Eliminação Paralela de Termos Dominantes no Problema da Mochila

Flávio Regis de Arruda
Instituto de Matemática e Estatística
Universidade de São Paulo
regis@ime.usp.br

Alfredo Goldman
Instituto de Matemática e Estatística
Universidade de São Paulo
gold@ime.usp.br

Resumo

Um dos problemas mais conhecidos em otimização combinatória é o problema da mochila ilimitado. Devido a sua importância diversos autores buscaram formas eficientes de resolvê-lo. Por um lado, diversos estudos da paralelização deste problema foram feitos. Por outro lado, várias técnicas de eliminação de objetos também foram encontradas. Neste trabalho nós unimos as duas possibilidades apresentando algoritmos paralelos para a eliminação de objetos. Para validação dos algoritmos, os mesmos foram implementados em Java e executados em um cluster de computadores.

1. Introdução

Os problemas da mochila constituem uma classe de problemas dos mais estudados em otimização combinatória em virtude de suas diversas aplicações práticas bem como por seu interesse teórico. Além disto estes também aparecem como subproblemas em diversos outros problemas práticos e de otimização combinatória.

Toda a classe de problemas da mochila pertence a família dos problema $NP-$completos, porém usando algoritmos de programação dinâmica diversos problemas da classe podem ser solucionados em tempo pseudo-polinomial. O tempo de execução, nestes casos, está diretamente relacionado ao produto do número de objetos pelo tamanho da mochila. Nós estamos especialmente interessados pelo problema da mochila ilimitado, onde podem ser utilizados vários objetos do mesmo tipo. Dado um problema da mochila, o tamanho, ou capacidade da mochila, assim como os objetos e seus respectivos tamanhos (ou pesos) são dados. Observando o fato que certos objetos são mais interessantes do que outros é possível reduzir o espaço de busca. Formalmente a eliminação de objetos redundantes se faz através de relações de dominância [14].

Nos algoritmos para a resolução do problema da mochila ilimitado as relações de dominância desempenham um papel fundamental pois permitem uma redução substancial no tamanho da instância do problema a ser resolvido. No entanto, nessas soluções o benefício é tão grande que a maior parte do tempo de resolução do problema acaba sendo gasto na etapa de eliminação dos termos dominantes. Baseado nessa observação experimentamos paralelisar algoritmos de eliminação de termos dominantes para diversas instâncias e verificar se esta constitui uma boa estratégia para ganhar desempenho na solução de problemas da mochila.

Inicialmente faremos um resumo sucinto dos principais estudos para a resolução do problema da mochila em paralelo. Em seguida apresentaremos a definição formal do problema, e na seção seguinte algumas relações de dominância. Na seção 5 mostramos o algoritmo para eliminação paralela, e em seguida os resultados obtidos sobre diversos conjuntos de testes. Finalmente, na conclusão apresentamos uma discussão sobre os resultados obtidos e propostas para trabalhos futuros.

2. O Problema da Mochila

O problema da mochila (*Knapsack Problem* ou somente *KP*) pode ser enunciado de forma muito simples da seguinte forma:

Um viajante levará consigo apenas uma mochila para sua viagem. Sua mochila possui uma dada capacidade *e deve ser preenchida com alguns objetos que lhe sejam úteis durante a viagem. Cada objeto possui um* peso *e um dado* valor *para o viajante e este possui apenas uma unidade de cada objeto a ser escolhido. Quais objetos devem ser levados pelo viajante em sua mochila de forma a* maximizar o valor *da mochila?*

Seja w_j o peso do j-ésimo objeto e p_j seu valor. Se o objeto x_j aparece na mochila, então $x_j = 1$ caso contrário $x_j = 0$. Se denotarmos por c a capacidade da mochila, e por n a quantidade de objetos disponíveis para escolha e $F(c)$ como o maior valor obtido para a mochila de capacidade c usando os n objetos, então o problema pode ser formulado algebricamente como a seguir:

$$F(c) = max \sum_{j=1}^{n} p_j x_j \quad (p_j > 0) \tag{1}$$

$$sujeito\, a \sum_{j=1}^{n} w_j x_j \leq c \quad (w_j, c > 0) \tag{2}$$

Onde: $x_j = 0$ ou $x_j = 1$.

Esta versão do problema é conhecida como problema da mochila binária (*Binary KP* ou *0-1KP*).

