Modelo Computacional para Apoiar a Configuração de Produtos em Linha de Produtos de Software

Juliana Alves Pereira, Eduardo Figueiredo, Thiago Noronha

Laboratório de Engenharia de Software (LabSoft), Departamento de Ciência da Computação (DCC), Universidade Federal do Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

{juliana.pereira, figueiredo, tfn}@dcc.ufmg.br

Abstract. *Software Product Line (SPL) is a set of software systems QUE sharing a set of characteristics to satisfy specific needs of a particular domain. The configuration of a product among thousands of possible combinations of features has proved to be impractical even for small SPL. To support companies in semiautomatic product extraction and maximize customer satisfaction, this paper proposes a model based on search algorithms. The model has two implementations: (i) exhaustive enumeration with preprocessing and backtracking and (ii) a greedy algorithm. Due the NP-complete nature of the problem, our experiments revealed that the exhaustive enumeration is impractical for larger instances. Moreover, the greedy heuristic implementation solves the problem in polynomial time with 92% of accuracy.*

Resumo. *Linha de produtos de software (LPS) é um conjunto de sistemas de software que compartilham um conjunto de características, e que satisfazem as necessidades específicas de um determinado segmento de mercado. A configuração de produtos, dentre as milhares de combinações de características possíveis, tem se mostrado inviável mesmo para pequenas LPS. A fim de apoiar as empresas na configuração semi-automática de produtos que maximizem a satisfação de clientes, este trabalho propõe um modelo baseado em algoritmos de busca e otimização. O modelo foi implementado utilizando duas soluções: (i) algoritmos de enumeração exaustiva com pré-processamento e backtracking e (ii) heurística gulosa. Devido à complexidade NP-completo, os experimentos realizados mostraram que a solução por enumeração exaustiva é inviável para grandes instâncias do problema. Por outro lado, a implementação da heurística gulosa resolve o problema em tempo polinomial com uma taxa de acerto de 92%.*

1. Introdução

A crescente necessidade de desenvolvimento de sistemas de software maiores e mais complexos exige melhor suporte para reutilização de software [10]. Muitas técnicas, como Linha de Produtos de Software (LPS), têm sido propostas para oferecer avançado suporte a reutilização de software. LPS pode ser definida como um conjunto de sistemas de software definidos sobre uma arquitetura comum, que compartilham um mesmo conjunto de características e satisfazem as necessidades específicas de um determinado segmento de mercado [10]. Uma característica é um incremento na funcionalidade de um sistema de software [1]. LPS explora o fato de sistemas em um mesmo domínio serem semelhantes e terem potencial para reutilização.

Grandes empresas de software, tais como Boeing, Hewlett Packard, Nokia e Siemens, adotam LPS para desenvolver seus produtos [11]. LPS permite às empresas

rápida entrada no mercado por fornecer capacidade de reutilização em larga escala com customização em massa [10]. Pela adoção desta técnica, espera-se obter ganhos na produtividade, produtos mais confiáveis e preço mais acessível [14]. Esses potenciais benefícios prometem um aumento significativo na satisfação do cliente. Entretanto, a customização de produtos para maximizar a satisfação do cliente é uma tarefa cuja complexidade aumenta exponencialmente à medida que aumenta o número de características envolvidas e restrições a serem consideradas na LPS [1][2][3][8].

Para minimizar este problema, este trabalho propõe um modelo para apoiar a configuração de produtos em LPS através da utilização de algoritmos de busca e otimização. O objetivo do modelo é maximizar a satisfação do cliente. Este modelo foi implementado inicialmente utilizando algoritmos de enumeração exaustiva com pré-processamento e backtracking [5]. Devido à natureza NP-Completo [5] deste problema, à medida que a instância do problema cresce, estes algoritmos passam a exigir muitos recursos computacionais (essencialmente, tempo de processamento) para encontrar a solução ótima. Portanto, é proposta a utilização de um algoritmo guloso de busca, baseado em heurísticas, para encontrar soluções suficientemente boas em um menor horizonte de tempo.

Os estudos experimentais realizados mostram a complexidade exponencial da resolução por enumeração exaustiva, mesmo com o uso de pré-processamento e backtracking. Por outro lado, o algoritmo guloso proposto possui complexidade polinomial. Além do ganho em recursos computacionais, o algoritmo guloso apresenta uma taxa de acerto de 92% em relação aos algoritmos de enumeração exaustiva como mostram os resultados dos experimentos.

O restante do artigo está organizado da seguinte forma. A Seção 2 introduz os conceitos fundamentais de LPS e define o problema tratado pelo trabalho. A Seção 3 apresenta a modelagem do problema e os algoritmos implementados. A Seção 4 discute os resultados dos experimentos computacionais realizados. A Seção 6 conclui o trabalho e propõe direções para trabalhos futuros.

