FACULDADES INTEGRADAS DE CARATINGA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

WÉRDE SILVA REIS

ESTUDO SOBRE O PROBLEMA DA MOCHILA E MÉTODOS DE PROGRAMAÇÃO DINÂMICA APLICADOS NA DISTRIBUIÇÃO DE CARGAS EM VEÍCULOS

CARATINGA
2010

WÉRDE SILVA REIS

ESTUDO SOBRE O PROBLEMA DA MOCHILA E MÉTODOS DE PROGRAMAÇÃO DINÂMICA APLICADOS NA DISTRIBUIÇÃO DE CARGAS EM VEÍCULOS

Trabalho de Conclusão de Curso apresentado à banca examinadora da Faculdade de Ciência da Computação, das Faculdades Integradas de Caratinga, como exigência parcial para obtenção do grau de Bacharel em Ciência da Computação, sob orientação do professor Paulo Eustáquio dos Santos.

FIC / CARATINGA
2010

ESTUDO SOBRE O PROBLEMA DA MOCHILA E MÉTODOS DE PROGRAMAÇÃO DINÂMICA APLICADOS NA DISTRIBUIÇÃO DE CARGAS EM VEÍCULOS

WÉRDE SILVA REIS

Trabalho de Conclusão de Curso apresentada à banca examinadora da Faculdade de Ciência da Computação das Faculdades Integradas de Caratinga, como exigência parcial para obtenção do grau de bacharel em Ciência da Computação.

Aprovada em 22 de fevereiro de 2010.

Banca examinadora:

_____

Paulo Eustáquio dos Santos
Prof. – orientador

_____

Prof.

_____

Prof.

DEDICATÓRIA:

Este trabalho e dedicado a:

“Karen de Oliveira Silva Reis” e

“Helidianne de Oliveira Silva Reis”

AGRADECIMENTOS:

Entendo que este é o momento de agradecer não apenas as pessoas que contribuíram na elaboração deste trabalho, mas sim a todas as pessoas envolvidas no trajeto que começou no início da graduação, esta que tem neste trabalho seu como marco definitivo. Logo para não ser injusto não citarei nomes:

Agradeço primeiramente a “Deus”, pois Ele tudo emana e a Ele tudo deve ser creditado. Agradeço também a minha família pelo apoio paciência e carinho. Agradeço aos meus amigos e colegas pelo companheirismo e cumplicidade. Agradeço aos meus mestres pela paciência e dedicação. Agradeço ao meu orientador que mais que um mestre se demonstrou um amigo. E por fim agradeço a todos cujo nome que por alguma razão não foi citado.

Apenas com a ajuda de todos foi possível chegar aonde cheguei.

EPÍGRAFE:

Certa vez, a ocasião da ordenação de grupo de bispos pelo Papa no vaticano, foi perguntado a um dos jovens bispos o que lhe vinha à mente naquele que seria o momento mais importante de sua vida. E então o jovem respondeu:

"Graças a Deus, nada na minha vida eu consegui sozinho."

Autor desconhecido.

RESUMO

Quando se busca encontrar o melhor resultado possível para algum problema, principalmente problemas onde exista uma lista grande de solução, entende-se este problema como sendo um problema de otimização.

Este é o caso do problema estudado, o mesmo se refere a uma empresa de representação comercial que tem como objetivo de otimizar o processo de distribuição dos pedidos em veículos que são contratados para realizar o transporte dos mesmos para os clientes. Este é um problema que se encaixa no conceito de otimização. Deseja-se encontrar uma solução que reduza ao máximo o número de veículos, reduzindo assim os custos com a contratação dos mesmos.

Para tal foram utilizados métodos inerentes a Pesquisa Operacional e a Ciência da Computação a fim de classificar e propor uma solução para o problema, desta forma pode-se diagnosticar que o problema se tratava de uma variação do “Problema da Mochila” e com o tal propôs-se uma solução baseada em Programação Dinâmica.

Solução esta que foi testada e comparada com outros algoritmos existentes na literatura e que são também comumente usados na solução de problemas com características semelhantes. Também efetuou-se comparações das respostas apresentadas pela solução proposta em relação às obtidas através dos métodos atuais que são absolutamente manuais, para isso se utilizou dados reais extraídos dos próprios processos que foram analisados na criação da solução apresentada.

Após todas as etapas acima citadas chegou-se a uma solução fundamentada e desenvolvida a partir de princípios científicos que atende às necessidades da empresa e que obteve um desempenho muito bom frente às outras soluções propostas.

Palavras Chaves: “Pesquisa Operacional”, “Programação Dinâmica” e “Problema da Mochila”

ABSTRACT

In seeking to find the best possible outcome for a problem, especially problems where there is a large list of solution, it is understood this problem as an optimization problem.

This is the case the problem under study, it refers to a company of commercial representation that aims to optimize the process of distributing applications on vehicles hired to carry out the transportation of them to customers. This is a problem that fits the concept of optimization. We want to find a solution that reduces the number of vehicles, thereby reducing the cost of hiring them.

For that we used methods inherent in Operations Research and Computer Science in order to classify and propose a solution to the problem, this way you can diagnose the problem it was a variation of the "Knapsack Problem" and with this proposed a solution based on Dynamic Programming.

Solution which was tested and compared with other algorithms in the literature and are also commonly used in solving problems with similar characteristics. Also made to compare responses from the proposed solution compared to those obtained by current methods that are completely manual, it was used for real data extracted from the very processes that were analyzed in the creation of the solution presented.

After all the steps mentioned above reached a settlement founded and developed on the basis of scientific principles to meet the needs of the company and obtained very good performance compared to other proposed solutions.

Keywords: "Operational Research", "Dynamic Programming" and "Knapsack Algorithm"

SUMARIO:

1. INTRODUÇÃO

A busca por melhores resultados vem motivando várias empresas a investirem maciçamente em pesquisas que tem como principal objetivo melhorar os processos produtivos das empresas. Minimizar custos, maximizar a produtividade, distribuir melhor os recursos são entre outras, ações que fazem com que as empresas consigam extrair o melhor dos seus recursos.