Se permitirmos a x_j assumir qualquer valor inteiro maior que zero então temos a forma mais genérica da versão unidimensional do problema da mochila. Tal problema é conhecido como problema da mochila ilimitada (*Unbounded Knapsack Problem* ou *UKP*). Nesse artigo trataremos apenas desta última classe de problema da mochila.

Apesar do nome, o problema da mochila não é útil apenas na solução de problemas de empacotamento. Existem também aplicações na área da criptografia, engenharia naval, gerenciamento de projetos e finanças entre outras [17, 20].

Como um exemplo de aplicação do problema da mochila suponha que um gerente de uma empresa possua no seu orçamento c reais para investir em projetos dentro do seu departamento. Após uma pesquisa realizada por sua equipe, o gerente recebe um relatório com n diferentes projetos que trariam reduções de custo ou aumento de produtividade ao departamento como um todo. Associado a cada projeto j existe um retorno de p_j reais e um custo para sua realização de c_j reais. O gerente pode encontrar uma distribuição ótima de seu orçamento resolvendo um problema da mochila binária.

As duas técnicas mais utilizadas para resolver esse problemas são algoritmos de *Branch and Bound* e Programação Dinâmica, para uma leitura mais aprofundada sobre o assunto sugerimos o livro de Martello e Toth [17]. Como o problema é sabidamente $NP-$completo [9] começou-se a estudar formas de reduzir o tamanho do espaço de busca, e uma das maneiras mais importantes que foram encontradas é através da eliminação de termos redundantes, também conhecidos como termos dominados.

3. Trabalhos Relacionados

Existem vários estudos sobre a resolução do problema da mochila em paralelo [10]. Sendo que, algumas das soluções são adequadas apenas para o problema da mochila binário, principalmente usando os algoritmos de duas listas [13]. Outras resolvem o problema de forma aproximada. Nesta seção serão apresentados apenas os principais estudos sobre as resoluções exatas do problema da mochila ilimitado.

As soluções paralelas para o problema da mochila ilimitado são baseadas, principalmente, em técnicas que usam programação dinâmica [1, 2, 3, 4, 5, 6, 7, 12, 15, 16, 18, 20]. Sendo que quase sempre o modelo de computação utilizado é de grão fino, isto é, o volume de comunicação é próximo à quantidade de processamento. Apenas um dos estudos citados [18] apresenta resultados que podem ser utilizados diretamente em máquinas paralelas existentes atualmente.

Por outro lado, como em algoritmos baseados em programação dinâmica para a resolução do problema da mochila a complexidade depende diretamente do número de objetos ($O(mc)$), a eliminação de objetos redundantes pode melhorar significativamente o tempo de processamento. Este fato já é conhecido há bastante tempo [14]. Também é interessante ressaltar que várias formas de eliminação de objetos foram desenvolvidas recentemente [21, 22]. Entretanto não conhecemos estudos em que formas de eliminação de objetos tenham sido paralelizadas.

3.1 Nossa Contribuição

Neste trabalho nós propomos algoritmos paralelos para a eliminação de objetos para o problema da mochila ilimitado. Os algoritmos propostos são perfeitamente convenientes às máquinas paralelas atuais. Os mesmos foram implementados em Java, e testados em um *cluster* de PCs.

4. Relações de dominância

Para um dado objeto k pertencente a uma instância do problema da mochila, denomina-se o termo k de dominado (ou redundante) se o valor da solução ótima $F(c)$ não se altera quando k for removido do problema. Ou seja, para um termo redundante k pode-se fixar $x_k = 0$ no problema original.

Teorema 1: Sejam j e k dois objetos pertencentes a uma instância do problema da mochila ilimitada. Se:

$$w_j \leq w_k \text{ e } p_j \geq p_k \tag{3}$$

então o objeto k é dominado pelo objeto j.

Prova: Basta substituir a(s) ocorrência(s) de k na solução original pelo objeto j e percebe-se que a função $F(b)$ é maior ou igual a anterior (pois $p_j \geq p_k$). Ao mesmo tempo a soma dos pesos continua menor ou igual a capacidade c (dado que $w_j <= w_k$), portanto as quantidades associadas a objeto(s) do tipo k são nulas em uma solução ótima.

Definição 1: Sejam j e k dois objetos pertencentes a uma instância de um problema da mochila. Se j e k satisfazem (3) então diz-se que o objeto k é simplesmente dominado pelo objeto j. Esta relação de dominância foi proposta pela primeira vez por Gilmore and Gomory [11]. Veja um exemplo na Figura 1.