2. O Problema de Configuração de Produtos em LPS

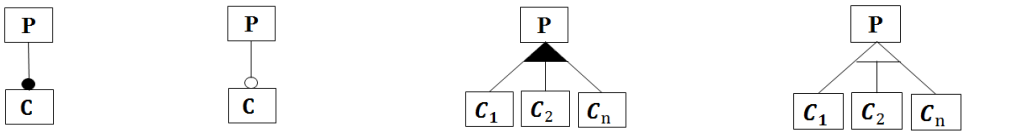
O problema de configuração de produtos em LPS corresponde ao problema de customizar um produto para atender as necessidades especificas de um cliente. A Seção 2.1 introduz os conceitos de LPS e a Seção 2.2 descreve o problema tratado pelo trabalho.

2.1. Linha de Produtos de Software

Em uma Linha de Produtos de Software (LPS), características comuns a um domínio formam o núcleo e outras características definem pontos de variação [10]. Uma característica é um incremento na funcionalidade de um sistema de software [1]. Em uma LPS, o domínio do sistema é decomposto em características coerentes, bem definidas, independentes e facilmente combináveis. Assim, diferentes configurações de produtos podem ser possíveis mediante a inclusão ou exclusão de determinadas características [4]. A comercialização de produtos que se diferenciam por variações em suas características está se tornando comum e grandes empresas têm investido em LPS [11]. Por exemplo, os automóveis de um mesmo modelo se diferenciam por itens como airbag. Existem características comuns a todos os produtos e outras características podem variar de um produto para outro.

Modelo de características é o padrão para representar a variabilidade de uma LPS e o espaço de possíveis configurações de produtos [13]. Um modelo de características, geralmente representado utilizando árvores, possibilita a visualização hierárquica das características de uma LPS e suas relações [12][13]. As características do modelo de características são classificadas em mandatórias, opcionais, alternativas não exclusivas (*OR*) e alternativas exclusivas (*XOR*).

As características mandatórias, representadas por um círculo cheio na Figura 1a, devem obrigatoriamente estar presente na composição de um produto. As características opcionais, representadas por um círculo vazio na Figura 1b, podem opcionalmente estar presentes na composição de um produto. No caso do grupo de características alternativas não exclusivas, representadas por arestas interligadas e conectadas por um arco cheio na Figura 1c, uma ou mais sub-características podem ser selecionadas para composição de um produto. No grupo de características alternativas exclusivas, representado por arestas interligadas e conectadas por um arco vazio na Figura 1d, apenas uma sub-característica pode ser selecionada para composição de um produto.



(a) Mandatória (b) Opcional (c) Alternativa não Exclusiva (d) Alternativa Exclusiva

Figura 1 Classificação das características

Além das características e suas relações, um modelo de características pode incluir regras de composição referentes às restrições adicionais para combinações de características [10]. As regras de composição de características são responsáveis pela validação da configuração de produtos. Elas normalmente descrevem uma restrição de inclusão ou de exclusão enunciadas da forma: se a característica C_1 for incluída, então a característica C_2 também deve ser incluída (ou excluída). Por exemplo, no modelo de características de um automóvel (vide Figura 2). Onde, *Som Automotivo* e *Bateria do Tipo 1* são características da LPS. Uma regra de composição: *Som Automotivo* requer (inclusão) *Bateria do Tipo 1* é definida. Neste caso, se a característica *Som Automotivo* for selecionada para compor um produto, obrigatoriamente a característica *Bateria do Tipo 1* deve ser selecionada para compor o mesmo produto.

2.2. Descrição do Problema

A configuração de produtos em LPS corresponde a customizar produtos para clientes baseando-se em restrições (como por exemplo, preferências e orçamento). Isso implica em selecionar características da LPS que satisfaçam restrições impostas pelas empresas fornecedoras do produto e pelos clientes que irão adquirir o produto. Na implementação deste problema, como exemplificado pela Figura 2, as restrições são representadas pelo orçamento disponível para aquisição do produto, preço e grau de importância das características que compõem o produto. Assim, com base no orçamento que o cliente dispõe para aquisição do produto, o objetivo é otimizar a configuração de características e apoiar as empresas na configuração de produtos que maximizem a satisfação de clientes.

As principais constantes definidas pelo modelo do problema são: (i) o limite de orçamento que o cliente possui para aquisição do software ($O \in Reais$); (ii) o grau de importância da característica *i* para o cliente ($G_{1..n} \in [0, 1]$); (iii) o preço em adicionar a característica *i* ao produto a ser entregue ao cliente ($P_{1..n} \in Reais$); e (iv)

a expressão proposicional na forma normal conjuntiva derivada a partir da árvore modelada (E(C)). Onde, C(n)∈ Naturais representa a quantidade de características presentes no modelo e $C_i \in \{0, 1\}$ representa a não inclusão ou inclusão da característica *i* no produto configurado. Adicionalmente, todas as restrições definidas pelo modelo devem ser satisfeitas.

