Configuração de Produtos em Linha de Produtos de Software

Markos Almeida, Johnatan Oliveira, Eduardo Figueiredo[1]

[1]Departamento de Ciência da Computação
da Universidade Federal de Minas Gerais
(DCC/UFMG) – Belo Horizonte – MG – Brasil

markosviggiato@gmail.com, {johnatan.si, figueiredo}@dcc.ufmg.br

Abstract. *A software product line (SPL) is formed by a set of software systems that share common characteristics. In SPL, a characteristic represents a desirable feature of interest in the system. The feature-model is used in LPS to document and configure the product. The main problem of this process is to derive the product configuration mode that meets the customer's requirements. To solve this problem, we developed an exact algorithm and a heuristic. The experiments show that the exact algorithm (brute force) is able to find the optimum configuration for the real problems of literature, when 20-60 features are evalueated. The difference between the exact algorithm and heuristic is 19,09%. However, for SPL greater than 50 features, the exact algorithm was unable to find the optimal solution. The heuristic, in turn, was able to identify an acceptable configuration within 30 seconds of execution.*

Resumo. *Uma linha de produto de software (LPS) é formada por um conjunto de sistemas de software que compartilham características em comum. Em LPS, uma característica (feature) representa uma funcionalidade de interesse desejável no sistema. O modelo de características é utilizado em LPS para documentar e configurar o produto. O principal problema deste processo é derivar a configuração do produto de modo que atenda aos requisitos do cliente. Para resolver este problema, desenvolvemos um algoritmo exato e uma heurística. Os experimentos mostram que o algoritmo exato (força bruta) é capaz de encontrar a configuração ótima para os problemas reais da literatura, sendo avaliado de 20 a 60 características. A diferença entre o algoritmo exato e a heurística é de 19,09%. Entretanto, para LPS maiores que 50 características, o algoritmo exato foi incapaz de encontrar a solução ótima. Já a heurística, foi capaz de identificar uma configuração aceitável em 30 segundos de execução.*

1. Introdução

Torna-se mais comum encontrar empresas que desenvolvem sistemas de informação que necessitam de desenvolvimento ágil, seguro e com qualidade. Nesse contexto está presente a reutilização de componentes de software. A LPS é uma das técnicas fundamentais nesse processo. Uma LPS pode ser definida como um conjunto de sistemas especificados, modelados e implementados em termos de funcionalidades comuns e variáveis [Apel et al. 2013]. As funcionalidades podem ser apresentadas como características de domínio que satisfaçam requisitos específicos de um segmento de mercado [Pohl et al. 2005].

A reutilização no contexto de LPS é definida como as características comuns de um determinado produto e cliente. Assim, considerando um conjunto de sistemas pertencentes a um mesmo domínio de negócio, a aplicação da técnica de LPS poderia viabilizar a reutilização de código [Pohl et al. 2005]. Isso proporcionaria vantagens como redução nos custos de produção e confiabilidade ao produto, uma vez que esses componentes já foram desenvolvidos e testados em outras ocasiões [Chitchyan et al. 2015].

Com a adoção da LPS, os sistemas possuem uma arquitetura comum e podem se beneficiar do reuso de código. Outra influência da LPS é no processo de desenvolvimento. Nessa fase é definido como serão executadas as etapas de análise de requisitos, implementação e outras. Contudo, existe um problema atrelado à configuração (ou customização) de produtos em uma LPS. O desafio de configurar uma LPS é alcançar o máximo possível a satisfação de um cliente. Essa configuração possui alta complexidade, dependendo fortemente da quantidade de características da LPS, além das regras que definem a configuração dos produtos [Batory 2005, Mendonca et al. 2009].

