Metrics on Feature Models to Optimize Configuration Adaptation at Run Time

Luis Emiliano Sanchez
Universidad Nacional del Centro
de la Provincia de Buenos Aires
Tandil, Argentina
lsanchez@alumnos.exa.unicen.edu.ar

Sabine Moisan, Jean-Paul Rigault
INRIA Sophia Antipolis Méditerrannée
Route des Lucioles
06902 Sophia Antipolis Cedex, France
{sabine.moisan,jean-paul.rigault}@inria.fr

***Abstract*—Feature models are widely used to capture variability, commonalities and configuration rules of software systems. We apply this technique to model component-based systems with many variants during specification, implementation, or run time. This representation allows us to determine the set of valid configurations befitting a given context, especially at run time. A key challenge is to determine the configuration most suitable, especially with respect to non-functional aspects: quality of service, performance, reconfiguration time... We propose an algorithm for selecting the configuration that optimizes a given quality metrics. This algorithm is a variant of the Best-First Search algorithm, a heuristic technique suitable for feature model optimization. The algorithm is parameterized with several strategies and heuristics on feature models leading to different optimality and efficiency properties. We discuss the algorithm, its strategies and heuristics, and we present experimental results showing that the algorithm meets the requirements for our real time systems.**

***Index Terms*—software metrics, feature model optimization, heuristic search in graphs, real time adaptation, search-based software engineering**

I. Introduction

Feature Modeling is a Model-Driven Engineering method that has emerged as a simple technique for representing commonalities and varying aspects of software product lines and their configuration rules. Features correspond to selectable concepts of a system at a chosen abstraction level: functional and non-functional requirements, environment and context restrictions, run time components... Systems and their variability are represented by *Feature Models* that can be manipulated through *model transformations*. A feature model is a set of features organized along a tree-like structure that defines the relationships among features in a hierarchical manner, with logical relations (optional, mandatory, alternative...). Extra constraints such as implication or exclusion of features restrict the valid combination of features across different sub-trees (*cross-tree constraints*). Thus a feature model defines the set of valid *configurations* of a system.

In previous work [1] we proposed the use of feature models for the representation and dynamic adaptation of component-based systems such as video-surveillance processing chains. Following *model at run time* techniques, we are able to determine the set of valid configurations to apply in a given execution context. Of course only one configuration can be applied at a given time and the problem is to select the "best" one. Here, "best" is a trade-off between several *quality attributes*. A quality attribute is a monotonic quantification of an aspect of the system quality. Examples are response time, accuracy, availability...

It is thus necessary to rank the possible configurations. Our approach is to define *quality metrics* for comparing configurations. The challenge comes down to the well studied problem of *Feature Model Optimization* [2], i.e. to find a valid configuration that minimizes a given *objective function*. This combinatorial optimization problem is known to be intractable in general because the number of valid configurations increases exponentially with respect to the number of optional features. It is also a multi-objective (or multi-criteria) optimization problem involving several aspects simultaneously.

The solution proposed here relies on the *Best-First Search algorithm* [4], a well studied technique for problems that are deterministic, observable, static, and completely known, as feature models. This algorithm offers a set of proven characteristics with respect to completeness, efficiency, and optimality that meet the requirements of real time systems.

The remainder of this paper is organized as follows: Section 2 presents related works. Section 3 briefly describes the application domain and introduces metrics for comparing configurations. Section 4 details the configuration selection algorithm with its associated strategies and heuristics. Section 5 experimentally compares several variants of the algorithm. Section 6 concludes and outlines future work.

II. Related Work

The algorithmic approaches to feature model optimization differ in the applied technique, the application domain, and the properties concerning time efficiency and optimality.

Feature model optimization can be addressed as a Constraint Satisfaction Problem in [5]. General purpose CSP solvers can cope with any mathematical expression as objective function. However, this approach has exponential complexity because it requires to explore and evaluate all configurations.