Todos os ramos do conhecimento humano existem com um objetivo específico de melhorar a vida do homem na sociedade, logo realizar um estudo que não tem como fundamento principal a melhoria de uma atividade humana, em qualquer aspecto que seja se torna algo não apenas sem sentido como também absolutamente inviável. Sobre este prisma, esta proposta visa empregar métodos científicos de áreas como, Ciência da Computação e Pesquisa Operacional com o objetivo de resolver um problema cotidiano de uma empresa regional aproximando assim a Ciência estudada nas academias dos problemas do dia a dia vividos pelas empresas em geral.

Este trabalho concentra-se em uma empresa de representação comercial chamada “Dinâmica Rural”, esta empresa tem sua sede na cidade de Ipatinga – MG, e atua no ramo de comércio e representação de produtos agropecuários.

No tangente a sua atuação no campo da representação comercial a empresa trabalha fazendo o papel de intermediária entre clientes (fazendas, lojas, e pessoas físicas) e as fábricas de diversos insumos (rações, fertilizantes, defensivos, sementes, etc.) onde os clientes utilizam a empresa para adquirir estes insumos diretamente da fabrica. A empresa conta ainda com uma frota de veículos terceirizados que são os responsáveis pelo transporte dos produtos desde a fábrica até o cliente final.

Fazendo um papel de representante, a empresa não efetua diretamente as vendas, a mesma é responsável apenas pela intermediação Cliente – Fábrica, ficando responsável por todas as atividades inerentes a este processo. Entre estas atividades está o objeto de estudo desta pesquisa, pois, a empresa é responsável pela distribuição dos pedidos dos clientes entre os veículos que são contratados para realizarem os transportes. Esta atividade que atualmente é feita sem o auxilio de nenhum recurso tecnológico, o que acarreta um empenho de grande quantidade de recursos da empresa destinados a este fim.

A proposta deste trabalho é desenvolver e analisar uma solução que atenda às necessidades da empresa, utilizando os recursos científicos disponíveis para classificar o problema, propor uma solução e analisar seus respectivos resultados.

Uma das abordagens mais utilizadas para solução deste problema é a utilização de uma técnica conhecida como Programação dinâmica, onde basicamente se propõe a divisão do problema principal a fim de encontrarmos uma solução que satisfaça este sub-problema e que venha a fazer parte da posterior solução final conseguida através da junção de todas as sub-soluções.

Porém apenas esta abordagem não foi suficiente para a solução desse problema, foi necessário o uso de técnicas como Algoritmos Gulosos, no apoio a construção da solução, e técnicas de revisão da solução, a fim de que se aproxime da solução ótima.

Por fim formam realizados testes nos quais a proposta de solução foi confrontada com soluções construídas a partir de outros algoritmos nomeadamente uma

solução baseada em um algoritmo de Força Bruta e um algoritmo produzido através de um Método Guloso, ambos explicados nas seções 2.2.1 e .2.23 respectivamente.

A proposta de solução também foi comparada com o método utilizado pela empresa atualmente. Método este completamente manual, que apesar de muito mais lento que qualquer técnica automática, produz soluções muito boas no ponto de vista da qualidade das soluções apresentadas, ou seja, resultados com uma quantidade mínima de desperdício de espaço nos veículos e conseqüentemente um reduzido número dos mesmos.

Este trabalho foi concebido com o intuito de otimizar o processo de distribuição dos pedidos entre os veículos e segue a seguinte estrutura:

Em princípio foi discutida uma série de temas que dão base teórica e técnica a solução apresentada, neste se encontrarão uma síntese de todo o conhecimento usado na confecção da solução, nomeadamente conceitos sobre o problema da mochila e sobre Programação Dinâmica, bem como sua ligação com problema em si.

Em seguida, tratou-se da metodologia aplicada na solução, onde se explanara todos os processos que foram utilizados durante a construção na solução que foi proposta neste trabalho. Será descrita toda evolução do algoritmo bem como seus pontos fortes e francos.

Por último tem-se uma descrição detalhada dos testes e seus resultados que serão discutidos a fim de determinarmos o grau de sucesso da solução proposta. E, em fim, seguem as conclusões e considerações finais, bem como as propostas para evoluções que podem ser feiras dentro do tema discutido.

2. O PROBLEMA DA MOCHILA

2.1. Introdução

Freqüentemente, em diversos pontos da atividade econômica as organizações vêem-se frente a frente com problemas de otimização. Estes por sua vez podem nem sempre se demonstrarem com a clareza necessária e nisso tem-se a necessidade de que os responsáveis possam analisá-los e tomar as decisões necessárias para que sua atividade obtenha resultados ótimos.

Problemas aparentemente diferentes como cortar objetos grandes (por exemplo, bobinas, placas, paralelepípedos) para a produção de itens menores em quantidades bem definidos, ou empacotar itens pequenos dentro de espaços bem definidos são problemas idênticos, considerando que um ítem cortado de uma certa posição pode ser visto como ocupando aquela posição (daí a referência na literatura a Problemas de Corte e Empacotamento).

Há que se considerar que o número de combinações possíveis dos itens dentro de um objeto (cada combinação possível é chamada padrão de corte) é, em geral, muito grande e, a tentativa de enumerá-las completamente é inviável do ponto de vista prático. Uma função objetivo pode ser definida medindo, por exemplo, perdas no caso do problema de corte, ou vazios no caso de empacotamento, ou custos, ou número de objetos usados, etc. Um problema de corte e empacotamento consiste em determinar um padrão de corte que minimize a função objetivo.

