problem Instance

A Knapsack problem [6] is an NP problem where there is a knapsack of fixed size and the goal is to place as many items as possible from a set into the knapsack. An MMKP problem is a variation on the Knapsack problem where the items are divided into X disjoint sets and at most one item from each set may be placed in the knapsack. Values are typically assigned to each item and the goal is to maximize the value of the items placed in the knapsack.

MMKP problems can be reduced to optimal feature selection problems with resource constraints. First, a single root feature denoting the solution is created. Next, for each set, a required child feature representing the set is added to the feature model as a child of the root. For each set, the corresponding feature in the feature model is populated with a child XOR group[1] containing the items in the set. The available resources for the feature selection problem are defined as the size of the knapsack. The resources consumed by each feature are assigned to the length, width, and height of the original MMKP item. An example feature model of an MMKP problem is shown in Figure 1.

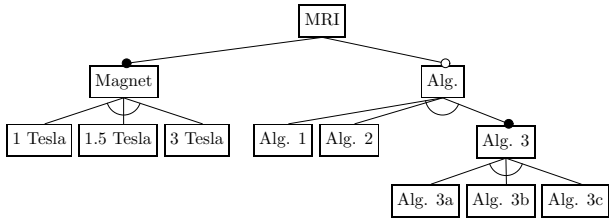

Figure 2: An Example MRI Feature Model

FCF works by performing the reverse process– transforming feature selection problems with resource constraints into MMKP problems. The steps in the FCF technique are designed to flatten the hierarchically structured feature model into a number of independent MMKP sets to form an MMKP problem. Figure 2 shows an example feature model of an MRI machine and Figure 3 illustrates the equivalent MMKP problem. Each item in these sets represents a potential *valid* partial feature selection from the feature model. There are an exponential number of potential feature selections and thus some of the potential configurations must be filtered out to limit the time complexity of the technique. FCF performs this filtering in the third step of the algorithm, described in Section 3.4.

Since each MMKP set that is produced by FCF

[1] An XOR group is a set of features of which exactly one of the features may be selected at a time.

```
MMKP Set 1:
<MRI, Magnet, 1 Tesla>,
<MRI, Magnet, 1.5 Tesla>,
<MRI, Magnet, 3 Tesla>

MMKP Set 2:
<Alg., Alg 1>,
<Alg., Alg 2>,
<Alg., Alg 3, Alg. 3a>,
<Alg., Alg 3, Alg. 3b>,
<Alg., Alg 3, Alg. 3c>

Note:
(items/feature selections denoted with '< >')
```

Figure 3: MMKP Sets for MRI Feature Model

contains items representing valid partial feature selections, the technique must ensure that choosing items from any of the X MMKP sets produces a feature selection that is both valid and complete. The FCF algorithm accomplishes this task in its first step (see Section 3.2) by creating one MMKP set for the subtree of features directly required by the root feature. The remaining MMKP sets are produced from the subtrees of features that are connected to the root through an optional feature. The technique does not currently support cross-tree constraints, although this is part of our future research.

3.2 Step 1: Cutting the Feature Model Graph

The first step in FCF is to subdivide the feature model into a number of independent subtrees. The goal is to choose the subtrees so that the selection of features in one subtree does not affect the selection of features in other subtrees. One MMKP set will later be produced for each subtree.

We define features that are optional or involved in an XOR group or a cardinality group as *choice points*. A cardinality group is a group of features that when selected must adhere to a cardinality expression (*e*.g., select $2 \ldots 3$ of the features X, Y, and Z). An XOR group is a special case of a cardinality group where exactly one feature from the group must be selected (*e*.g., it has cardinality $1 \ldots 1$). Starting from the root, a depth-first search is performed to find each optional feature with no ancestors that are choice points. At each optional feature with no choice point ancestors, a cut is performed to produce a new independent subtree, as shown in Figure 4.

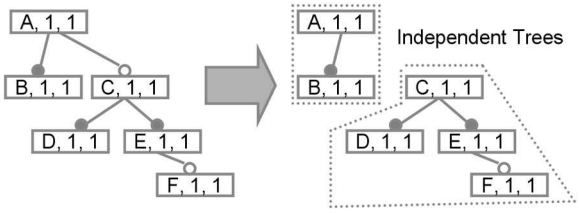

Figure 4: Cutting to Create Independent subtrees

3.3 Step 2: Converting All Feature Constraints to XOR Groups

Each MMKP set forms an XOR group of elements. Since MMKP does not support any other relationship operators, such as cardinality, nor does it support hierarchy, we must flatten each of the subtrees and convert all of their relationship types into XOR. This conversion allows the conversion of the feature model's independent subtrees into a series of MMKP sets.