Planning algorithms can find an approximate solution. Besides, [6] deals with the multi-objective nature of the problem, by defining the objective function as a linear combination of different quality attributes, weighted along stakeholder preferences. In addition, it takes into account inequality constraints

978-1-4673-6284-9/13/$31.00 © 2013 IEEE

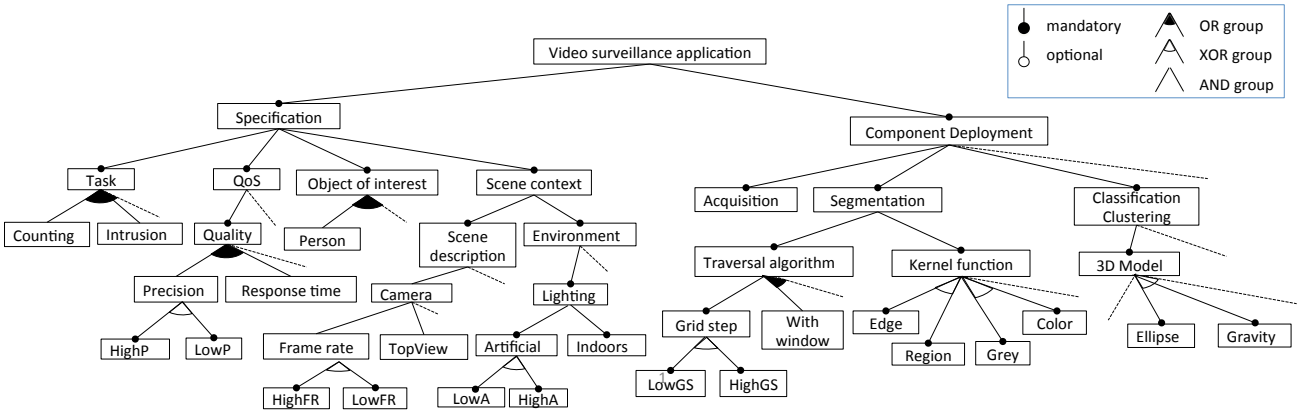

Fig. 1. Excerpts of a feature model for the specification and component view of a video surveillance system

to deal, e.g., with bounded resource. However, this approach is limited to additive objective functions.

Other solutions are inspired by traditional AI algorithms. For instance, [8] defines a feature model transformation, called *Filtered Cartesian Flattening*, that refactors a feature model so that the optimization problem becomes a Multi-Dimensional Multi-Choice Knapsack Problem. Although this approach takes into account inequality constraints, the objective function is limited to a single quality attribute and therefore cannot deal with the optimization of several requirements simultaneously.

Configuration selection may also be based on genetic algorithms. In [7] the authors address the optimization problem as in the previously mentioned work [8]. Therefore, it cannot deal simultaneously with several criteria either.

Several aspects distinguish our work. First, we want to address optimization of feature models in *real time*, whereas most studies consider static analysis of feature models and configuration selection of software product lines at design time only. Secondly, our approach offers different strategies and heuristics providing either an optimal solution or approximate ones. Thus, it can be applied at design time, where optimality is desirable, as well as at run time where computational efficiency is mandatory. Lastly, although we do not yet consider inequality constraints, we address the optimization of the most usual forms of objective functions suggested as quality metrics as shown in table I: addition, product, maximum and minimum of attributes, and linear combinations of them to weight requirements thus dealing with different quality attributes simultaneously.

III. Application and Metrics

The run time *system configuration* of a video-surveillance processing chain can be simply defined as a set of running components, that can be tuned, removed, added or replaced dynamically. We use feature modeling to represent the variability of both the specification and component views of such systems, as is shown in figure 1. We also include cross-tree constraints that formalize extra feature dependencies. In this feature model, it is important to distinguish between *concrete* and *abstract* features. The first ones are low-level features that reference concrete software elements, like deployable components or configuration parameters. By contrast, abstract features are used to represent specification aspects and to organize the whole system.