Algoritmo 1: Seja A o conjunto formado por objetos de um problema da mochila.

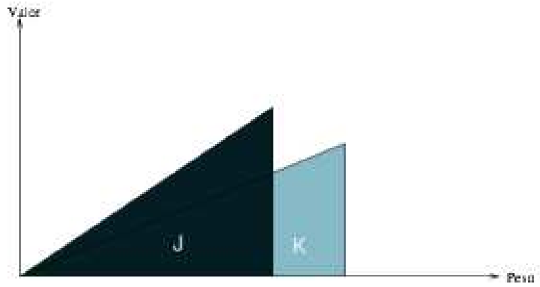

Figura 1. Relação de dominância simples

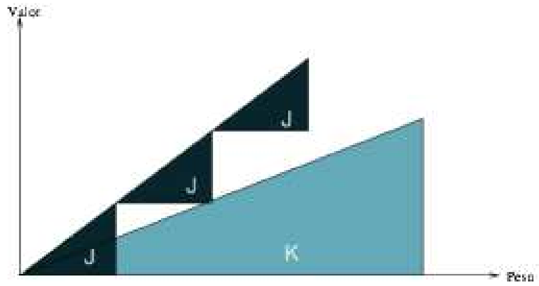

Figura 2. Relação de dominância múltipla

No algoritmo abaixo são removidos todos os elementos simplesmente dominados realizando-se $O(n^2)$ operações.

REMOVE DOMINADOS SIMPLES (A)
Ordene os elementos tal que $w_j < w_{j+1}$
e se $w_j = w_{j+1}$ então $p_j <= p_{j+1}$
Para $j \leftarrow 1$ até $|A|$ faça
  Para $k \leftarrow 1$ até $|A|$ faça
    Se (3) acontecer faça
      $A \leftarrow A \backslash \{k\}$

Teorema 2: Sejam j e k dois objetos pertencentes a uma instância do problema da mochila ilimitada. Se:

$$\lfloor w_k / w_j \rfloor p_j \geq p_k \quad (4)$$

então o objeto k é dominado pelo objeto j.
Prova: Seja $x_k = \alpha > 0$ e $x_j = \beta$ uma possível solução. Uma solução melhor pode ser construída fazendo $x_k = 0$ e $x_j = \beta + \lfloor w_k / w_j \rfloor \alpha$. De fato, a nova solução é possível já que a partir de (4) temos $\lfloor w_k / w_j \rfloor \alpha p_j \geq \alpha p_k$.

Definição 2: Sejam j e k dois objetos pertencentes a uma instância de um problema da mochila. Se j e k satisfazem (4) então diz-se que o objeto k é multiplamente dominado pelo objeto j. Veja um exemplo na Figura 2.

Algoritmo 2: Seja A o conjunto formado por objetos de um problema da mochila.
Então o algoritmo abaixo remove todos os elementos multiplamente dominados em $O(n^2)$ operações.

REMOVE DOMINADOS MÚLTIPLOS (A)
Ordene os elementos tal que $p_j / w_j < p_{j+1} / w_{j+1}$
e se $p_j / w_j = p_{j+1} / w_{j+1}$ então $w_j <= w_{j+1}$
Para $j \leftarrow 1$ até $|A|$ faça
  Para $k \leftarrow 1$ até $|A|$ faça
    Se (4) acontecer faça
      $A \leftarrow A \backslash \{k\}$

Essa relação de dominância foi introduzida por Martello e Toth e além de ser uma generalização do conceito de dominância simples é parte fundamental do um algoritmo de Branch and Bound [17] para resolução exata de *UKP* onde a quantidade de objetos é grande (dezenas ou centenas de milhares de objetos). Dudzinski [8] mostrou que diversas instâncias de *UKP* podem ser reduzidas para instâncias com menos de uma dúzia de elementos apenas pela utilização de uma redução baseada na eliminação de termos redundantes como pré-processamento. Como na maioria desses casos a maior parte do tempo computacional é gasto na redução do problema, este artigo visa estudar o comportamento de algoritmos de redução num ambiente paralelo de processamento.
Apesar das técnicas de eliminação de termos redundantes apresentarem bons resultados em grande parte das instâncias de problemas da mochila, existem instâncias do problema nas quais poucos ou nenhum elemento é eliminado. Tais instâncias são chamadas de instâncias fortemente correlatas e [17, 20] fornecem uma boa cobertura do assunto.