Cardinality groups are converted to XOR groups by replacing the cardinality group with an XOR group containing all possible combinations of the cardinality group's elements that satisfy the cardinality expression. Each new item produced from the Cartesian product has the combined resource consumption and value of its constituent features. Since this conversion could create an exponential number of elements, we bound the maximum number of elements that are generated to a constant number K. Rather than requiring exponential time, therefore, the conversion can be performed in constant time.

The conversion of cardinality groups is one of the first steps where approximation occurs. We define a filtering operation that chooses which K elements from the possible combinations of the cardinality

group's elements to add to the XOR group. All other elements are discarded.

Any number of potential filtering options can be used. Our experiments evaluated a number of filtering strategies, such as choosing (1) the K highest valued items, (2) a random group of K items, and (3) a group of K items evenly distributed across the items' range of weights. We define a feature's weight or size as the amount of each resource consumed by the feature. We found that selecting the K items with the best ratio of $\frac{Value}{\sqrt{\sum rc_i^2}}$, where rc_i is the amount of the i_{th} resource consumed by the item, provided the best results. This sorting critera has been used successfully by a number of other MMKP algorithms [2]. An example conversion with $K = 3$ and random selection of items is shown in Figure 5.

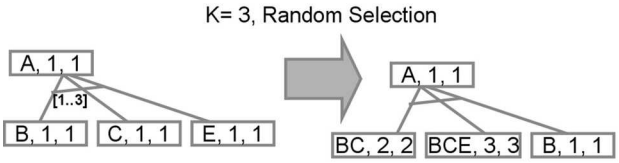

Figure 5: Converting a cardinality group to an XOR Group with K=3 and Random Selection

Individual features with cardinality expressions attached to them are converted to XOR groups using the same method. The feature is considered as a cardinality group containing M copies of the feature, where M is the upper bound on the cardinality expression (*e.g.* $[L..M]$ or $[M]$). The conversion then proceeds identically to cardinality groups.

Optional features are converted to XOR groups by replacing the optional feature O with a new required feature O'. O' in turn, has two child features, O and $\emptyset$ forming an XOR group. O' and $\emptyset$ have zero weight and value. An example conversion is shown in Figure 6.

3.4 Step 3: Flattening with Filtered Cartesian Products

For each independent subtree of features that now only have XOR and required relationships, an

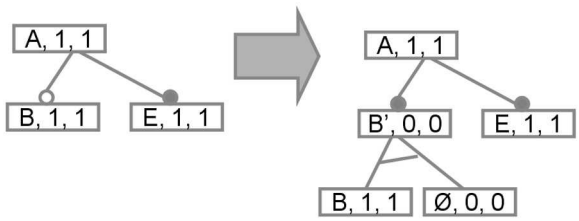

Figure 6: Converting an Optional Feature into an XOR Group

MMKP set needs to be produced. Each MMKP set needs to consist of a single top-level XOR group. To create a single top-level XOR group, we perform a series of recursive flattening steps using filtered Cartesian products to produce an MMKP set containing complete and valid partial feature selections for the subtree.

For each feature with a series of required children, a set of items is produced from a filtered Cartesian product of the sets produced by recursively running the algorithm on its children. For each feature with an XOR group child, a set of items is produced consisting of the Cartesian product of the feature and the union of the sets produced by recursively applying the algorithm to the features in its XOR subgroup. This process is repeated until a single set of items remains. A visualization of this process is shown in Figure 7.

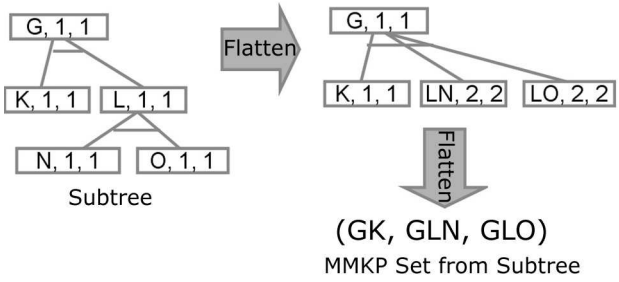

Figure 7: Flattening an XOR Group

Once each independent subtree has been converted into an MMKP set, we must mark those sets which represent optional configuration choices. For each set that does not include the root feature, we add an item

$\emptyset$ with zero weight and zero value indicating that no features in the set are chosen. This standard MMKP method handles situations where choosing an item from some sets is optional. Since the root feature must always be chosen, the $\emptyset$ item is not added to its set.