A função objetivo é maximizar $z(n) = \sum_{i=1}^{n} G_i C_i$.

Sujeito a: $\sum_{i=1}^{n} P_i C_i \leq O$, satisfazendo E(C).

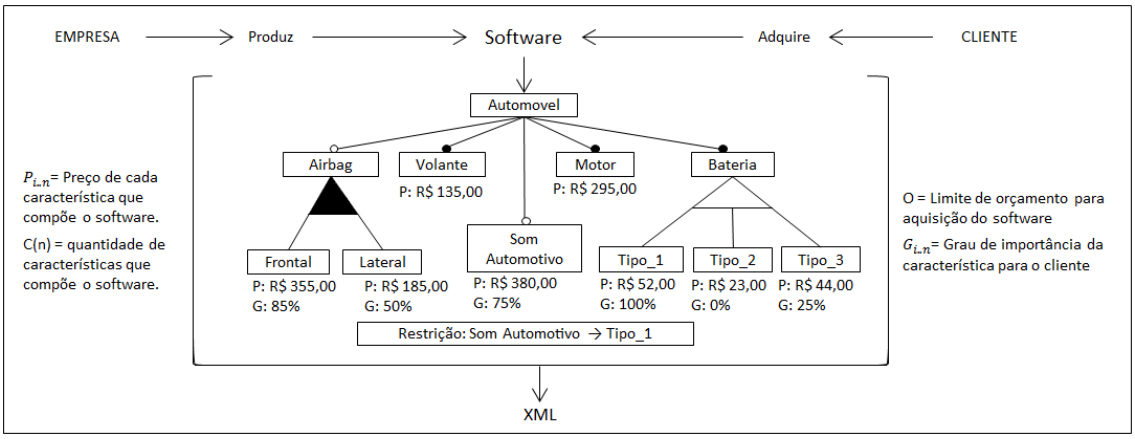


Figura 2 Modelo para apoiar a configuração de produtos em uma LPS

3. Modelo Computacional Proposto

Durante a configuração de produtos, o crescimento exponencial da quantidade de combinações possíveis das características em um modelo varia em relação à quantidade de características que o modelo possui. A Seção 3.1 descreve o modelo de otimização utilizado, e as Seções 3.2 e 3.3 apresentam a implementação de uma solução exponencial (algoritmo de enumeração exaustiva) e polinomial (solução gulosa) implementadas para o problema.

3.1. Modelo de Otimização

O problema de otimização combinatória descrito na Seção 2 pertence à classe de complexidade NP-completo [5]. Uma proposta para a análise automatizada de modelos de características é baseada no mapeamento dos modelos em resolvedores SAT (problema da satisfazibilidade booleana) [3]. As regras para a tradução de modelos de características às restrições são listadas na Tabela 1. Estas regras possibilitam a modelagem do problema através de problemas conhecidos de otimização. Modelamos o problema como uma combinação de MAX-SAT Ponderado [9] com o Problema da Mochila [7].

Tabela 1. Representação matemática dos tipos de características para SAT

Relação	SAT
Mandatória	$(\neg P \vee C) \wedge (\neg C \vee P)$
Opcional	$\neg C \vee P$
Alternativa não Exclusiva	$(\neg P \vee C_1 \vee C_2 \vee \ldots \vee C_n) \wedge (\neg C_1 \vee P) \wedge (\neg C_2 \vee P) \wedge \ldots \wedge (\neg C_n \vee P)$
Alternativa Exclusiva	$(C_1 \vee C_2 \vee \ldots \vee C_n \vee \neg P) \wedge (\neg C_1 \vee \neg C_2) \wedge \ldots \wedge (\neg C_1 \vee \neg C_3) \wedge (\neg C_1 \vee P) \wedge (\neg C_2 \vee \neg C_3) \wedge \ldots \wedge (\neg C_2 \vee \neg C_n) \wedge (\neg C_2 \vee P) \wedge (\neg C_{n-1} \vee \neg C_n) \wedge \ldots \wedge (\neg C_{n-1} \vee P) \wedge (\neg C_2 \vee P)$