Diversos trabalhos foram propostos para fornecer uma configuração válida de uma LPS. Pode-se citar o estudo realizado por [White 2008], que propõe técnicas baseadas em programação por restrições para que se encontre uma configuração boa ou ótima da LPS. Outro trabalho foi desenvolvido por [Khalfaoui et al. 2015], que aborda o problema por meio das transformações de modelo, com a intenção de se reduzir o espaço de busca de configurações válidas de LPS. Nosso trabalho propõe a otimização das configurações por meio da relação benefício/custo.

Neste artigo, nós propomos uma solução para o problema de configuração de produtos, por meio de dois algoritmos: I) Algoritmo exato (força bruta) testa todas as combinações possíveis para configurar uma LPS da melhor forma possível e, II) Heurística por meio do paradigma guloso. A proposta implementada leva em consideração as seguintes informações: I) o tipo das características (mandatória, opcional ou exlusiva), II) custo de cada característica, III) orçamento do cliente e IV) satisfação do cliente.

Para avaliar a nossa proposta, conduzimos um experimento simulando de 20 a 60 características de LPS. Através dos resultados, nós concluimos que é possível configurar uma LPS de modo a garantir a satisfação do cliente, sem ultrapassar o orçamento inicial, além de alcançar a qualidade do produto. Por meio da avaliação, percebemos que os algoritmos implementados são capazes de identificar as melhores configurações factíveis de produtos em LPS.

O restante do artigo está organizado da seguinte forma. A seção 2 apresenta o referencial teórico que fornece base para compreensão geral do artigo. A seção 3 apresenta os algoritmos desenvolvidos para solucionar o problema em configurar produtos em LPS. A seção 4 apresenta uma avaliação dos algoritmos propostos. A seção 5 descreve os resultados obtidos através da avaliação, por meio de um estudo empírico, da configuração de LPS proposta. A seção 6 apresenta as ameaças para a validade do nosso estudo. A seção 7 discute os trabalhos relacionados. Finalmente, a seção 8 conclui o artigo com uma discussão e apresenta os trabalhos futuros.

2. Referencial Teórico

Essa seção descreve os conceitos que norteiam o artigo e que são fundamentais para a compreensão do mesmo. Na seção 2.1 tem-se uma apresentação da definição de LPS e na seção 2.2 é mostrado um modelo por meio do qual pode-se representar uma LPS, o modelo de características.

2.1. Linha de Produto de Software

Uma LPS é uma coleção de produtos que compartilham um conjunto comum de características que atendam às necessidades específicas de uma determinada área [Donohoe 2000]. A LPS lida com o desenvolvimento, gerenciamento de software e abrange as áreas de engenharia de domínio e engenharia de aplicação. Na primeira, a variabilidade dos componentes soluções de uma LPS é explicitamente capturada por modelos de variabilidade [Pohl et al. 2005]. Na área de engenharia de aplicação, produtos customizados derivam da LPS por meio do uso desses modelos e da implatação de componentes soluções necessários.

A LPS começou a ser introduzida no ambiente de desenvolvimento de software recentemente e se mostrou um dos avanços mais promissores nessa área. Vista como uma técnica eficiente no contexto da reusabilidade, ela foca em fornecer produtos customizados a custos razoáveis a partir de componentes reusáveis. Dentre as motivações para se utilizar a LPS durante o desenvolvimento de software, pode-se citar a redução do custo de desenvolvimento e a melhoria da qualidade, de acordo com [Pohl et al. 2005].

Um diferencial técnico que a LPS apresenta é o uso de características para distinguir seus membros.Essa característica representa um incremento na funcionalidade do programa. Um membro particular da LPS é definido por uma única combinação de características. O conjunto de todas as combinações define os membros da LPS [Batory 2005]. É importante destacar que, apesar da LPS ser utilizada no contexto de reusabilidade, pode haver variação do projeto de um software para outro, ou seja, pode-se ter a necessidade de um produto customizado.