O tamanho do conjunto de possibilidades para o corte ou empacotamento é enorme e suas possibilidades de combinação com objetivos de otimização são maiores ainda. Devido a isto, este se encaixa em um grupo de problemas que pertencem a uma

categoria especifica, nesta categoria de problemas estão vários clássicos da literatura de pesquisa operacional, tais como os problemas da mochila, *bin-packing*, SAT, caixeiro viajante, dentre outros, os quais são considerados NP-Completo (Garey & Johnson, 1979) assunto que trataremos adiante.

Nas últimas duas décadas têm sido publicados vários trabalhos de revisão nos problemas de corte e empacotamento, como Hinxman (1980), Dyckhoff *et al*. (1985), Dyckhoff (1990), Martello & Toth (1990), Dowsland & Dowsland (1992), Dyckhoff & Finke (1992), Sweeney & Parternoster (1992), Wäscher & Gau (1996), Dyckhoff *et al*. (1997), Lodi *et al*. (2002) e revistas de prestígio em Pesquisa Operacional têm publicado edições especiais sobre o tema tais como Dyckhoff & Wäscher (1990), Bischoff & Wäscher (1995), Arenales *et al*. (1999). Estes foram responsáveis por grandes contribuições na busca de uma solução que, pelo menos se aproxime da melhor solução possível.

Nota-se também o desenvolvimento de várias técnicas de resolução, em geral especializando procedimentos consagrados da Pesquisa Operacional, tais como: enumeração implícita, programação dinâmica, relaxação lagrangeana, busca em grafos e heurísticas.

No entanto o problema da mochila se demonstra cada vez mais complexo pois, a cada estudo é descoberta uma nova variação do mesmo com características próprias que fogem do modelo inicial. Daí a dificuldade da criação de um modelo geral.

Há na literatura um grande número de problemas de corte e empacotamento e um número próximo de tentativas de classificação e de solução para estes. Houve uma tentativa no trabalho de Dyckhoff (1990) de classificá-los conforme algumas

características. Em particular, uma primeira característica consiste nas dimensões relevantes do processo de corte. Assim podemos ter os problemas de corte unidimensionais com apenas uma dimensão relevante para o processo de corte, como por exemplo, o corte de bobinas (de papel, aço, tecidos, plásticos, etc) e barras. O clássico problema da mochila, onde se procura encher uma mochila com objetos tentando maximizar o valor transportado pela mesma, pode ser visto como um problema de corte unidimensional quando os objetos da mochila não apresentam dimensão, apenas peso. Outros problemas bidimensionais, onde duas dimensões são relevantes, têm aplicações diversas, como por exemplo, o corte de placas de madeira, chapas de aço, placas de vidro, tecido, etc, este é comparado ao problema da mochila quando o mesmo apresenta dimensões, é o caso de problemas como alocação em estoque, carregamento de paletes, carregamento de caminhões dimencionados (*"Baú"*), etc.

Uma outra característica dos problemas de corte e empacotamento decorre da elevada repetição de itens a serem produzidos, de modo que uma solução do problema exige o corte de vários objetos em estoque com repetição de padrões de corte (o número de objetos cortados por um mesmo padrão de corte passa a ser uma variável fundamental na formulação do problema, que é aproximado com sucesso por um problema de otimização linear). Este problema é conhecido na literatura como *Problema de Corte de Estoque* e os trabalhos pioneiros de Gilmore & Gomory (1961, 1963, 1965) utilizaram a técnica de geração de colunas e permitiram pela primeira vez que problemas práticos pudessem ser resolvidos com sucesso. A terminologia de problema de corte de estoque é também usada por Dyckhoff para problemas onde a repetição dos padrões de corte é baixa (por exemplo, apenas um objeto deve ser cortado por um padrão particular), mas há uma demanda a ser atendida, mesmo que em quantidade pequena (por exemplo, um ou dois itens). Neste caso, a solução arredondada

da relaxação por otimização linear pode não ser uma boa aproximação, tornando-se necessária uma outra abordagem. As características destacadas por Dyckhoff (1990) permitem uma classificação dos problemas de corte e empacotamento, muito embora a diversidade de problemas abrigados na mesma categoria, faz com que esta categorização não seja muito bem aceita.

Técnicas de resolução de problemas de corte de estoque (sejam baseadas na geração de colunas ou heurísticas gulosas, segundo Hinxman, 1980 e Wäscher & Gau, 1996) dependem fundamentalmente da resolução de um subproblema de corte e empacotamento para a determinação do melhor padrão de corte para um particular objeto em estoque. Nesse subproblema de corte não há obrigatoriedade de satisfazer a demanda (caso a demanda seja baixa, deve-se cuidar para que a demanda não seja excedida, caso contrário, o padrão de corte gerado não seria usado uma única vez).

Uma variedade de problemas da mochila vem sendo estudada há várias décadas pela comunidade de Pesquisa Operacional, porém há ainda uma lista muito grande de casos a serem estudados. Assim como outros problemas NP-completos, apenas uma pequena parte de seus casos usos foi estudada.

2.1.1 problemas NP-completos

2.1.1.1 Complexidade dos algoritmos

A definição para a classe dos problemas NP-Completos está diretamente ligada à teoria da Complexidade computacional, que é usada para medir o “tempo

computacional” necessário para se resolver um problema, e desta maneira saber se ele é ou não eficiente.

Analisando os algoritmos (suas estruturas de repetição e de decisão) em função de suas entradas podemos dizer se um algoritmo é ou não eficiente. Para que isto ocorra a função do tempo de complexidade do algoritmo, “*O()*”, tem de descrever uma função polinomial em relação a sua entrada “*n*”. Logo, se a ordem de complexidade o algoritmo for: O(1), O(n), $O(n^2)$, $O(n^3)$, $O(n^4)$... este é um algoritmo polinomial. Para funções de complexidade memores que a entrada, *Log n*, por exemplo, apesar de não ser um polinômio também entram na classe dos algoritmos polinomiais. Quanto aos problemas cuja complexidade seja: $O(2^n)$, $O(3^n)$, O(n!), $O(n^n)$, etc, são chamados exponenciais e outros cuja complexidade não pode ser expressa por um polinômio.