O modelo de otimização proposto neste trabalho usa inicialmente o mapeamento de modelo de características para SAT [3]. O SAT é um problema da lógica proposicional. Seu objetivo é determinar um conjunto de valores (verdadeiro ou falso) para as variáveis proposicionais que faça uma dada expressão na forma normal conjuntiva (FNC) satisfazível - ou mostre que nenhum conjunto de valores a satisfaz [5]. Para a modelagem do problema de otimização SAT é necessário representar o modelo de características como um conjunto das relações traduzidas seguindo as regras da Tabela 1. Além das regras descritas, uma restrição adicional é a inclusão da raiz em todas as clausulas (Figura 1). Por exemplo, para a resolução do problema como um modelo de otimização SAT, decompomos o modelo de características apresentado na Figura 2 na expressão abaixo.

$$E(C) = ((\neg Automovel \vee Volante) \wedge (\neg Volante \vee Automovel)) \wedge ((\neg Automovel \vee Motor) \wedge (\neg Motor \vee Automovel)) \wedge ((\neg Automovel \vee Bateria) \wedge (\neg Bateria \vee Automovel)) \wedge ((\neg Som\_Automotivo \vee Automovel)) \wedge ((\neg Airbag \vee Automovel)) \wedge ((\neg Bateria \vee Tipo\_1 \vee Tipo\_2 \vee Tipo\_3) \wedge (\neg Tipo\_1 \vee Bateria) \wedge (\neg Tipo\_2 \vee Bateria) \wedge (\neg Tipo\_3 \vee Bateria) \wedge ('Tipo\_1 \vee \neg Tipo\_2) \wedge (\neg Tipo\_2 \vee \neg Tipo\_3)) \wedge ((\neg Tipo\_2 \vee Frontal) \wedge (\neg Frontal \vee Tipo\_2)) \wedge ((\neg Airbag \vee Frontal \vee Lateral) \wedge (\neg Frontal \vee Airbag) \wedge (\neg Lateral \vee Airbag))$$

Após a decomposição do modelo em uma expressão na FNC, o problema é modelado como um problema de otimização MAX-SAT Ponderado. O problema MAX-SAT Ponderado é uma unidade do problema SAT. Ele atribui um peso a cada cláusula [9]. Denotaremos esses pesos por $G_{1..n}$, onde o objetivo é selecionar as características com um maior grau de importância para o cliente. Posteriormente, a analogia com o Problema da Mochila é realizada. No Problema da Mochila temos uma situação em que o objetivo é preencher uma mochila com o maior valor possível, dado objetos de diferentes pesos e valores, não ultrapassando a capacidade máxima da mochila [7]. Fazendo uma analogia ao problema abordado neste trabalho, o objetivo é obter uma configuração de produto composto por características que somam um maior grau de importância (valor) para o cliente e que não ultrapassem o orçamento disponível (capacidade máxima), baseando-se nos preços de desenvolvimento de cada uma das características.

3.2. Algoritmos de Enumeração Exaustiva

Por ser um problema combinatório, é possível encontrar a melhor configuração do produto por enumeração exaustiva. Ou seja, podem-se enumerar todas as soluções possíveis, comparando-as para identificar a melhor solução. A instância deste problema de decisão é uma expressão booleana escrita somente com operadores *AND*, *OR*, *NOT*, *variáveis* e *parênteses*. Para resolução, as variáveis recebem valores binários e as expressões são solucionadas através da representação por uma árvore, na qual as folhas são operandos e os nós internos são operadores. A enumeração de todos os produtos que possam ser extraídos da LPS analisada gera todas as sequências possíveis de dados.

Este trabalho implementou inicialmente o algoritmo de enumeração exaustiva com pré-processamento e backtracking. O algoritmo de pré-processamento tem como objetivo condensar o modelo de características analisado. Assim, características classificadas como mandatórias, características que possuem um grau de importância elevado para o cliente e características que excedem o orçamento disponível são pré-processadas e agrupadas na árvore (juntamente com a soma de seus preços de

desenvolvimento). A Figura 3 mostra um exemplo da execução do algoritmo de pré-processamento, para o modelo de características apresentado na Figura 2.