A system configuration $\mathbb{C}$ is represented by a feature model *full configuration* defined as a 2-tuple $\langle S, D\rangle$ where S and D are sets of selected and deselected features respectively, such that $S \cap D = \emptyset$, $S \cup D = F$ (set of all features). Context changes or user interactions imply to dynamically reconfigure the model (selecting or deselecting features). The reconfiguration seldom results in a full system configuration but rather in a *partial configuration* (i.e., a sub-model) of the feature model. A partial configuration represents the set of valid full configurations compatible with a given execution context. It is defined as a 3-tuple $\langle S, D, U\rangle$ where U is the set of *unselected* (i.e., *unassigned*) features, such that S, D and U are pairwise disjoint and that $S \cup D \cup U = F$.

The objective of the *configuration selection algorithm* is to derive an optimal full configuration from a given partial configuration. This process consists in selecting or deselecting unselected features until U becomes empty. It can propagate other selections and deselections in order to satisfy logical relations and cross-tree constraints. Of course, invalid configurations are skipped.

An *optimal* system configuration $\mathbb{C}$ minimizes a given *objective function*. In our case we consider linear weighted functions $L(\mathbb{C}) = \sum_{a \in A} w_a \times M_a(\mathbb{C})$, where A is the set of quality attributes of interest, w_a is the weight of quality attribute a, and M_a is the *quality metrics* associated with attribute a. M_a may have different forms according to the nature of the attributes. There exist a wide range of quality attributes to evaluate run-time operations of a system. Some are general: response time, memory consumption, availability... Besides, video-surveillance systems exhibit specific attributes: accuracy and sensitivity of detection or tracking algorithms, reconfiguration time, relevance of object classification... Different quality metrics functions can be used to evaluate all these attributes. We identified four of them that are appropriate most of the time and that have mathematical properties suitable for optimization

in feature models. Table I presents a simplified version of them. The linear combination of these metrics in an objective function such as $L(\mathbb{C})$ allows us to deal simultaneously with several quality attributes, different weights, and different quality metrics functions.

TABLE I
DIFFERENT FORMS OF METRICS FOR A GIVEN QUALITY ATTRIBUTE a.

Quality metrics function	Quality Attributes
$M^{+}(\mathbb{C}) = \sum_{f \in S} a(f)$	memory consumption, reconfiguration time...
$M^{\times}(\mathbb{C}) = \prod_{f \in S} a(f)$	accuracy, availability...
$M^{M}(\mathbb{C}) = \max_{f \in S}(a(f))$	latency, response time in parallel execution...
$M^{m}(\mathbb{C}) = \min_{f \in S}(a(f))$	security...

To compute the objective function, we enriched all features in the model with a slot $a(f)$ for each quality attribute a. Concrete features may have predefined values for some attributes. The other slots are initialized by default with the neutral element of their specific metrics function: 0 for addition, 1 for product, ∞ for minimum, and $-\infty$ for maximum. Owing to the tree structure of feature models and their configurations, the metrics in table I can be recursively computed for each quality attribute. Note that deselected features usually do not contribute to the metrics, that is why they do not appear in table I, but some of them may have an impact. For instance, a newly deselected feature in a configuration may imply to take into account the corresponding component shutdown time, just as we consider its component setup time when the feature is selected. Equation 1 shows the evaluation of a quality attribute a in the particular case of an additive metrics (M^{+} in table I).

$$M_a^{+}(\mathbb{C}) = R_a^{+}(r)$$
$$R_a^{+}(f) = \begin{cases} a(f) & \text{if } f \text{ is leaf} \\ a(f) + \sum_{f' \in S'} R_a^{+}(f') & \text{otherwise} \end{cases} \quad (1)$$

The recursive evaluation starts from the root feature r of the model, that belongs to S by definition. S' is the set of selected children of a feature f.