2.1.1.2 Tipos dos problemas

Existem algumas classes gerais nas quais os problemas podem ser descritos, entre elas estão os problemas de decisão, localização e de otimização.

Problemas de decisão são aqueles que o objetivo é encontrar uma resposta geralmente SIM ou NÃO, sua estrutura é simples e sua complexidade geralmente é polinomial

Problemas de localização são os que se objetiva encontrar um conjunto de valores que satisfaça um conjunto de propriedades, isto é muito encontrados em casos como buscas de palavras em textos, seqüenciamento de cadeias etc. embora mais

complexos que os problemas de decisão, normalmente ainda tem complexidade polinomial.

Caso o conjunto de propriedades acima citado envolver algum critério de otimização (encontrar o menor valor, o maior número, etc.) o mesmo se caracteriza como sendo um problema de otimização, o que normalmente é mais complexo que o problema de localização, pois tem de obedecer a critérios adicionais de busca.

2.1.1.3 Classes dos Problemas

Em se tratando de complexidade computacional os problemas são divididos em nas classes P, NP, NP-Completo e NP-Dificil.

A Classe P engloba os problemas que são solucionados através de algoritmos que possuem uma complexidade polinomial, ou seja, existe um algoritmo que consiga encontrar a solução ótima que possua complexidade polinomial em relação à sua entrada. Para provar que um problema é da Classe P basta apresentar um algoritmo polinomial que o resolva, mas para provar que ele não pertence a esta classe, deve-se provar que não existe algoritmo polinomial para o problema, o que é muito mais difícil, pois não se consegue provar que um problema não pertence à Classe P, apenas porque não se conseguiu encontrar ainda um algoritmo polinomial que o resolva.

A Classe NP, por sua vez tem uma definição um pouco mais completa, ela define todos os problemas de decisão cuja justificativa para a resposta SIM possa ser verificada em tempo polinomial. NP quer dizer Não-deterministico Polinomial, ou seja, sua verificação pode ser feita por uma “máquina Não-determinística” (não entraremos

como algoritmos padronizados para a solução de problemas computacionais) em tempo polinomial. Este é o caso do Ciclo Hamiltoniano, pois dado um Grafo G, e um conjunto de vértices pertencentes a G, existe um algoritmo que possa verificar se há um Ciclo Hamiltoniano que passe por estes vértices em tempo polinomial. O mesmo ocorre com diversos outros problemas descritos na literatura e que também pertencem a classe dos problemas NP.

Os problemas da Classe NP-Completo, são os problemas mais importantes da Ciência da Computação, pois todos os problemas desta classe são equivalentes entre si, ou seja, para pertencer a NP-Completo o problema tem que pertencer a NP e ser reduzido polinomialmente a outro problema que já pertençam à classe dos problemas NP-Completos. Portanto, para saber se um problema é NP-Completo basta conseguir transformá-lo polinomialmente em um outro problema NP-Completo.

Esta definição é muito poderosa porque através dela pode-se concluir que ao encontrarmos um algoritmo que resolva um problema NP-Completo, ele solucionará qualquer outro problema desta classe, logo, se o algoritmo encontrado for polinomial, provaríamos que NP-Completo = P.

A Classe NP-Dificil, é a classe que compreende os problemas que são transformáveis polinomialmente em um problema NP-Completo, porém, a verificação da resposta SIM de seu problema de Decisão não pode ser feita por uma Máquina de Turing (máquina não-determinística), ou seja, os problemas da classe NP-Difícil, pertencem a Classe NP-Completo e não pertencem a NP. Esta classe engloba praticamente todos os problemas de otimização, já que verificar se uma solução é realmente a melhor entre todas as outras possíveis é uma tarefa que, na maioria das vezes, não pode ser feita em tempo polinomial.

A figura 1 descreve a classificação dos problemas e suas abrangências em relação aos problemas das outras classes.

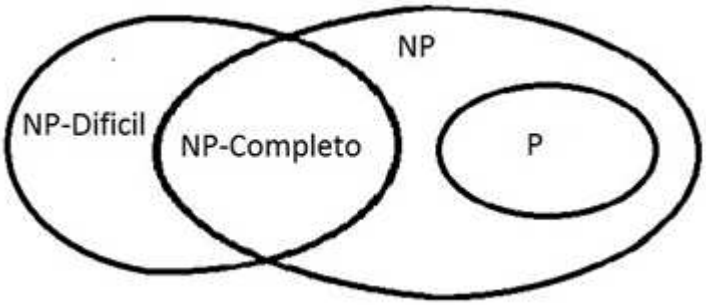


Figura 1: Intersecção entre as classes dos problemas

2.2. Métodos para solução do Problema da Mochila

O problema da mochila (“knapsack problem”) é um clássico nas áreas de Pesquisa Operacional, Análise de Algoritmos e Inteligência Artificial. Ele consiste em, dado um conjunto de elementos, e tendo cada elemento um valor de utilidade e/ou um peso ou dimensão, conseguir um subconjunto de elementos que maximize a utilidade total da mochila sem ultrapassar um limite de seu peso total. Uma analogia freqüentemente usada é a de um ladrão que precisa carregar o máximo que conseguir de uma casa, maximizando o valor total do roubo, ou de alpinista que precisa levar na mochila itens que garantam sua sobrevivência em sua escalada, por isso, a necessidade de maximizar a quantidade e a qualidade dos itens levados. Daí a razão do nome do problema.

O “problema da mochila” pode ser definido em termos formais da seguinte maneira:

> [1]*Dado um conjunto Cn de n itens, representados por Cn = {1,2,...,n}, cada item i ∈ Cn tem um peso pi e utilidade ui (pi > 0 e ui > 0). Determinar um conjunto S CONTIDO Cn tal que a soma dos pesos dos elementos de S seja menor ou igual à capacidade da mochila L e que a utilidade total dos elementos de S seja a maior possível.*

Pode-se resolver esse problema sob três abordagens diferentes: com força bruta, programação dinâmica ou algoritmo de aproximação.