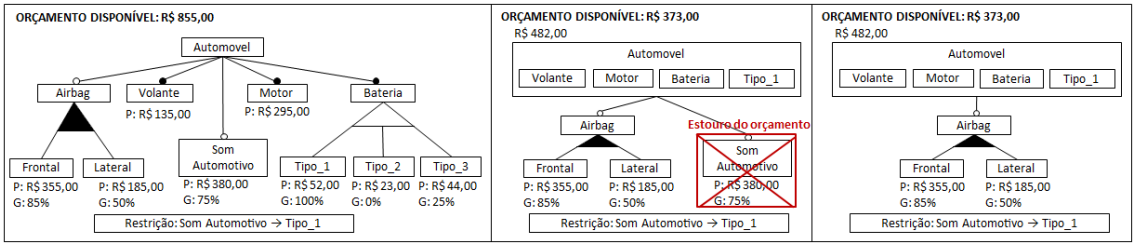


Figura 3 Pré-processamento do modelo de características

Backtracking é um algoritmo refinado da busca por enumeração exaustiva, no qual boa parte dos caminhos que não levam à solução desejada são eliminados sem serem explicitamente examinados [5]. Para a implementação do backtracking, o algoritmo analisa o preço e grau de importância de cada uma das características presentes no modelo. Um dos pontos é detectar através de restrições do problema as características e/ou grupos de características mortas (*dead features*). Ou seja, a detecção de características que por influência das restrições e do resultado parcial da configuração se tornam inutilizáveis, não podendo ser oferecidas para o cliente. A eficiência desta estratégia depende das possibilidades de limitar a busca (poda da árvore), eliminando os ramos que não levam à solução desejada.

Apesar de estes algoritmos apresentarem métodos exatos de resolução, conduzindo a soluções ótimas globais. A eficiência do algoritmo de enumeração exaustiva com pré-processamento e backtracking pode ser comprometida por problemas como: (i) um número irrelevante de características mandatórias presentes no modelo e (ii) restrições do modelo não favorecem a realização de podas. Assim, sua formulação e/ou resolução exata pode levar a uma complexidade intratável por seu longo tempo de execução, dado que a quantidade de soluções possíveis para o problema cresce exponencialmente em função do número de características. Logo, optou-se por colocar em segundo plano a solução ótima e buscar algoritmos que forneçam boas soluções em tempo polinomial. Dessa forma, é necessário à implementação de um algoritmo não exato (heurística) para maximização de ganhos de eficiência e de custo. Como solução para o problema, utilizamos a heurística gulosa descrita a seguir.

3.3. Heurística de Solução Gulosa

Este trabalho propõe um algoritmo que usa heurística gulosa. A heurística gulosa é aplicada a problemas que necessitam obter um subconjunto que satisfaça algumas restrições. Qualquer subconjunto que satisfaça as restrições é chamado de solução viável [5]. O que queremos é uma solução viável que maximize a função objetivo descrita na Seção 2.2. Chamamos a solução viável que satisfaz a função objetivo de solução ótima. Construímos um algoritmo que trabalha em estágios, selecionando uma característica por vez e, em cada estágio, é tomada uma decisão considerando que uma característica particular é uma solução ótima, na esperança de que esta escolha leve até a solução ótima global. O algoritmo implementado consiste das etapas detalhadas na Tabela 2.

A Figura 4 mostra um exemplo da execução do algoritmo guloso, para o modelo de características apresentado na Figura 2, até que o orçamento disponível seja excedido. Apesar da complexidade deste algoritmo ser polinomial, nem sempre ele nos fornece o melhor resultado possível. Isso ocorre, por exemplo, durante a seleção de uma

característica com grau de interesse elevado para o cliente, mas que possui preço alto. A seleção desta característica pode impossibilitar a seleção de outras características. Neste caso, a combinação de duas ou mais características de preço menor poderia resultar em um produto de maior interesse para o cliente.

Tabela 2 Heurística gulosa

Algoritmo. Heurística Gulosa (Entrada: Modelo de Características em XML)
01. Pré-processamento do modelo de características
02. Verificar nível 2 da árvore de características
03. Se existir característica XOR
04. Ci = selecionar característica XOR com maior Gi
05. Se não
06. Se existir um grupo de características OR e nenhuma característica do grupo tenha sido selecionada
07. Ci = selecionar característica OR do grupo com maior Gi
08. Se não
09. Ci = selecionar característica com maior Gi
10. Se Ci é uma característica abstrata e todas as características filhas de Ci possuem P > O
11. Eliminar Ci e todas suas filhas
12. Se não
13. Se Ci é uma característica abstrata
14. Ci é configurada para o cliente
15. Se não
16. Se Pi ≤ O
17. Ci é configurada para o cliente e Pi é decrementado do O
18. Se Ci compõe um grupo de características XOR
19. Eliminar as demais características XOR que pertencem ao mesmo grupo de Ci
20. Características filhas de Ci (se houver) passam a pertencer ao nível 2 da árvore
21. Se não
22. Eliminar Ci e todas suas filhas (se houver)
23. Executar a linha 02 até que todas as características da árvore sejam verificadas