A customização e personalização de softwares são ainda bastante limitadas para muitas aplicações, além de serem baseadas em diferentes técnicas que não são bem integradas entre si [Rabiser et al. 2009]. Customizar aplicações já existentes não é uma tarefa fácil para usuários finais. Foi demonstrado que uma LPS fornece uma abordagem razoável para se realizar a customização de sistemas de software complexos [Rabiser et al. 2009].

2.2. Modelo de características

Pode-se representar uma LPS por meio de um modelo de características (*feature-model*), que define as características do sistema e suas restrições de uso. Atualmente, a metodologia mais utilizada organiza as características em uma árvore, que é usada para especificar os membros da LPS [Batory 2005].

A representação do modelo de características através da árvore é composta por uma raiz, que indica o tipo de produto que é gerado pela LPS em questão. Além disso, a árvore possui outros nós, derivados da raiz, que representam cada uma das características da LPS. A literatura apresenta 4 tipos de características que podem estar presentes no modelo, a saber:

- **Mandatórias**: Se uma característica filha é mandatória, esta deve ser incluída em todos os produtos em que a característica pai aparecer;
- **Opcionais**: Se uma característica filha é definida como opcional, esta pode opcionalmente ser incluída em produtos em que a característica pai aparecer;
- **Alternativas inclusivas (OR):** Um conjunto de características é definido como alternativo se apenas uma das características pode ser selecionada quando a característica pai é parte do produto;
- **Alternativas exclusivas (XOR):** Uma relação entre um conjunto de características é definido como uma relação ou se uma ou mais características podem ser incluídas quando a característica pai é parte do produto.

Pode-se ver esses quatro tipos de características apresentados no exemplo completo do modelo de características do aplicativo de Smartphone na Figura 1.

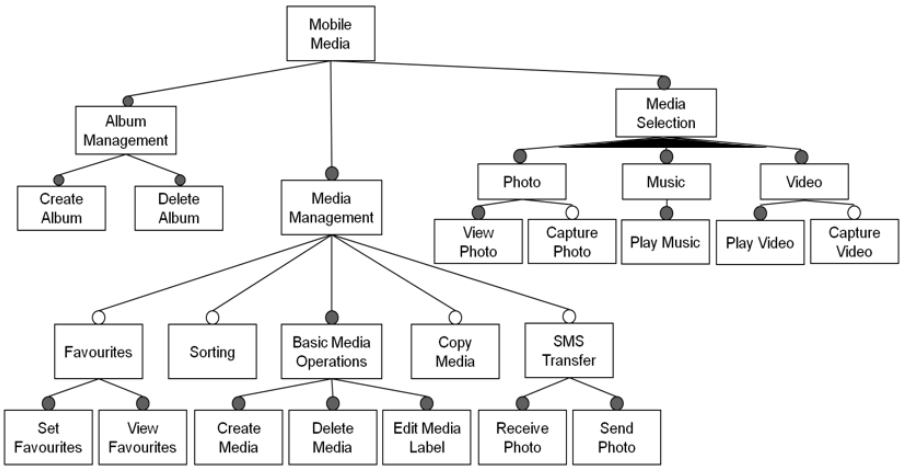

Figura 1. Modelo de característica representativo

3. A Abordagem Proposta

Nesta seção, apresentamos nossa abordagem proposta para a configuração da LPS, sendo composto por dois algoritmos: I. Algoritmo Extato (força bruta), II. Heuristica (algoritmo guloso). Na seção 3.1, apresentamos a modelagem matemática do problema como sendo do tipo do problema da mochila. Já na seção 3.2, são apresentados os algoritmos de força bruta e guloso.

3.1. Modelagem do Problema

Dadas as condições de configuração de uma LPS, em que tem-se que tomar decisões quanto a inclusão ou não de certas características dependendo das restrições, verificou-se a semelhança entre esse problema e o problema da mochila, que se mostrou adequado para essa situação, visto que uma característica corresponde a um item a ser inserido na mochila. O problema da mochila tem sido largamente estudado pelo fato dele poder ser utilizado para modelar problemas inseridos em diversos ambientes, como carregamento de carga e orçamento de capital na área industrial [Xiong and Ning 2014].