Há também abordagens menos genéricas e soluções que unem duas ou mais abordagens, porem estas podem nos dar uma visão das possíveis soluções, bem como de sua complexidade de solução.

Tem-se essas três abordagens, mostrando os algoritmos e estruturas de dados utilizadas, assim como discussões de limite de instâncias, complexidade e tempos envolvidos na execução de cada algoritmo.

---

[1] Cormen, Thomas H.,Leiserson, Charles E., Rivest, Ronald L., Stein, Cliford, “Algoritmos, Teoria e Prática”Luis F., Editora Campus, 2003.

2.2.1 Solução ótima

Se tratando de um problema NP-completo, a única maneira de garantirmos que a solução encontrada é a solução ótima, é buscarmos a melhor solução em um conjunto contendo todas as soluções possíveis.

Um algoritmo simples para resolver o “problema da mochila” é apresentado a seguir. Ele é dito ser de “força bruta”, pois calcula todas possíveis combinações de itens a fim de se encontrar a maior utilidade que é possível colocar com um determinado peso na mochila.

ALGORITMO 1 (Força Bruta)

algoritmo forca_bruta()

```
1.  utilidade_maxima <- 0
2.  sequencia_otima <- NIL
3.  for i = 0 to (num_itens(Itens) - 1)
4.        do sacola.utilidade <- 0
5.        sacola.peso <- 0
6.        forca_bruta_r(i, toSTR(i), sacola)
7.    i = i + 1
8.  imprime(sequencia_otima,utilidade_maxima)
```

ALGORITMO 2 (Força Bruta)

algoritmo forca_bruta_r(posItens, seqTestada, sacola)

```
1.  sacola.utilidade <- sacola.utilidade + Itens[posItens].utilidade
2.  sacola.peso <- sacola.peso + Itens[posItens].peso
3.     if (sacola.peso > capacidade_mochila) then return
4.     if (sacola.utilidade > utilidade_maxima) then
5.        do utilidade_maxima <- sacola.utilidade
```

```
6.      sequencia_otima <- seqTestada
7    for  i = (posItens + 1) to (num_itens(Itens) - 1)
8.      do forca_bruta_r(i, concatSTR(seqTestada,toSTR(i)), sacola)
```

Na implementação os valores dos itens, conjunto Cn, são representados por um arranjo de *structs*, onde o mesmo contém o peso e a utilidade de cada item; e a seqüência ótima resultante da escolha dos itens que maximizam a utilidade é representada por um arranjo simples.

A complexidade do algoritmo de combinação, sem adição de nenhuma otimização, é dada pela fórmula:

$$\sum_{k=1}^{n} \frac{n!}{k!(n-k)!}$$

Formula 1: Descreve a complexidade computacional do algoritmo Força Bruta

Onde n é o número de elementos da mochila.

2.2.2 Programação Dinâmica

Programação Dinâmica é uma técnica computacional que consiste em dividir o problema em subproblemas (problemas menores), onde suas soluções são utilizadas na construção da solução final, esta técnica é apoiada no Princípio da Optimalidade, que garante a validade da solução final desde que o problema se encaixe em certos padrões.

É também uma técnica regressiva (Button Up), ou seja, sua solução é buscada a partir de soluções menores, partindo daí para a solução final.

Ao resolver o “problema da mochila” por programação dinâmica, dividi-se o problema em subproblemas menores, onde as soluções destes subproblemas são computados e armazenados em um registro. A partir dessas soluções menores as soluções maiores são computadas, até alcançar a solução final, assim nunca recalculando soluções de um subproblema que já foi resolvido.

Seja UT(i,M) o valor máximo de utilidade total que pode ser conseguido resolvendo o problema da mochila, dados Ci = {1,2,...,i} e uma mochila de capacidade

$$
\begin{cases}
UT(0,M) = 0 \\
UT(i,M) = UT(i-1,M) & \text{, se } Pi > M \\
UT(i,M) = \max [UT(i-1,M),\ UT(i-1, M - Pi) + Vi] & \text{, se } Pi <= M
\end{cases}
$$

M. Então pelo princípio da optimalidade descrito acima, temos a seguinte relação de recorrência para o problema da mochila:

O algoritmo consiste em resolver a relação de recorrência até encontrar o valor de UT(n,M).

ALGORITIMO 3 (Programação Dinâmica)

algoritmo prog_dinamica(Itens,capMochila)

```
1. for c = 0 to capMochila
2.     do matriz[0,c] <- 0;
3.       for i = 1 to num_itens(Itens)
4.        do aux1 <- matriz[i-1,c]
```

```
5.          if Itens.peso[i] <= c
6.                  then aux2 <- matriz[i-1,c-Itens.peso[i]] + Itens.utilidade[i]
7.          else aux2 <- 0
8.      matriz[i,c] <- max(aux1,aux2)
9.  return (matriz[num_itens(Itens),capMochila])
```

Programa 2: Proposta inicial para a implementação do algoritmo por Programação Dinâmica

Na implementação os valores da função UT são armazenados em uma matriz de inteiros, onde as linhas são indicadas pelos itens em Cn e as colunas pelas capacidades possíveis da mochila. Além disso, os itens (conjunto Cn) são representados por um arranjo de structs, que possui as informações de peso e utilidade.

A seqüência ótima obtida através da função UT indicada pela matriz é representada também por um arranjo simples.

O algoritmo consiste em dois laços do tipo ‘for’ e comandos simples, de complexidade O(1). Portanto, a complexidade será dada pelo número de iterações dos dois laços. O primeiro (linhas 2 a 8) contém outro laço aninhado (linhas 3 a 8). Vê-se que a complexidade total do primeiro laço é O(nP), onde P é a capacidade da mochila, e a do segundo é O(n). Portanto, a complexidade do algoritmo é O(nP) + O(n) = O(nP). Esta complexidade corresponde tanto à complexidade de tempo como de espaço. Sendo por tanto um algoritmo de complexidade polinomial.