5. Eliminação paralela

Diversos estudos de algoritmos para a resolução do problema da mochila em paralelo são encontrados na literatura [10]. Também encontram-se publicações sobre relações de dominância em algoritmos seqüenciais [8, 17, 20]. Mas, apesar de uma grande busca bibliográfica, não tomamos ciência da existência de nenhum trabalho publicado sobre a implementação de algoritmos paralelos de eliminação de termos dominados, assim como os ganhos e problemas que surgem com a paralelização desse problema.

Como estávamos interessados em estudar o comportamento de versões paralelas dos algoritmos 1 e 2 de forma independente do algoritmo de resolução do problema da mochila, somente estes algoritmos (versões seqüenciais e paralela) foram implementados. Para os testes utilizamos um *cluster* de PCs (Pentium III 733MHZ-256Mb) com sistema operacional Linux numa rede Ethernet 100.

Foram gerados dois conjuntos de arquivos de testes, um para dominância simples e outro para dominância múltipla. Em cada um dos conjuntos haviam 20 arquivos onde variamos o número de objetos do problema (5000, 10000, 50000 e 10000) e a porcentagem de elementos dominados (10%, 30%, 50%, 70% e 90%). Para cada um dessas 40 instâncias do problema da mochila foram medidos os tempos de execução seqüencial e paralelo, este último utilizando p =2, 4 e 8 processadores. Na eliminação paralela, em cada uma das máquinas foram instalados serviços implementados em Java com métodos remotos (RMI) de eliminação de termos dominados utilizando os algoritmos seqüenciais 1 e 2.

Seja A o conjunto formado pelos objetos de uma instância do problema da mochila ilimitada (*UKP*), a eliminação foi feita da seguinte maneira:

ELIMINAÇÃO PARALELA(A)
Para $i = 1$ até $i = \lfloor |A|/p \rfloor$ faça em paralelo:
    Execute método remoto de eliminação no processador p_i sobre subdomínios disjuntos de A de tamanho $|A|/p$
    Execute o método local de eliminação para os termos restantes.

Do ponto de vista do desenvolvimento da aplicação, a escolha por desenvolver a aplicação utilizando-se JavaRMI mostrou vantajosa no momento de transformar uma implementação de objetos locais em objetos remotos. Outro ponto de destaque é o fato de permanecer transparente para as demais classes do sistema a localização (remota ou local) dos objetos contendo os métodos de eliminação utilizados.

6. Resultados obtidos

Os resultados obtidos estão sinteticamente representados pelos gráficos das figuras 3, 4, 5 e 6. Em cada figura temos no eixo x o número de processadores utilizados, e no eixo y o tempo total do experimento. Como foi verificado que os algoritmos paralelos utilizados são bem estáveis, isto é não houve variações de tempo entre diversas execuções, cada ponto nos gráficos corresponde ao tempo de uma execução.

Na Figura 3 podemos ver os resultados dos algoritmos seqüencial e paralelo para eliminação usando dominância simples. Como nessa figura as instâncias de teste tem poucos objetos (até 10000), mesmo quando existe a eliminação de 90% deles, não é possível obter ganho de tempo com relação ao algoritmo seqüencial. Isto pode ser explicado por que o custo das comunicações iniciais (dividir os objetos entre as máquinas) e final (concentrar os objetos após a primeira eliminação) é considerável. Também é interessante notar que para dois processadores este custo chega a ser maior do que o ganho proporcionado pela eliminação simultânea.

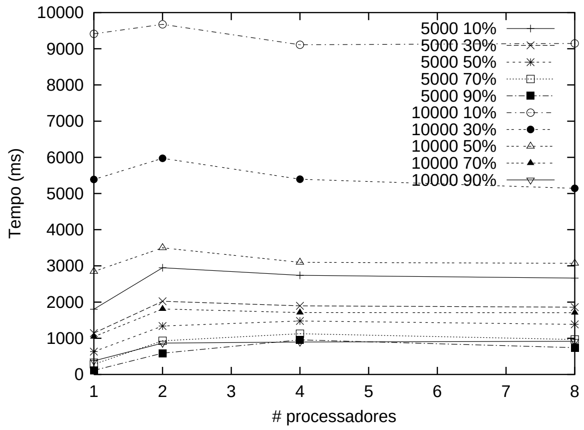

Figura 3. Gráfico da eliminação usando dominância simples

Na Figura 4 podemos ver que quando as instâncias são maiores, e o número de objetos eliminados é maior do que 10% é possível obter um pequeno ganho de desempenho, mesmo para dois processadores.