No problema da mochila, tem-se um conjuntos de N itens disponíveis para serem escolhidos, de forma que a escolha de um subconjunto daqueles deve maximizar

os benefícios sem ultrapassar a capacidade máxima da mochila [Pisinger 1995]. Para a solução do problema de configuração de LPS, a função objetivo a ser maximizada é $f(n) = \sum_{i=1}^{n} B_i C_i$, onde $C_i \in$ 0,1 representa se determinada característica i foi incluida ou não. Sujeito à $\sum_{i=1}^{n} P_i C_i \leq O$ (é limitado pelo orçamento do cliente / capacidade da mochila). O algoritmo exato foi implementado de acordo com a modelagem proposta por [McGrail 2004] que consiste no problema da mochila.

Com essa abordagem, a modelagem inicialmente é através de um mapeamento do modelo de características da LPS para SAT (problema da satisfatibilidade booleana), isto é, o modelo de características é traduzido para uma expressão lógica seguindo um conjunto de regras em que deve-se determinar se certa característica pertence ou não ao produto. Por fim, é realizado uma analogia do problema com o Problema da Mochila, onde se busca uma configuração de produto que ofereça o maior benefício ao cliente (valor dos elementos da mochila) limitado superiormente pelo orçamento (capacidade da mochila) considerando-se o custo de cada característica da LPS.

3.2. Algoritmos

Foram implementadas duas soluções distintas para o problema apresentado neste artigo. A finalidade de duas implementações distintas consiste em compará-las, para identificar a melhor solução. A primeira solução corresponde ao algoritmo no paradigma de força bruta que encontra uma solução ótima para o problema, sendo oriundo da literatura e proposto por [Pan and Martín-Vide 2005]; a segunda solução é uma heurística gulosa proposta pelos autores, sendo inspirada em outra heurística da literatura. Ambos os algoritmos foram adaptados da literatura para solucionar outro problema comum que é o problema da mochila [Jaszkiewicz 2002].

O algoritmo exato é capaz de encontrar a solução ótima para o problema usando o paradigma de força bruta. Dado que este algoritmo gera todas as possíveis configurações da LPS solicitada pelo cliente, de forma a maximizar a sua satisfação. O algoritmo faz uso da abordagem *Backtracking*, sendo responsável por analisar o custo e o benefício de cada característica do modelo. A fim de verificar a existência de características que não podem ser incluídas no produto final.

O algoritmo força bruta recebe como entrada: as características para compor a LPS, o orçamento do cliente, o benefício associado a cada característica e o custo associado a cada característica. O algoritmo, então, aloca e manipula uma matriz que contém todas as possíveis combinações n características a fim de determinar uma combinação que gera maior satisfação do cliente. Se denotarmos por T(n) o consumo de tempo no pior caso podemos dizer que:

$$T(0) = 1$$

Seja a equação de recorrência:

$$T(n) = T(n–1) + T(n–1) + \theta(1)$$

A solução desta recorrência tem a forma $2^0 + 2^1 + ... + 2^n$. Assim, $T(n) = \theta(2^n)$.

A heurística gulosa proposta nesse trabalho, por sua vez, é fruto de pesquisas bibliográficas do problema da mochila. A proposta utilizada para resolver o problema da mochila foi adaptada e utilizada para solucionar o problema de configurar os produtos em

LPS. Este algoritmo segue basicamente duas regras importantes. A primeira consiste em incluir na solução todas as características mandatórias. Se o saldo do cliente for menor do que o custo das características mandatórias, o algoritmo retorna que não existe solução viável neste caso.