Due to the existence of a neutral element and to the associativity of the four quality metrics, the metrics in table I and the recursive equations 1 are equivalent. Therefore, the linear weighted objective function can also be redefined as a linear combination of the recursive functions R_a as shown in equation 2.

$$L(\mathbb{C}) = \sum_{a \in A} w_a \times M_a(\mathbb{C}) = \sum_{a \in A} w_a \times R_a(r). \quad (2)$$

IV. CONFIGURATION SELECTION ALGORITHM

The configuration selection algorithm is based on the Best-First Search algorithm. This algorithm performs a systematic search over an abstract structure called *state-space graph*. In our problem, this structure is a directed graph where nodes are valid states of the problem (partial and full valid configurations) and edges are feature model transformations (selection and deselection of unselected features).

From a given initial partial configuration, the configuration selection algorithm (Algorithm 1) generates new nodes by selecting and deselecting features. It uses an heuristic function to estimate the quality of these nodes in order to drive the search. The algorithm succeeds when it reaches a full configuration (goal node). Due to the directed graph structure of the state-space graph, the algorithm needs to remember the visited nodes in order to avoid redundant paths. As this compromises the efficiency of the algorithm, we build the state-space graph as a binary tree. To do that, for each partial configuration (nodes of the state-space graph), the algorithm chooses one unselected feature and generates at most two successors, one selecting this feature and the other deselecting it. A simple example of a binary tree structured state-space graph is depicted in figure 2.

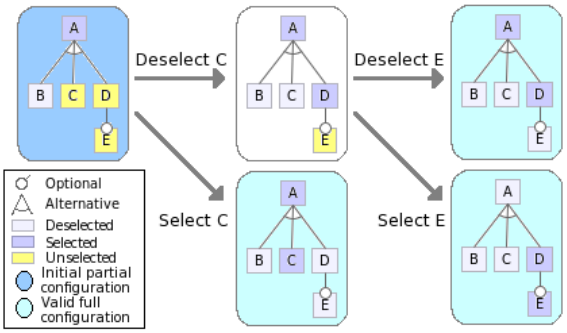

Fig. 2. Example state-space graph

The algorithm, implemented in C++, has three variation points. First, the *OPEN set* is a container where the algorithm adds and removes the generated nodes (together with the value of the heuristics used to prioritize partial solutions). It can be implemented with different structures, like a stack or priority queue, corresponding to different *search strategies* (discussed in section IV-A).

Second, *the heuristics* h (lines 11 and 15 of Algorithm 1) constitutes another variation point. Each node of the state-graph represents generally a partial configuration, that is a *set* of full configurations. The later set increases exponentially with the number of unselected features. Thus exactly determining the value of the optimal full configuration with respect to an objective function is difficult. Hence the algorithms uses an *heuristic function* to get a quick estimate of the objective function. Several heuristics are possible and discussed in section IV-B.

Last, *feature selection* (line 8) chooses the next unselected feature candidate to be selected/deselected. Following existing work in Constraint Satisfaction Problems, we intend to choose features in order to avoid decisions that can lead to invalid nodes. For instance, *structural metrics* on feature models (measuring size, complexity and other structural properties) could be used for this purpose. However, in our current implementation we merely choose the unselected features in an

Algorithm 1 Configuration Selection Algorithm

Input: $initial$ // partial configuration
Output: $goal$ // full configuration
1: add $initial$ into $OPEN$ set
2: **while** $OPEN$ is not empty **do**
3: remove some configuration $conf$ from $OPEN$
4: **if** $conf$ is full configuration **then**
5: $goal \leftarrow conf$
6: exit successfully
7: **else**
8: $f \leftarrow$ next unselected feature in $conf$ // feature selection
9: $nS \leftarrow conf$ where f is selected
10: **if** nS is valid **then**
11: evaluate nS with h; add pair <nS,$h(nS)$> to $OPEN$
12: **end if**
13: $nD \leftarrow conf$ where f is deselected
14: **if** nD is valid **then**
15: evaluate nD with h; add pair <nD,$h(nD)$> to $OPEN$
16: **end if**
17: **end if**
18: **end while**

arbitrary order; indeed, we prefer to focus on the interactions between the search strategies and their companion heuristics.