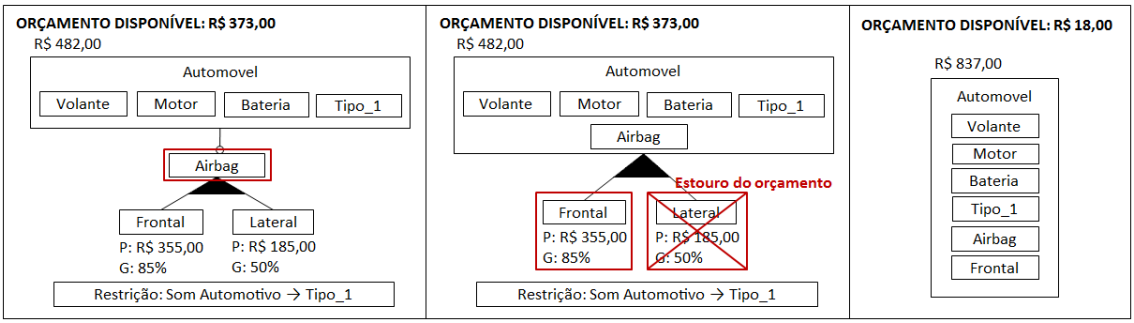
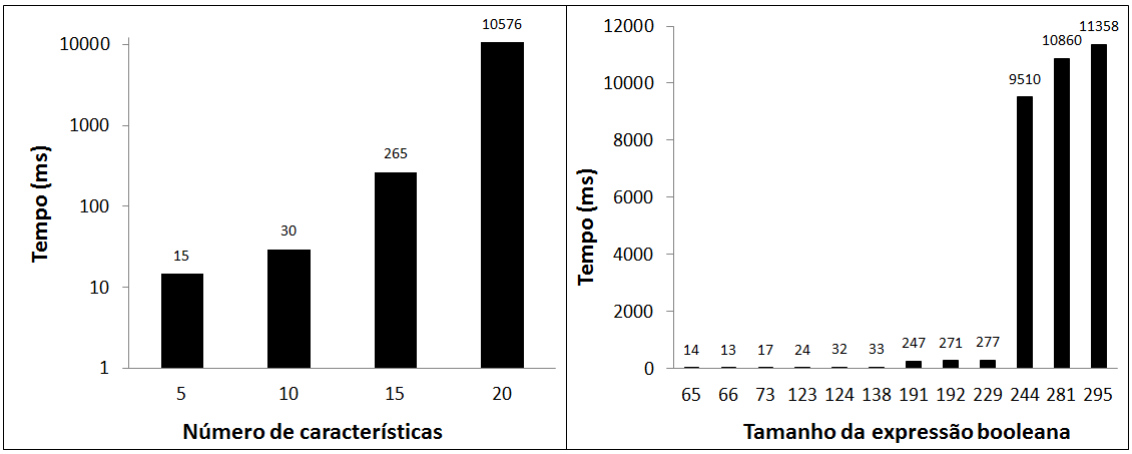


Figura 4 Exemplo da execução do algoritmo guloso

4. Experimentos Computacionais

Os experimentos foram realizados para 12 LPS que se diferenciam com relação ao número de características, classificação das características e restrições adicionais. Realizamos os experimentos agrupando as LPS em 4 grupos com 3 LPS por grupo. Variamos em 5, 10, 15 e 20 o número de características presentes nas LPS analisadas. Esta variação não excede 20 características, uma vez que a solução exata (por enumeração exaustiva) para grandes LPS é intratável devido a complexidade exponencial do problema. Os modelos de características foram obtidos do repositório SPLOT [8]. SPLOT é uma ferramenta Web de código fonte aberto que possui um repositório com centenas de modelos de características criados por usuários, além de modelos de características gerados automaticamente pela ferramenta. Os modelos de características na ferramenta podem ser exportados em formato XML e usados como entrada para os algoritmos implementados.

Verificamos o tempo médio utilizado por cada grupo durante a execução da enumeração exaustiva. A Figura 5a mostra que à medida que a quantidade de características aumenta, obtemos um crescimento exponencial do tempo de execução do algoritmo. Uma das características deste problema é que quanto maior o tamanho da expressão a ser solucionada, maior o tempo de processamento do algoritmo (Figura 5b). Neste caso o tempo independe da quantidade de produtos válidos gerados pela expressão. Isso ocorre porque o algoritmo de enumeração exaustiva testa todas as combinações possíveis, sejam elas válidas ou inválidas.