A segunda regra ocorre se as características mandatórias tiverem sido incluídas na solução. O algoritmo, então, tenta incluir as características opcionais e XOR. Entretanto, antes da inclusão, o algoritmo verifica se existe um caminho na árvore até a característica. Caso não exista, as características que compõem esse caminho são adicionadas. Por fim, verifica-se novamente o orçamento, se for possível adicionar o caminho a característica será inserida, caso contrário, ela não será incluída.

O algoritmo guloso recebe como entrada: às características para compor a LPS, o orçamento do cliente, o benefício associado a cada característica e o custo associado a cada característica. Após a inserção das características mandatórias, o algoritmo utiliza as características restantes e divide o benefício pelo custo, como apresentado na equação: $\frac{B_i}{C_i}$

Seja n a quantidade de características de uma LPS dada como entrada. Assim, a complexidade de tempo da heurística implementada é: $O(n)$ para inclusão de todas as características mandatórias. Entretanto, o algoritmo busca as características mandatórias (obrigatórias) e depois busca as opcionais, efetuando o cálculo $\frac{B_i}{C_i}$ para localizar as melhores características opcionais com relação ao custo e benefício. Assim, no total, a complexidade da heurísticas é $O(n^2)$

4. Configuração da Avaliação

Para avaliar os algoritmos implementados, nós nos inspiramos em características apresentadas na literatura [Mendonca et al. 2009] e utilizamos de 20 a 60 características. Para executar este experimento, o seguinte cenário foi criado:

Entrada: de 20 à 60 características, com características mandatórias.

Limitação: o orçamento máximo do cliente para montar a LPS: R$ 600,00 reais

Os experimentos foram executados pelos autores deste artigo, em computadores distintos, porém com a mesma configuração. O ambiente de execução é apresentado a seguir:

Arquitetura dos computadores: Intel Pentium IV 2 GHz (clock real de 2000.777) Single, cache 512 KB, 2 GB de RAM.

Sistema Operacional: Linux 14.10, 32 bits, kernel 3.16.

Cada algoritmo foi executado por 5 vezes para garantir resultados fidedignos, no término de cada execução a média de tempo foi calculada. O tempo de execução de cada algoritmo, foi recuperado através do tempo gasto completar a sua execução, descrita pelo autor [Keppel 1996]. Cada algoritmo foi implementado na linguagem de programação Java pelos autores deste artigo.

5. Resultados

A partir dos critérios discutidos na seção 4, foram obtidos os resultados de tempo de execução de cada algoritmo. Os resultados são apresentados na Tabela 1. Sendo possível observar que a heurística proposta é capaz de encontrar configurações aceitáveis em se-

gundos para 50 características. Já o algoritmo de força bruta, consome além de mais processamento, aproximadamente 4horas para encontrar uma solução para 50 características.

Tabela 1. Tempo de execução/entradas distintas

Características	Força Bruta	Heurística
20	40 ’s	0,200 microsegundos
30	3 minutos	0,33 ’s
40	21 minutos	0,897’s
50	4 horas	1,541’s
60	-	2,53’s

Não basta avaliar apenas o tempo de execução de cada algoritmo. É imprescindível a avaliação da qualidade das respostas geradas por cada algoritmo. Essa análise de qualidade está descrita na Tabela 2. Sendo organizado de forma hipotética a relação custo benefício.

Tabela 2. Qualidade das respostas

Características	Força Bruta	Heurística
20	619,0/390,0	505,0/363,0
30	785,0/397,0	785,0/397,0
40	829,0/392,0	829,0/392,0
50	879,0/380,0	698,0/373,0
60	-	883,0/355,0

Analisando a Tabela 2, é possível identificar que os resultados alcançados pela heurística implementada possuem pequena discrepância em relação ao algoritmo exato para as entradas 20 e 50, entretanto, essa discrepância não apresenta diferenças significativas, tendo a visão de que, apesar da heurística implementada nem sempre alcançar a solução ótima, esta é capaz de solucionar o problema em menos tempo de execução em relação ao algoritmo exato.