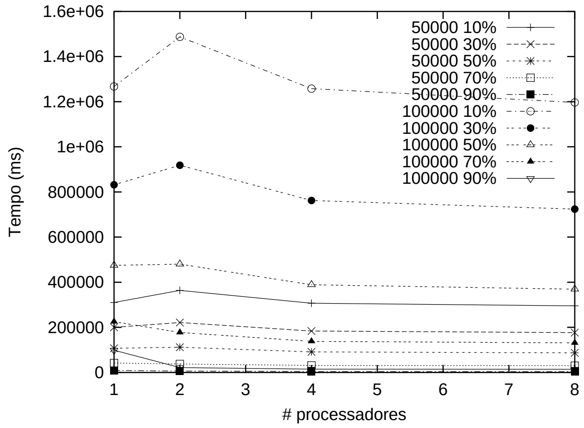

Figura 4. Gráfico da eliminação usando dominância simples

Nas figuras 5 e 6 temos os algoritmos com eliminação por dominância múltipla. Nestes casos o uso de até quatro processadores apresenta melhoras evidentes no tempo de processamento, mesmo para instâncias pequenas e um número reduzido de elementos dominados. Pode-se notar que para alguns exemplos, na figura 5, foi possível obter uma aceleração superlinear (ex: 10000 objetos com 90% de dominados). Isto se explica pois em alguns testes o número de elementos eliminados na última fase é bem reduzido, isto é, quase todos os elementos são eliminados em paralelo em tempo proporcional a $(n/2)^2$.

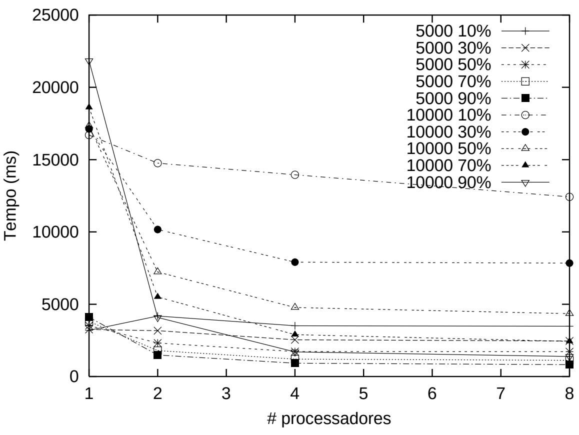

Figura 5. Gráfico da eliminação usando dominância múltipla

Para instâncias maiores o algoritmo de eliminação por dominância múltipla apresenta resultados promissores sempre (ver figura 6). Neste caso é possível constatar melhoras no tempo de execução mesmo quando se passa de 4 para 8 máquinas.

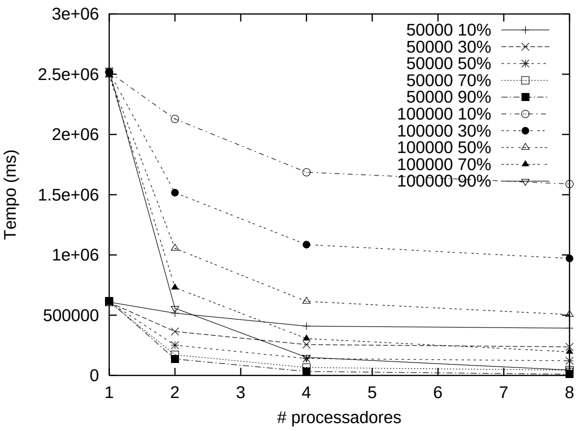

Figura 6. Gráfico da eliminação usando dominância múltipla

De uma maneira geral percebe-se que em ambos os algoritmos, quanto maior a porcentagem de elementos dominados e o número de objetos, maior o *speedup* obtido, principalmente no algoritmo que usa dominância múltipla. Entretanto a melhora no tempo, quando se passa de 4 para 8 máquinas não chega a ser significativa.

7. Conclusão

Com este trabalho ficou claro que é interessante utilizar algoritmos mais sofisticados para a eliminação de objetos no caso de grandes instâncias do problema da mochila.