(a) Tempo médio para obter a solução ótima

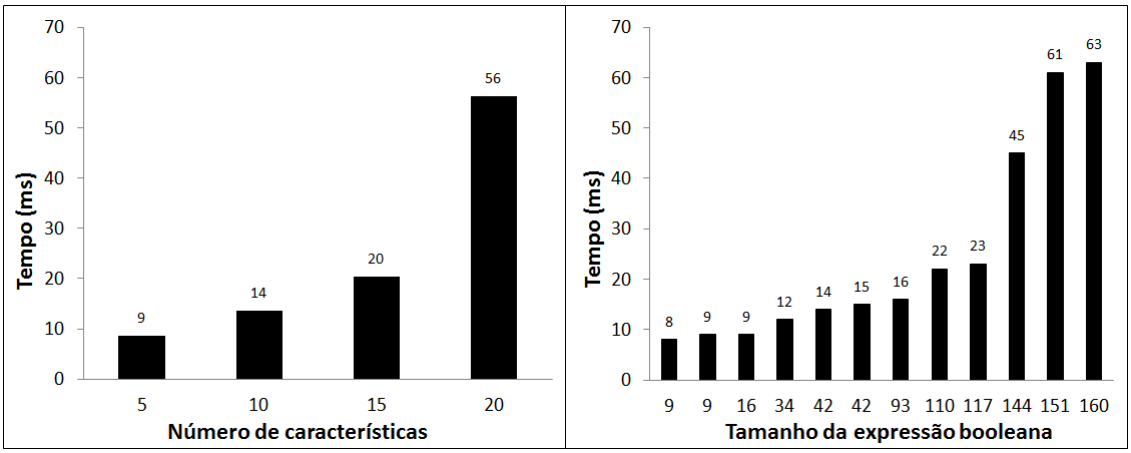
(b) Tempo gasto para obter a solução ótima em relação ao tamanho da expressão

Figura 5 Experimentação por enumeração exaustiva

Realizamos os mesmos testes e avaliações para o algoritmo de enumeração exaustiva com pré-processamento e backtracking. O objetivo é observar o quanto este algoritmo é mais eficiente comparado ao algoritmo anterior, que implementa apenas o algoritmo de enumeração exaustiva sem estas otimizações. A Figura 6b mostra que com a implementação da técnica de pré-processamento, o tamanho da expressão correspondente a cada uma das LPS analisadas serão reduzidas. Por exemplo, o tamanho da expressão para uma mesma LPS passa de 65 caracteres para 9 caracteres (Figura 5b e 6b). Adicionalmente, com a implementação da técnica backtracking o tempo de execução será minimizado e boa parte das soluções que não levam a solução ótima serão eliminadas, ou seja, a diminuição do número de soluções possíveis facilita a resolução (Figura 6a). Por exemplo, o tempo médio de execução somente com o algoritmo de enumeração exaustiva para um modelo de características com 20 características é 10576 ms, enquanto que, para o mesmo algoritmo com a implementação das técnicas de pré-processamento e backtracking o tempo médio de execução é 56 ms. Vale a pena ressaltar, que a implementação destas técnicas garantem a obtenção da solução ótima.

Apesar dos experimentos computacionais com o algoritmo de enumeração exaustiva com pré-processamento e backtracking demonstrarem um ganho significativo, o crescimento ainda é exponencial em função do tamanho do modelo de características. Portanto, foi avaliada a implementação da heurística gulosa para resolução do problema. Este algoritmo possui crescimento polinomial de tempo em função do tamanho da entrada. O tempo de processamento utilizado pelo algoritmo, para um modelo com 20 características, é menor que 10 ms. A Figura 7 mostra a taxa de acertos do algoritmo guloso durante a configuração de 10 produtos em uma LPS com 20 características, a partir de parâmetros (orçamento e grau de importância das características) fornecidos por 10 clientes com pretensões distintas. A taxa de acerto do algoritmo foi calculada

através da execução do algoritmo de enumeração exaustiva, pois ele nos fornece a solução ótima para o problema. De 10 produtos gerados, somente 3 não fornecem como resultado a solução ótima, ou seja, 3 configurações são compostas por características que não fazem parte da solução ótima e/ou deixam de compor características que fazem parte da solução ótima. Assim, para os testes realizados, a taxa de acerto do algoritmo guloso é em média 92%. Entretanto, apesar de nem sempre o algoritmo guloso conduzir a soluções ótimas globais, o algoritmo encontra a solução ótima global na maioria dos casos. Isso ocorre devido ao padrão balanceado adotado pelos modelos de características em uma LPS, o que sugere a aplicabilidade deste algoritmo para resolução de problemas reais de grandes dimensões.