3.5 Step 4: MMKP Approximation

The first three steps produce an MMKP problem where each set contains items representing potential partial configurations of different parts of the feature model. One set contains partial configurations for the mandatory portions of the feature model connected to the root. The remaining sets contain partial configurations of the optional subtrees of the feature model.

The final step in deriving an optimal architectural feature selection is to run an existing MMKP approximation algorithm to select a group of partial configurations to form the feature selection. For our implementation of FCF, we used a simple modification of the M-HEU algorithm [2] that puts an upper limit on the number of upgrades and downgrades that can be performed. Since FCF produces an MMKP problem, we can use any other MMKP approximation algorithm, such as C-HEU [10]) which uses convex hulls to search the solution space. The solution optimality and solving time will vary depending on the algorithm chosen.

3.6 Algorithmic Complexity

The complexity of FCF's constituent steps can be analyzed as follows:

- The first step in the FCF algorithm, cutting the tree, requires $O(n)$ time to traverse the tree and find where to make the top-level optional features.
- The second step of the algorithm requires $O(Kn * S)$ steps, where S is the time required to perform the filtering operation. Simple filtering operations, such as random selection, do not add any algorithmic complexity. In these cases, at most n sets of K items must be created to convert the tree to XOR groups, yielding $O(Kn)$. In our experiments, we selected the K items with the best value to resource consumption ratio. With this strategy, the sets must be sorted, yielding $O(Kn^2 \log n)$.
- The third step in the algorithm requires flattening at most n groups using filtered Cartesian products, which yields a total time of $O(Kn*S)$.
- The solving step incurs the algorithmic complexity of the MMKP approximation algorithm chosen.

This analysis yields a total general algorithmic complexity for FCF of $(n + (Kn * S) + (Kn * S) + MMKP+n) = O(Kn*S+MMKP+n)$. As long as a polynomial time filtering operation is applied, FCF will have an overall polynomial time complexity. For large-scale problems, this polynomial time complexity is significantly better than an exponential running time.

4 Results

To evaluate our FCF approximation technique, we generated random feature models and then created random feature selection problems with resource constraints from the feature models. For example, we would first generate a feature model and then assign each feature an amount of RAM, CPU, etc. that it consumed. Each feature was also associated with a value. We randomly generated a series of available resources values and asked the FCF algorithm to derive the feature selection that maximized the sum of the value attributes while not exceeding the randomly generated available resources. Finally, we compared the FCF answer to the optimum answer.

We performed the experiments using 8 dual processor 2.4ghz Intel Xenon nodes with 2 GB RAM. Each node was loaded with Fedora Core 4. We launched 2 threads on each machine, enabling us to generate and solve 16 optimal feature selection with resource constraints problems in parallel.

The results from solving 18,500 different feature selection problems, each with a feature model of 5,000 features and 2 resource types (RAM and CPU) is shown in Figure 8. We set the max set size, K, in the filtering steps to 2,500. The X axis shows the percentage of optimality. The Y axis shows the number of problem instances or samples that were solved with the given optimality. The overall average optimality was 93%.

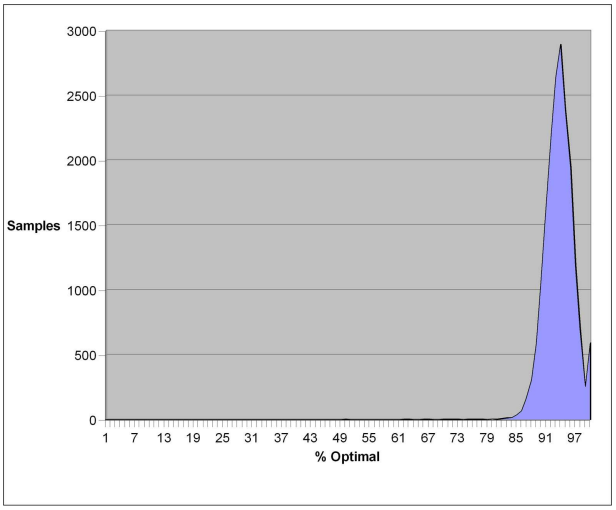

Figure 8: FCF Optimality on 18,500 Feature Models with 5,000 Features and 2 Resources

5 Related Work

Benavides et al. [4] present a technique for using Constraint Satisfaction Problems (CSPs) to model and solve feature selection problems. This technique can be modified to solve feature selection problems subject to resource constraints [13]. Their technique works well for small-scale problems, where an approximation technique is not needed. For larger-scale problems, however, their technique is too computationally demanding. In contrast, FCF performs well on these large-scale problems.