A. Search Strategies

Search strategies decide which node to visit next, defining different traversals. We choose two well-known informed strategies that rely on heuristic functions as choice criteria and that present interesting properties in terms of efficiency and optimality. They are analyzed empirically in section V.

1) Best-First search Star (BF):* This strategy is a variant of the A* algorithm and is described in [4] under the name BF*. We use a node-cost function instead of a path-cost one as in original A*. It favors optimality over efficiency. If the heuristic function is *exact*, i.e. if it computes the value of the best full configuration, the corresponding search reaches the optimal solution in the minimum number of visited nodes. As the heuristics becomes less exact, this strategy backtracks more often. However, it still ensures the optimality of the solution if the heuristics is *admissible*, i.e. if it never overestimates the value of the best solution, yet with exponential time and memory complexity in the worst case.

2) Greedy Best-First Search (GBFS): In this strategy the next visited node is the best successor of the current node. In contrast to BF*, it favors efficiency over optimality because it only backtracks when an invalid node is reached. If the heuristic is exact, it behaves as BF* finding the optimal solution. This strategy has exponential time cost in general but polynomial cost on feature models without cross-tree constraints (as confirmed by our experiments, see V). Regarding memory complexity, it is linear in general with respect to the number of unselected features.

B. Heuristics

Heuristics evaluate a partial configuration giving a quick estimate of the quality (value of the linear weighted function used as objective function) of the set of full configurations that it represents. The more exact the heuristics is, the more optimal and efficient the search strategies are. GBFS and BF* require the exact best value to drive the search to the optimal solution in optimal time. Unfortunately, due to cross-tree constraints, exact functions evaluable in polynomial time can seldom be defined. Yet, we can find admissible heuristics that are exact for *relaxed models* of the problem in such a way that BF* can reach an optimal solution within reasonable time and GBFS can improve the optimality of the approximate solution.

A *relaxed model* [4] is a model with fewer restrictions. The state-space graph of a relaxed model is a super-graph of the original solution space since removing restrictions introduces new nodes and edges in the graph. As the relaxed model just extends the state-space graph, any optimal solution of the original problem is also a solution (not necessarily optimal) in the relaxed version. Hence, an exact heuristics for a relaxed model is admissible for the original model. We defined two types of relaxed models A and B such that, for a given feature model FM, its relaxed models FM_A and FM_B are generalizations of FM, i.e., $[[FM]] \subseteq [[FM_B]] \subseteq [[FM_A]]$ where $[[FM]]$ denotes the set of valid full configurations of FM. For each relaxed model, we defined an *exact* heuristic function (resp. H_A and H_B) that is thus *admissible* for general feature models.

We present simplified versions of both heuristics illustrating the general idea. We do not consider details about evaluation of all kinds of feature relation (mandatory, optional...) or of choice of quality metrics (addition, product...).

1) Relaxed feature model of type A: This model has only mandatory and optional features (AND groups) and no cross-tree constraints. Features in XOR and OR groups are treated as optional. To evaluate partial configurations the heuristic function H_A is defined recursively as in equation (1) with an extra term for unselected features. For the latter, we chose the best value between selecting and deselecting the feature. Equation 3 describes the computation of H_A to minimize a quality attribute a with an additive associated metrics.

$$H_A(f) = \begin{cases} a(f) & \text{if } f \text{ is leaf} \\ a(f) + \sum_{f' \in S'} H_A(f') & \\ \quad + \sum_{f' \in U'} min(H_A(f'), 0) & \text{otherwise} \end{cases} \quad (3)$$