(a) Tempo médio para obter a solução ótima

(b) Tempo gasto para obter a solução ótima em relação ao tamanho da expressão

Figura 6 Experimentação por pré-processamento e backtracking

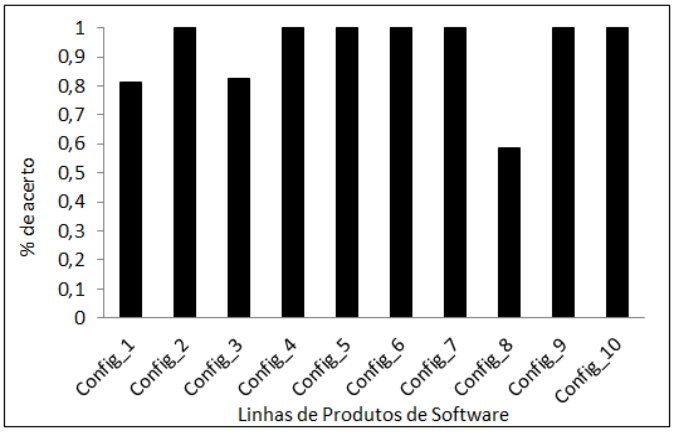


Figura 7 Taxa de acertos da heurística gulosa

5. Conclusão

Algoritmos de busca e otimização têm sido aplicados em diversas áreas da Engenharia de Software [6]. Neste trabalho, aplicamos estas técnicas para modelar o problema de configuração de produtos em uma LPS objetivando maximizar a satisfação do cliente. Dois tipos de algoritmos foram utilizados: enumeração exaustiva e solução gulosa. Os experimentos nos mostram que a enumeração exaustiva pode ser inviável, dado uma quantidade relativamente grande de características presentes em modelos reais. A utilização de algoritmos de pré-processamento e backtracking mostrou que o esforço necessário para encontrar a solução ótima tende a decrescer, devido ao número de caminhos que são eliminados, alcançando ganhos consideráveis em eficiência.

Por outro lado, mesmo utilizando pré-processamento e backtracking, o número de soluções viáveis para o problema continua a crescer exponencialmente em função do número de características. Assim, este trabalho propôs um algoritmo heurístico guloso. Este algoritmo se mostrou uma opção mais indicada para encontrar uma boa solução para o problema em significativamente menor tempo. A heurística proposta tem complexidade polinomial e apresenta uma taxa de 92% de acerto em relação à solução ótima, o que faz este modelo viável em aplicações práticas do problema.

Como trabalho futuro, pretende-se: (i) aplicar a heurística gulosa em LPS maiores, a fim de verificar o tempo necessário para execução, bem como a precisão dos resultados alcançados e (ii) considerar variáveis adicionais ao modelo proposto. Por exemplo, para maximizar os benefícios do ponto de vista da organização desenvolvedora, pretende-se incluir variáveis como a minimização do esforço de desenvolvimento e integração de características.

Agradecimentos

Este trabalho recebeu apoio financeiro da FAPEMIG, processos APQ-02376-11 e APQ-02532-12, e do CNPq processo 485235/2011-0.

Referências

[1] Batory, D. (2005) "Feature Models, Grammars, and Propositional Formulas", In Proc. of the International Software Product Line Conference (SPLC), pp. 7–20.

[2] Batory, D., Sarvela, J., Rauschmayer. (2004) "Scaling Step-wise Refinement", IEEE Transactions on Software Engineering, 30(6), pp. 355–371.

[3] Benavides, D. et al. (2006) "A First Step Towards a Framework for the Automated Analysis of Feature Models", In Managing Variability for SPL: Working With Variability Mechanisms.

[4] Boucher, Q. et al. (2010) "Introducing TVL, a Text-based Feature Modelling Language", In: Variability Modelling of Software-Intensive Systems (VaMoS).

[5] Cormen, T. et al. (2009) "Introduction to algorithms", Cambridge, EUA: Massachusetts Institute of Tecnology.

[6] Harman, M. (2007) "The Current State and Future of Search Based Software Engineering", In Proceedings of the Future of Soft. Eng. Workshop, Minneapolis.

[7] Martello, C. and Toth, P. (1990) "Knapsack Problems: Algorithms and Computer Implementations", John Wiley & Sons.

[8] Mendonça, M. et al. (2009) "S.P.L.O.T. – Software Product Lines Online Tools", Int'l Conf. on OO Programing, Systems, Languages, and Applications (OOPSLA).

[9] Pipatsrisawat K. et al. (2008) "Solving Weighted Max-SAT Problems in a Reduced Search Space: A Performance Analysis", Journal on Satisfiability Boolean Modeling and Computation.

[10] Pohl, K.; Bockle, G.; Linden, F. J. van der. (2005) "Software Product Line Engineering: Foundations, Principles and Techniques", Springer.

[11] Product Line Hall of Fame. Disponível em: http://splc.net/fame.html

[12] Sochos P. et al. (2004) "Feature-Oriented Development of Software Product Lines: Mapping Feature Models to the Architecture", OO and Internet-based Technology.

[13] Schobbens, P. Y. et al. (2006) "Feature Diagrams: A Survey and a Formal Semantics, In International Requirements Engineering Conference (RE).

[14] Sharp, D. C. (1998) "Reducing Avionics Software Cost Through Component Based Product Line Development". Software Technology Conference.