2.2.2.1 Principio da Optimalidade de Bellman

Para Bellman o qualquer estratégia ótima deveria ter a seguinte propriedade:

➔ Qualquer que seja o estado e decisão iniciais, as decisões restantes têm que construir uma estratégia ótima a partir do estado resultante da decisão inicial.

E deste modo podemos dizer que:

➔ Qualquer sub-estratégia será ótima se partir de uma estratégia ótima e respeitar suas regras de estados e decisões.

Apesar de poderoso este princípio é apenas uma orientação, pois não existe uma regra geral que o defina, por mais surpreendente que possa parecer.

Esta técnica baseia-se na idéia de que é comum a todos os problemas resolvíveis por Programação Dinâmica, que exista uma forma de dividi-los em sub-problemas com sub-soluções ótimas que possam ser agrupadas para a solução final, também ótima.

Esta estratégia acaba levando até o fim do problema antes de se começar à resolve-lo, o que faz com que em seu retorno ao princípio seja mais proveitoso para a solução final já que as instâncias intermediárias, muitas vezes, já são conhecidas.

É neste princípio que as técnicas de programação dinâmica se apóiam pois seguem o conceito de uma decisão ótima para um subproblema é parte da decisão ótima para o problema final.

2.2.3 Algoritmo por aproximação

As aproximações ou heurísticas são técnicas que não garantem a solução ótima e sim uma solução possível do ponto de vista do uso dos recursos, já que nem sempre as soluções exatas são viáveis por consumirem muitos recursos computacionais.

Estas técnicas procuram a melhor aproximação possível e em muitos casos são a melhor solução encontrada (é o caso dos problemas NP) e podem ser aplicadas a uma gama enorme de problemas, pois representam uma opção que une bons resultados a um baixo consumo de recursos.

Existem diversas heurísticas e cada uma é mais indicada para um grupo de problemas devido as suas particularidades e limitações.

Há também as meta-heurísticas que são uma espécie de união de técnica de mais de uma heurística ou de heurísticas com outros métodos de solução. Estes híbridos têm recebido grande atenção dos estudiosos, pois representam um enorme ganho na qualidade da solução e no consumo de recursos, conseguindo um melhor resultado utilizando menos recursos.

A seguir apresenta-se uma heurística gulosa para a solução do “problema da mochila”. Os seguintes passos são executados:

1 – Ordenar os itens segundo a razão valor/peso.

$$v1/p1 < v2/p2 < \ldots < vn/pn$$

2 – Inserir os itens na mochila na ordem reversa à ordem apresentada acima, até que um item não “caiba” na mochila, assim inserindo os itens com maior custo

benefício. Mediante isso, inserimos os itens que parecem ser mais promissores, justificando o fato do algoritmo ser considerado guloso, e uma vez selecionado um item, ele nunca mais vai ser selecionado novamente.

ALGORITMO 4 (Ordenação dos itens)

algoritmo prog_aprHeur(Itens,capMochila)

```
1.      sequencia_aproximada <- NIL
2.      quickSort(Itens, 0, num_itens(Itens) - 1)
3.      num_itens_selecionados <- aproxima(Itens,capMochila)
4.  imprime(num_itens_selecionados,sequencia_aproximada)
```

ALGORITMO 5 (Algoritmo de ordenação)

algoritmo quickSort(Itens, left, right)

```
1.        last <- left
2.   if (left >= right) then return
3.   troca(Itens,left,seleciona_aleatorio(left,right))
4.   for i = (left + 1) to right
5.      do if (Itens[i].utilidade / Itens[i].peso) < (Itens[left].utilidade / Itens[left].peso)
6.   then last = last + 1
7.   troca(Itens,left,last)
8.   quickSort(Itens,left,last-1)
9.   quickSort(Itens,last+1,right)
```

ALGORITMO 6 (Algoritmo que carrega a mochila.)

algoritmo aproxima(Itens,capMochila)

```
1.   qtdeDisponivel <- capMochila
2.   i <- (num_itens(Itens) - 1)
3.   iPosSeq <- 0
4.   While (qtdeDisponivel > 0 && i >= 1)
5.         if (Itens[i].peso <= qtdeDisponivel)
6.        then sequencia_aproximada[iPosSeq] = i
7.                iPosSeq = iPosSeq + 1
8.           qtdeDisponivel = qtdeDisponivel - Itens[i].peso
```

```
9.      i = i -1
10.  return iPosSeq
```

Programa 3: Proposta inicial para implementação do algoritmo utilizando uma heurística de Aproximação..

A estrutura de dados utilizada na implementação é semelhante ao do algoritmo de “força bruta”, incluindo apenas o número do índice do item no arranjo de *structs*. A ordenação é feita pelo método *Quicksort*, ordenando o próprio arranjo de itens.

A ordenação na linha 2 tem custo O(nlogn), pois é implementada utilizando o algoritmo *Quicksort* aleatório. O custo da linha 3 é obtido através do algoritmo aproxima, que possui um laço do tipo ‘*while*’ que é executado O(n) vezes. Portanto, a ordem de complexidade do algoritmo de aproximação é O(n) + O(nlogn) = O(nlogn).

3. METODOLOGIA.

3.1. O Problema.

Alocar cargas em veículos não é apenas selecionar os pedidos e distribuí-los entre os veículos disponíveis. Deve-se, portanto, pensar no peso, formato, tamanho e mais uma série de variáveis em relação às cargas. Há que se ter em consideração o preço do transporte e da carga, além da capacidade, disponibilidade e condições técnicas dos veículos. A cada variável acrescentada a complexidade do problema é também aumentada.