6. Ameaças à Validade

A nossa pesquisa foi baseada em trabalhos relacionados para apoiar o desenvolvimento dos algoritmos propostos. Em relação a avaliação dos algoritmos, desenvolvemos um cuidadoso estudo empírico para avaliar a sua eficiência à respeito da configuração de produtos em LPS. Entretanto, algumas ameaças à validade podem afetar os nossos resultados. As principais ameaças e seus respectivos tratamentos são discutidos abaixo com base nas categorias propostas por Wohlin et al. [Wohlin et al. 2012].

Validade de Construção. Antes de executar os algoritmos desenvolvidos, realizamos uma seleção manual das instâncias na literatura. No entanto, algumas ameaças podem afetar a seleção correta dessas instâncias, esses fatores humanos podem ter descartado instâncias relevantes para o estudo. Entretanto, selecionamos as principais instâncias citadas na literatura, como foi descrito na seção 4.

Validade Interna. Neste quesito pode ser considerada como uma ameaça o risco de generalizar a interpretação dos resultados, visto que a solução ótima em alguns casos não

foi identificada pelo algoritmo exato. Isso implica na perda do processo comparativo entre o algoritmo exato e a heurística. Contudo, a variação de diferença entre os algoritmos até o limite testado, foi de aproximadamente 19,09%.

Validade de Conclusão. A amostra pequena de instâncias, poderia fornecer conclusões impropias para este estudo. Assim como a falta de homogeneidade da amostra selecionada. Dessa forma, os resultados do estudo podem ser considerados indícios, e não resultados totalmente conclusivos.

Validade Externa. Neste aspecto é importante ressaltar que o estudo foi realizado em ambiente acadêmico, com instâncias fictícias. E, por isso, pode ser considerado uma ameaça à validade devido ao fato de que talvez o resultado alcançado poderia ser diferente caso o estudo tivesse sido aplicado em ambiente profissional ou com instâncias reais mais robustas do que foram avaliadas neste estudo. Porém, nós avaliamos várias instâncias, em computadores distintos e com baterias de testes contínuos. As instâncias avaliadas foram extraídas de pesquisas a partir da literatura.

7. Trabalhos Relacionados

Muitas abordagens têm sido propostas na literatura para a configuração da LPS. [White 2008], por exemplo, propõe o uso de técnicas de programação baseadas em restrições para desenvolver configurações variantes de LPS ótimas ou boas. Uma abordagem utilizada é a configuração automatizada de LPS sujeita a recursos limitados. Neste caso, foram desenvolvidas técnicas de configuração baseadas no problema da mochila cujas entradas são: I) os requerimentos da LPS e II) os recursos disponíveis para a configuração. A saída é a configuração ótima que se encaixa nos limites dos recursos.

Outro trabalho relacionado à configuração de LPS por ser verificado em [Khalfaoui et al. 2015]. Neste estudo, a abordagem por meio das transformações de modelo é utilizada com a intenção de se reduzir o espaço de busca relacionado às possíveis configurações de LPS, visto que, a partir do modelo de características, muitas configurações são possíveis. Um *framework* automático baseado nessa abordagem foi desenvolvido para gerar configurações válidas de LPS. Ele explora ascendentemente o diagrama de características, para cada nó uma configuração parcial é feita. No final, aquelas estruturas que violam as restrições da árvore são removidas.

8. Conclusão

Várias pesquisas sobre ferramentas automatizadas para criar e editar o modelo de características tem sido propostas na literatura. Entretanto, poucas ferramentas focam no processo de automatização para configurar o produto em LPS. Para ajudar a resolver este problema, este artigo apresenta um estudo sobre o processo de configuração de produtos em LPS. Nós modelamos o problema para encontrar a solução ótima, maximizando a satisfação do cliente como um problema de otimização.