2) Relaxed feature model of type B: In this less relaxed version, only cross-tree constraints are removed. The heuristic function H_B is similar to H_A but alternative groups (XOR) are treated differently: as only one of their children can be selected, the minimum value is chosen. OR groups are similar to AND groups but more complex and are not detailed here. For example, equation 4 describes the computation of H_B to minimize a quality attribute a with an additive metrics.

$$H_B(f) = \begin{cases} a(f) + \min_{f' \in U'}(H_B(f')) & \text{XOR} \\ \dots & \text{OR omitted} \\ H_A(f) & \text{otherwise} \end{cases} \quad (4)$$

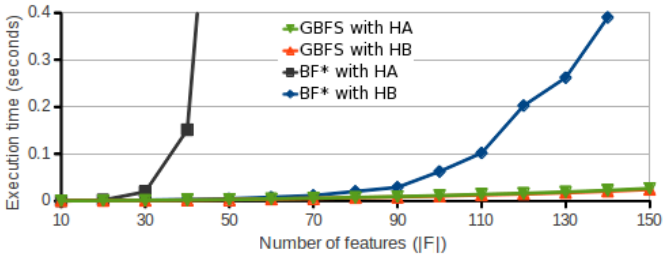

Fig. 3. Time efficiency comparison. BF* execution time rises exponentially and GBFS polynomially (green and red GBFS curves overlap).

By analogy, these heuristics are also defined for the product, maximum and minimum metrics. They evaluate partial configurations from leaves to root with polynomial time complexity. These heuristics are equivalent to equation (1) to evaluate full configurations, and thereby they are also equivalent to quality metrics in table I. Equation 2 applies, replacing R with H_A or H_B. Therefore, a general heuristics to estimate the value of a linear weighted objective function can be defined as the same linear combination of heuristic functions.

H_A and H_B are exact for the relaxed models A and B when the linear weighted function is reduced to one attribute (only one metrics). For the general case, these functions are not exact but only admissible: the estimated value is always less or equal than the real value of the best solution. The admissibility of these heuristics is an important property that ensures that solutions of BF* strategy are optimal for general linear weighted functions and general feature models.

V. Evaluation

We have run experiments and compared completeness, optimality and efficiency of the previous algorithms and heuristics. Regarding completeness, Best-First Search is a *systematic* algorithm [4]: it visits all the nodes in the worst case, but only once, which ensures completeness and termination.

We have already discussed in the previous section time and memory complexity of search strategies and optimality and efficiency of BF* when heuristics are admissible or exact. In this section we empirically measure these properties with a series of experiments using randomly generated feature models. Of course these models are validated, i.e. we verify that they have at least one valid full configuration.

Both search strategies (GBFS and BF*) combined with both heuristics (H_A and H_B) are tested and compared. Three main feature model parameters are considered for analyzing the scalability of these variants: number of features ($|F|$), number of cross-tree constraints, and number of terms (i.e., number of quality metrics $|A|$) of the linear objective function. We focused on optimality of solutions and execution time of algorithms, the most representative results for comparison purposes. Each experiment is performed with at least 100 feature models as samples, and the average value is computed in order to reduce the fluctuations due to random generation. All measurements were performed on Linux Fedora 14 with Intel Xeon CPU 2.4 GHz and 16 GB RAM.

A. Scalability

The first experiment (figure 3) evaluates the scalability of the four variants of the algorithms with respect to the total number of features. The feature models are generated as balanced trees with a branching factor of 5 children per feature. Feature relationships are selected randomly among *mandatory* or *optional*, XOR (alternative) and OR groups. The quality attribute values are set randomly with a uniform distribution over (0,1). No cross-tree constraints are included. The objective function is a linear weighted combination of four different quality metrics:
$L(C) = w_1 \times \sum_{f \in F} a_1(f) + w_2 \times \prod_{f \in F} a_2(f) + w_3 \times \max_{f \in F}(a_3(f)) + w_4 \times \min_{f \in F}(a_4(f))$.