O experimento também mostrou que algoritmos de eliminação baseados na relação de dominância simples além de reduzirem menos o espaço de busca num *UKP* do que algoritmos que usam dominância múltipla, também apresentam um baixo ganho quando paralelizados. Sendo que chegam até mesmo a ter um desempenho pior quando utilizados 2 processadores do que na implementação seqüencial.

Observa-se que o ganho na paralelização de algoritmos de dominância (especialmente dominância múltipla) além de ser diretamente proporcional ao tamanho da instância cresce com o aumento da porcentagem de objetos dominados dentro dessa instância.

Também é interessante notar que a linguagem Java apresenta recursos suficientes para implementações paralelas que podem apresentar ganhos de desempenho interessantes [19].

Pretendemos continuar pesquisando a possibilidade de paralelização de outros algoritmos de eliminação de termos redundantes em *UKP*. Temos especial interesse no estudo da relação de dominância introduzida em [21] chamada de *threshold dominance* que é mais genérica do que as outras relações de dominância.

Referências

[1] V. Aleksandrov and S. Fidanova. Non-uniform recurrence equations on 2d regular arrays. In *Advance in Numerical Methods and Applications, in Proc. of NMA94*, pages 217–225, August 1994.

[2] V. Aleksandrov and S. Fidanova. On the expected execution time for a class of non uniform recurrence equations mapped onto 1d array. *Parallel Algorithms and Applications*, 1:303 – 314, 1994.

[3] R. Andonov and F. Gruau. A 2D modular toroidal systolic array for the Knapsack problem. In *ASAP'91*, pages 458 – 472, 1991.

[4] R. Andonov and P. Quinton. Efficient linear systolic array for Knapsack problem. In *CONPAR'92*, number 634 in LNCS, pages 247 – 258, 1992.

[5] R. Andonov and S. Rajopadhye. Knapsack on VLSI: from algorithm to optimal circuits. *IEEE Transactions on Parallel and Distributed Systems*, 8(6):545 – 562, 1997.

[6] G-H. Chen, M-S. Chern, and J-H. Jang. Pipeline architectures for dynamic programming algorithms. *Parallel Computing*, 13(1):111 – 117, 1990.

[7] G-H. Chen and J-H. Jang. An improved parallel algorithm for 0/1 knapsack problem. *Parallel Computing*, 18(7):811–821, 1992.

[8] K. Dudzinski. A note on dominance relation in unbounded knapsack problem. *Operations Research Letters 10*, 1991.

[9] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman and Comp., San Francisco, 1979.

[10] T. Gerasch and P. Wang. A survey of parallel algorithm for one-dimensional integer Knapsack problems. *Infor*, 32(3):163–186, 1993.

[11] P.C. Gilmore and R.E Gomory. A linear programming approach to the cutting stock problem - part ii. *Operations Research 11*, 1963.

[12] A. Goldman and D. Trystram. An efficient parallel algorithm for solving the Knapsack problem on hypercube. *Journal of Parallel and Distributed Computing*, (accepted), 2002.

[13] E. Horowitz and S. Sahni. Computing partitions with applications to the Knapsack problem. *Journal of ACM*, 21:277 – 292, 1974.

[14] T.C. Hu. *Combinatorial Algorithms*, chapter 3. Addison Wesley, 1982.

[15] J. Lee, E. Shragowitz, and S. Sahni. A hypercube algorithm for the 0/1 Knapsack problem. *Journal of Parallel and Distributed Computing*, 5(4):438–456, 1988.

[16] J. Lin and J. Storer. Processor efficient hypercube algorithm for the Knapsack problem. *Journal of Parallel and Distributed Computing*, 13(3):332 – 337, 1991.

[17] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, New York, 1990.

[18] D.G. Morales, F. Almeida, C. Rodríguez, J. L. Roda, I.Coloma, and A. Delgado. Parallel dynamic programming and automata theory. *Parallel Computing*, 26:113 – 134, 2000.

[19] C.M. Pancake and C. Lengauer. Special issue on high performance java. *Communications of ACM*, 44(10), 2001.

[20] D. Pisinger. Algorithms for knapsack problems. *Ph.D Thesis*, 1995.

[21] V. Poirriez R. Andonov and S. Rajopadhye. Unbounded Knapsack problem: Dynamic programming revisited. *European Journal of Operational Research*, 123:394 – 407, 2000.

[22] N. Zhu and K. Broughan. On dominated terms in the general Knpasack problem. *Operations Research Letters*, 21:31–37, 1997.