Com a modelagem do problema, foi possível propor dois algoritmos: I) algoritmo exato que usa o paradgima de força bruta, encontrando a solução ótima para a configuração do produto e, II) algoritmo guloso com a heurística baseada no problema da mochila, que é capaz de encontrar soluções satisfatórias em pequeno espaço de tempo.

Nós avaliamos os algoritmos desenvolvidos com instâncias apresentadas na literatura. Essas instâncias ocilam entre 20 e 60 características. Verificamos por meio de estudos empíricos através de um experimento que o diferencial entre a solução ótima (algoritmo exato) e heurística (algoritmo guloso) é no máximo de 19,09%. Portanto, podemos concluir que a solução proposta pelos algoritmos são factiveis para resolver o problema da configuração de produtos em LPS.

Como trabalhos futuros, recomendamos o aperfeiçoamento dos nossos algoritmos exatos e heurísticas para adicionar outras características não funcionais no produto final. Adicionando as configurações necessárias para auxiliar os mantenedores da LPS no processo de manutenção. Visto que a proposta atual aborda a configuração inicial da LPS. Além disso, para simular a situação prática na medida do possível, os trabalhos futuros devem abordar estudos de caso na indústria, a fim para garantir que as instâncias realistas de requisitos estão sendo gerados.

Referências

Apel, S., Batory, D., Kästner, C., and Saake, G. (2013). *Feature-oriented software product lines: concepts and implementation*. Springer Science & Business Media.

Batory, D. (2005). Feature models, grammars, and propositional formulas. In *Proceedings of the 9th International Conference on Software Product Lines*, SPLC'05, pages 7–20, Berlin, Heidelberg. Springer-Verlag.

Chitchyan, R., Noppen, J., and Groher, I. (2015). What can software engineering do for sustainability: Case of software product lines. In *Proceedings of the Fifth International Workshop on Product LinE Approaches in Software Engineering*, PLEASE '15, pages 11–14, Piscataway, NJ, USA. IEEE Press.

Donohoe, P. (2000). *Software Product Line: Experience and Research Directions*. Springer Science+Business Media - LCC, New York, USA.

Jaszkiewicz, A. (2002). On the performance of multiple-objective genetic local search on the 0/1 knapsack problem - a comparative experiment. *IEEE Transactions on Evolutionary Computation*, 6(4):402–412.

Keppel, D. (1996). *Runtime Code Generation*. PhD thesis. AAI9637972.

Khalfaoui, K., Kerkouche, E., Chaoui, A., and Foudil, C. (2015). Automatic generation of SPL structurally valid products: An approach based on progressive composition of partial configurations. *2015 6th International Conference on Information and Communication Systems, ICICS 2015*, pages 25–31.

McGrail, Robert W, T. B. (2004). A grading dilemma or the abyss between sorting and the knapsack problem. *J. Comput. Sci. Coll.*, 19(5):97–107.

Mendonca, M., Branco, M., and Cowan, D. (2009). S.p.l.o.t.: Software product lines online tools. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, OOPSLA '09, pages 761–762, New York, NY, USA. ACM.

Pan, L. and Martín-Vide, C. (2005). Solving multidimensional 0–1 knapsack problem by p systems with input and active membranes. *Journal of Parallel and Distributed Computing*, 65(12):1578–1584.

Pisinger, D. (1995). Algorithms for Knapsack Problems. *Journal of Chemical Information and Modeling*, 53(9):1689–1699.

Pohl, K., Böckle, G., and Linden, F. J. v. d. (2005). *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Rabiser, R., Wolfinger, R., and Grünbacher, P. (2009). Three-level Customization of Software Products Using a Product Line Approach. *HICSS 2009: Proceedings of the 42nd Hawaii International Conference on System Sciences*, pages 1–10.

White, C. J. (2008). Optimizing the configuration of SPL variants.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media.

Xiong, X. and Ning, A. (2014). Cellular competitive decision algorithm for binary knapsack problem. *2014 10th International Conference on Natural Computation, ICNC 2014*, (1):503–507.