Since there are no cross-tree constraints, the GBFS strategy never backtracks and both GBFS versions are fully scalable with respect to the number of features, as shown on figure 3. For BFS, both versions obtain the optimal solution but one can see that performance strongly depends on the heuristics used. Therefore heuristics B is obviously better because of its outstanding improvement in performance.

B. Efficiency of Heuristics in BF*

The efficiency of heuristics is also analyzed with respect to others parameters. Figure 4 shows two experiments, where the number of cross-tree constraints and the number of terms in the objective function vary.

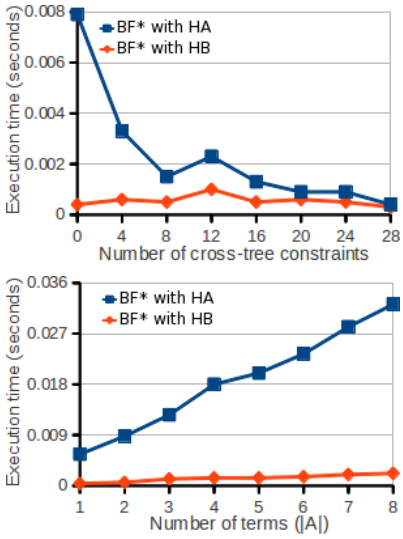

Fig. 4. Time efficiency comparison of heuristics using BF* strategy

In the first experiment (up), we consider a single additive function to minimize (i.e., a single quality attribute). Feature models are generated randomly with 30 features, because with this value the execution time of both heuristic functions is almost equal, as seen on figure 3. Simple cross-tree constraints are added, like "imply" and "exclude", that are 2-SAT clauses in conjunctive normal form (e.g $f_1 \rightarrow f_2 \equiv \neg f_1 \vee f_2$). The curves show how the time efficiency improves as cross-tree constraints are added. Even though the heuristics ignore them, incorporating restrictions reduces the number of valid configurations exponentially. The general case with k-SAT cross-tree constraints is not considered in these experiments although the algorithm can cope with it.

In the second experiment (down), feature models without cross-tree constraints are evaluated. For each instance of the

experiment, a new metrics that evaluates the contribution of a different attribute is added to the linear objective function. The curve shows how the execution time increases with the number of metrics: indeed the linear combination of heuristic functions becomes less exact with respect to the linear combination of quality metrics as new terms are added, leading to worse performance of BF*.

C. Optimality of Heuristics in GBFS

The execution time of both heuristics is similar ($O(n^2)$, where n is the number of features). Therefore they do not have a significant impact on the efficiency of the GBFS strategy, as shown in figure 3. However, it is interesting to compare the impact of the heuristics on the optimality of solutions, the strategy being constant. Figure 5 shows two experiments using the GBFS strategy, where the number of cross-tree constraints and the number of evaluated metrics vary. The optimality of a solution is measured by the following ratio: $|worst - solution| \,/\, |worst - best|$, where the value of best and worst solutions are calculated with BF*.

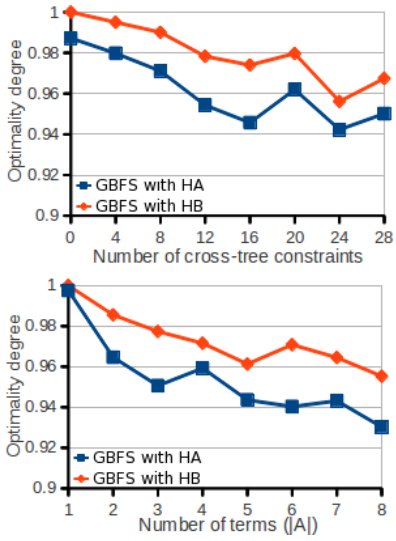

Fig. 5. Optimality comparison of heuristics using GBFS strategy

Both experiments were conducted in the same conditions as before (30 features per sample, branching factor of 5, one single additive metrics to minimize for the first experiment, and no cross-tree constraints for the second). They show how optimality decreases as the density of constraints increases and as the linear objective function incorporates more terms, because, as explained before, under these conditions the heuristics becomes less exact. However, both heuristics yield configurations that are more than 90% optimal. In an other experiment, we have shown that the number of features has no impact on optimality.