Com isso em mente, o objetivo é sempre a melhoria do processo conseguindo assim diminuição de custos, aumento da produtividade e como conseqüência, maior lucratividade. Porém otimizar um processo não é uma tarefa simples. Além das dificuldades cotidianas (problemas financeiros, falta de pessoal, prazos, etc.) existem as dificuldades técnicas já que otimização é um processo que envolve um profundo estudo.

O caso estudado refere-se a uma empresa de representação comercial que necessita de uma solução para agilizar seu processo produtivo. Dentre os vários problemas encontrados está o da distribuição dos pedidos em cargas nos veículos que fazem as entregas para os clientes. Isso por si só já seria um desafio que, quando levado em consideração a quantidade de pedidos a serem transportados e a quantidade de veículos envolvidos, torna-se uma missão muito mais árdua.

Tendo estes dados em mente, será levado em consideração apenas o peso dos pedidos a serem transportados e a capacidade dos veículos de transporte, isso com o intuito de definir um bom escopo para o trabalho, tendo em vista também que esta

operação é morosa e complexa, além de ser um dos mais importantes processos da empresa.

A empresa em questão realiza o seguinte trabalho:

- Um cliente faz o pedido à empresa.
- Em seguida ela repassa o pedido para a fábrica.
- Em datas previamente acertadas a empresa contrata veículos para irem até a fábrica a fim de transportar os pedidos dos clientes e levá-los até os mesmos.
- A empresa indica para a fábrica quais pedidos serão transportados por um determinado veículo.
- Os veículos contratados realizam as entregas dos pedidos diretamente da fábrica até os clientes.

Desta forma as seguintes situações se fazem relevantes:

1. A empresa não possui estoque físico relevante para esta atividade (existe um estoque que atende urgências, mas este foge ao escopo do trabalho).

   Por não haver estoques, a empresa não realiza carregamentos; apenas indica a fábrica.
2. As entregas são de responsabilidade do veículo contratado (há casos em que a empresa intervém, porém isto não é regra);
3. A carga de cada veículo é definida com os pedidos dos clientes, e este trabalho é de responsabilidade da empresa.
4. As rotas de entregas também são definidas pela empresa.

5. Existe um grupo de veículos que atende a cada área de entrega, fazendo cada um, se necessário, todas as rotas desta área.
6. Com fim de organização, existem datas para o fechamento dos pedidos em sua respectiva entrega. Logo, todos os pedidos de um determinado período são entregues simultaneamente.
7. Não se faz necessário ter aspectos como o método de formação da carga pois as mesmas se tratam apenas de sacas de aproximadamente 40 Kg, podendo as mesmas serem acomodas a critério do responsável pelo carregamento. Aspectos como altura e largura da carga também não são considerados já que o primeiro limite físico do veículo a se esgotar é justamente o peso aspecto este que já esta sendo tratado.

Em resumo: Os clientes fazem os pedidos a empresa de representação, esta os remete a fábrica. E a mesma, tendo em mãos os pedidos, se encarrega de carregá-los nos veículos, conforme indicação da empresa de representação, e estes últimos realizam a entrega dos pedidos aos clientes. Este fluxo descreve um sistema onde a empresa de representação faz apenas um papel de intermediadora entre as partes. Porém ela tem uma função de vital relevância que é a de contratar os veículos para transportar os pedidos. Além disso é ela quem define qual veículo vai transportar cada pedido, de maneira que no ato do carregamento na fábrica, a mesma já deve estar ciente de quais pedidos formarão a carga de um determinado veículo.

Outro aspecto importante é que existem áreas onde cada veículo atua. Se um veículo pertence à área 2, ele não fará entregas na área 1 (salvo em casos especiais, porém isso geraria uma despesa extra) e, a princípio, este veículo faria todas as rotas desta área, anulando portanto, o problema de roteamento dos veículos. Claro que

otimizar também o roteamento dos veículos é algo interessante mas foge do escopo do trabalho.

Logo, existe uma lista de pedidos a serem entregues e uma lista de veículos contratados, e visa-se otimizar a distribuição dos pedidos entre os veículos a fim de termos a menor perda de capacidade de carga dos veículos, reduzindo assim o desperdício e a quantidade de veículos contratados, e conseqüentemente, reduzindo as despesas.

De um modo mais formal pode-se dizer que:

- Existe uma lista de pedidos P contendo n elementos
- Há tambem uma lista de veículos V contendo m elementos
- Para cada item Pn tem-se K que determina o peso de cada pedido
- Para cada item Vn tem-se C que determina a capacidade máxima de cada veículo

Assim temos :

$$\sum_{i=1}^{n} K \leq C$$

Como se deseja maximizar a função de peso para cada veículo pode-se dizer que a função objetivo é:

$$\forall\, Vn \in V \qquad \max\left(\sum_{i=1}^{n} K \leq C\right)$$

Um problema complexo e como tal deve-se começar sua solução pela modelagem do mesmo. Inicialmente é importante perceber que existem dois conjuntos de informação; o conjunto dos pedidos P= {P_1, P_2, P_3, P_4,..., P_n} e outro com os veículos V={ V_1, V_2, V_3, V_4, ..., V_n, }. Cada pedido possui uma propriedade extra que é o seu peso, e cada veículo também possui uma propriedade que é a sua capacidade. Logo, tem-se: P= { P_1, C_1; P_2, C_2; P_3, C_3; P_4, C_4;..., P_n, C_n } para o conjunto dos pedidos e V= { V_1, P_1; V_2, P_2; V_3, P_3; V_4, P_4; ... ,V_n, P_n } para os veículos.