Other approaches to automated feature selection rely on propositional logic, such as those presented by Mannion [8] and Batory [3]. These techniques were not designed to handle integer resource constraints and thus are not equipped to handle optimal feature selection problems subject to resource constraints. Moreover, these techniques rely on SAT solvers that use exponential algorithms. FCF is a polynomial-time algorithm that can handle integer resource constraints and thus can perform optimal feature selection subject to resource constraints on large-scale problems.

6 Conclusion

Approximation algorithms are needed to optimally select features for large-scale feature models subject to resource constraints. Although there are numerous approximation algorithms for other NP problems, they do not directly support optimal feature selection subject to resource constraints. The closest possible class of approximation algorithms that could be applied are MMKP approximation algorithms but these algorithms are not designed to handle the hierarchical structure and non-XOR constraints in a feature model. This lack of approximation algorithms limits the scale of model on which automated feature selection subject to resource constraints can be performed.

This paper shows how the *Filtered Cartesian Flattening* (FCF) approximation technique can be applied to optimally select features subject to resource constraints. FCF creates a series of filtered Cartesian products from a feature model to produce an equivalent MMKP problem. After an equivalent MMKP problem is obtained, existing MMKP approximation algorithms can be used to solve for a feature selection. The empirical results in Section 4 show how FCF can achieve an average of at least 93% optimality for large feature models. The results can be improved by using more exact MMKP approximation algorithms, such as M-HEU [2].

From our experience with FCF, we have learned the following lessons:

- For small-scale feature models (*e.g.*, with < 100 features) MMKP approximation algorithms do

not provide optimal results. For these smaller problems, exact techniques, such as those designed by Benavides et al. [4], should be used.

- For large-scale feature models (*e.g.*, with > 5,000 features) exact techniques typically require days, months, or more, to solve optimal feature selection problems subject to resource constraints. In contrast, FCF typically requires between 1-5 seconds for a 5,000 feature problem.

- Once a feature model has been subdivided into a number of independent subtrees, these subtrees can be distributed across independent processors to flatten and solve in parallel. The FCF technique is thus highly amenable to multi-core processors and parallel computing.

An implementation of FCF is included with the open-source GEMS Model Intelligence project and is available from `eclipse.org/gmt/gems`.

References

[1] M. Akbar, E. Manning, G. Shoja, and S. Khan. Heuristic Solutions for the Multiple-Choice Multi-Dimension Knapsack Problem. *International Conference on Computational Science*, pages 659–668, May 2001.

[2] M. Akbar, E. Manning, G. Shoja, and S. Khan. Heuristic Solutions for the Multiple-Choice Multi-Dimension Knapsack Problem. pages 659–668. Springer, May 2001.

[3] D. Batory. Feature Models, Grammars, and Prepositional Formulas. *Software Product Lines: 9th International Conference, SPLC 2005, Rennes, France, September 26-29, 2005: Proceedings*, 2005.

[4] D. Benavides, P. Trinidad, and A. Ruiz-Cortés. Automated Reasoning on Feature Models. *17th Conference on Advanced Information Systems Engineering (CAiSE05, Proceedings), LNCS*, 3520:491–503, 2005.

[5] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, 2002.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT, 1990.

[7] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. FORM: A Feature-Oriented Reuse Method with Domain-specific Reference Architectures. *Annals of Software Engineering*, 5(0):143–168, January 1998.

[8] M. Mannion. Using first-order logic for product line model validation. *Proceedings of the Second International Conference on Software Product Lines*, 2379:176–187, 2002.

[9] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient SAT solver. *Proceedings of the 38th conference on Design automation*, pages 530–535, 2001.

[10] M. Mostofa Akbar, M. Sohel Rahman, M. Kaykobad, E. Manning, and G. Shoja. Solving the Multidimensional Multiple-choice Knapsack Problem by constructing convex hulls. *Computers and Operations Research*, 33(5):1259–1273, 2006.

[11] R. Rabiser, P. Grunbacher, and D. Dhungana. Supporting Product Derivation by Adapting and Augmenting Variability Models. *Software Product Line Conference, 2007. SPLC 2007. 11th International*, pages 141–150, 2007.

[12] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press Cambridge, MA, USA, 1989.

[13] J. White, K. Czarnecki, D. C. Schmidt, G. Lenz, C. Wienands, E. Wuchner, and L. Fiege. Automated Model-based Configuration of Enterprise Java Applications. In *The Enterprise Computing Conference, EDOC*, Annapolis, Maryland USA, October 2007.