VI. Conclusion and Future Work

From our experiments, the GBFS strategy with heuristics H_B appears as the ideal option for real time systems that have to adapt in bounded time. This strategy ensures polynomial time complexity for relaxed feature models and guarantees over 90% optimality, which is good enough for our purpose.

On the other hand, BF* strategy with heuristics H_B is ideal for assisting design decisions, such as product configuration and generation of software product lines from feature models. This search strategy takes a significant time to compute a solution. It is not appropriate in real time unless the feature models have less that about 130-140 features for which the execution time is about 0.4 seconds. But for decisions during design, extra time is acceptable to obtain the optimal configuration.

This work only considers the most important variants of the Best-First Search algorithm. However, we can extend the algorithm incorporating new variants in both dimensions: search strategies and heuristic functions.

Regarding search strategies, hybrid solutions could be considered [4]. Only the implementation of the $OPEN$ set (see algorithm 1) needs to be changed. For example, it could start as a priority queue and continue as a stack, combining BF* and GBFS strategies, and thereby getting different efficiency-optimality trade-offs. The A* algorithm is a popular version of the Best-First Search algorithm. Although it uses a path cost function instead of a node cost function as we do, we could integrate some of the improvements proposed by its many variants into our approach.

Regarding heuristics, we have not yet considered non-admissible ones, i.e. heuristics that overestimate the value of the best solution. They could allow us to relax the optimality requirement: BF* with non-admissible heuristics does not ensure optimality but its performance improves significantly.

The proposed approach still has a third variation point that was not explored yet. As mentioned in section IV, using CSP heuristics for feature selection could improve the performance of the algorithm in feature models with many cross-tree constraints, especially with k-SAT clauses (with $k > 2$) since in this case finding an optimal solution is harder [3].

Finally, the approach could be extended to incorporate inequality restrictions, e.g., maximum memory consumption, maximum number of running components... as part of the definition of the optimization problem. The general algorithm scheme doesn't need to be changed. Only the design of new specific heuristics should be considered.

References

[1] S. Moisan, J. Rigault, M. Acher, P. Collet, P. Lahire. *Run Time Adaptation of Video-Surveillance Systems: A Software Modeling Approach*. Proc. ICVS 2011, LNCS 6962, , pp 203-212, Springer.

[2] D. Benavides, S. Segura, A. Ruiz Cortes. *Automated Analysis of Feature Models 20 Years Later: A Literature Review*. Journal Information Systems, Volume 35, Issue 6, pp 615-636. September, 2010.

[3] M. Mendonca, A. Wasowski, and K. Czarnecki. *SAT–based analysis of feature models is easy*. In Proceedings of the Software Product Line Conference, 2009.

[4] J. Pearl. *Heuristics: Intelligent search strategies for computer problem solving*. Reading, Mass. Addison-Wesley, 1984.

[5] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. *Automated reasoning on feature models*. Advanced Information Systems Engineering: 17th Int'l conf, pp. 491–503., 2005.

[6] S. Soltani, M. Asadi, D. Gašević, M. Hatala, E. Bagheri. *Automated planning for feature model configuration based on functional and non-functional requirements*. In Proc. 16th International Software Product Line Conference. Vol. 1, pp. 56-65, 2012.

[7] J. Guo, J. White, G. Wang, J. Li, and Y. Wang. *A genetic algorithm for optimized feature selection with resource constraints in software product lines*. Journal of Systems and Software, vol. 84, no. 12, pp. 2208–2221, Dec. 2011.

[8] J. White, B. Dougherty, and D. C. Schmidt. *Selecting highly optimal architectural feature sets with Filtered Cartesian Flattening*. J. Systems & Software, vol. 82, p. 1268–1284, 2009.