P_1	P_2	P_3	P_4	P_5	...	P_n
2,5	10	12,9	5	4,3		16

V_1	V_2	V_3	V_4	V_5	...	V_n
8	10	16	4	10		4

Figura 2: Relação entre capacidade e veículo, e, pedido e peso

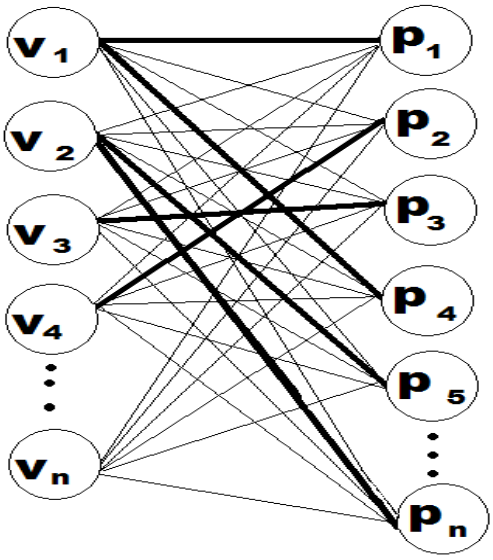
Para cada pedido um Peso e para cada Veículo uma quantidade

Cada veículo tem uma capacidade máxima que não deve ser excedida. A primeira vista parece simples, mas enganos quando da alocação dos mesmos nos caminhões, e isso pode fazer com que se perca espaço. Este espaço perdido, acumulado em diversos veículos, pode ser responsável pela contratação de mais veículos, aumentando assim os custos.

Isso ocorre devido ao fato de que cada veículo pode conter todos os pedidos que são menores ou iguais à sua capacidade. Ou seja, pode-se alocar em um veículo qualquer pedido até que sua capacidade seja atingida. Além disso, existe o fato de se ter

mais de um veículo disponível, então quando se pensa em otimização é necessário pensar na possibilidade de que um pedido em vários veículos e a decisão de qual veículo utilizar para cada pedido é crucial.

Dividindo o problema em dois conjuntos e assumindo que cada parte de um conjunto pode se ligar livremente com as partes do outro conjunto, como foi descrito acima, temos então um [2]grafo bipartido completo.



A direta, tem-se a coluna com a representação dos pedidos

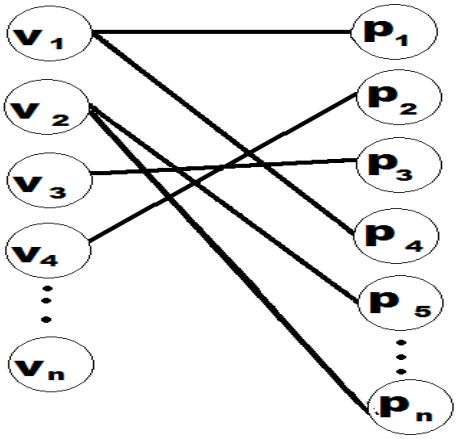
E a esquerda, a coluna com a representação dos veículos

Figura 2: Representação da relação entre os veículos e pedidos e suas possíveis ligações.

[2] O grafo é uma representação gráfica das relações existentes entre elementos de dados. Ele pode ser descrito num espaço euclidiano de n dimensões como sendo um conjunto V de vértices e um conjunto A de curvas contínuas (arestas)

Grafo completo é é um grafo simples em que todo vértice é adjacente a todos os outros vértices.
Grafo bipartido é aquele em que *V* pode ser dividido em dois
subconjuntos disjuntos não vazios *V1* e *V2*.
– Cada aresta conecta um vértice de *V1* e um vértice de *V2*.
• Grafo bipartido completo: cada um dos elementos de *V1* é adjacente a cada um dos elementos de

O pedido P_1 pode ser alocado nos veículos V_1, V_2, ..., V_n, porém ele efetivamente apenas será alocado em um deles. Porém, em contra partida, o veículo V_1 poderá receber qualquer pedido entre P_1, P_2, ..., P_n, até que sua capacidade seja alcançada.

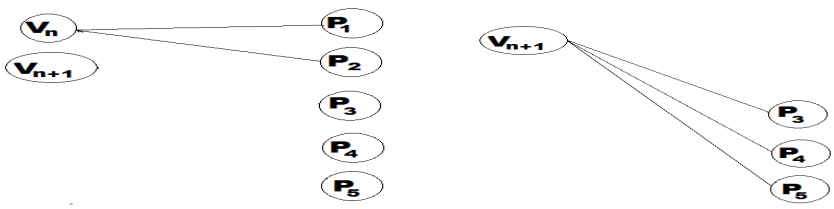


Cada pedido é alocado em apenas um veículo, porém os veículos comportam quantos pedidos sua capacidade suportar

E os pedidos são alocados até que a capacidade seja alcançada.

Figura 3: Representação de um possível carregamento onde cadê pedido está ligado a um veículo.

Cada pedido alocado em um veículo deixa de estar disponível para os demais veículos. Cada veículo completo deixa de estar disponível para a alocação de novos pedidos.



O veículo Vn e os pedidos P_1 e P_2 não esto disponíveis na segunda interação.

Figura 4: Representação de uma possível iteração entre os paços dos carregamentos.

Este problema se enquadra quase perfeitamente na definição do Problema da Mochila, onde as diferenças principais são a falta do valor de utilidade, já que todos os pedidos têm o mesmo valor neste quesito, e a existência de vários veículos, várias mochilas, com diversas capacidades diferentes. Esta última diferença é de vital importância no problema, pois como o objetivo é otimizar a distribuição dos pedidos tem-se sempre de levar em consideração ao alocar um pedido que pode haver outro veículo que possa receber aquele pedido.

3.2. O desenvolvimento da solução

Um fator muito importante para o sucesso de uma solução é a formatação de sua entrada, pois nela devem estar contidas as principais informações e estas, por sua vez, devem estar organizadas de forma a garantir um bom desempenho para os algoritmos apresentados.

Os algoritmos propostos atuam sobre uma entrada previamente padronizada que utiliza dois arquivos de texto retirados anteriormente da base de dados do programa original. Estes arquivos detêm respectivamente os dados dos pedidos e dos veículos que são orientados das seguintes formas:

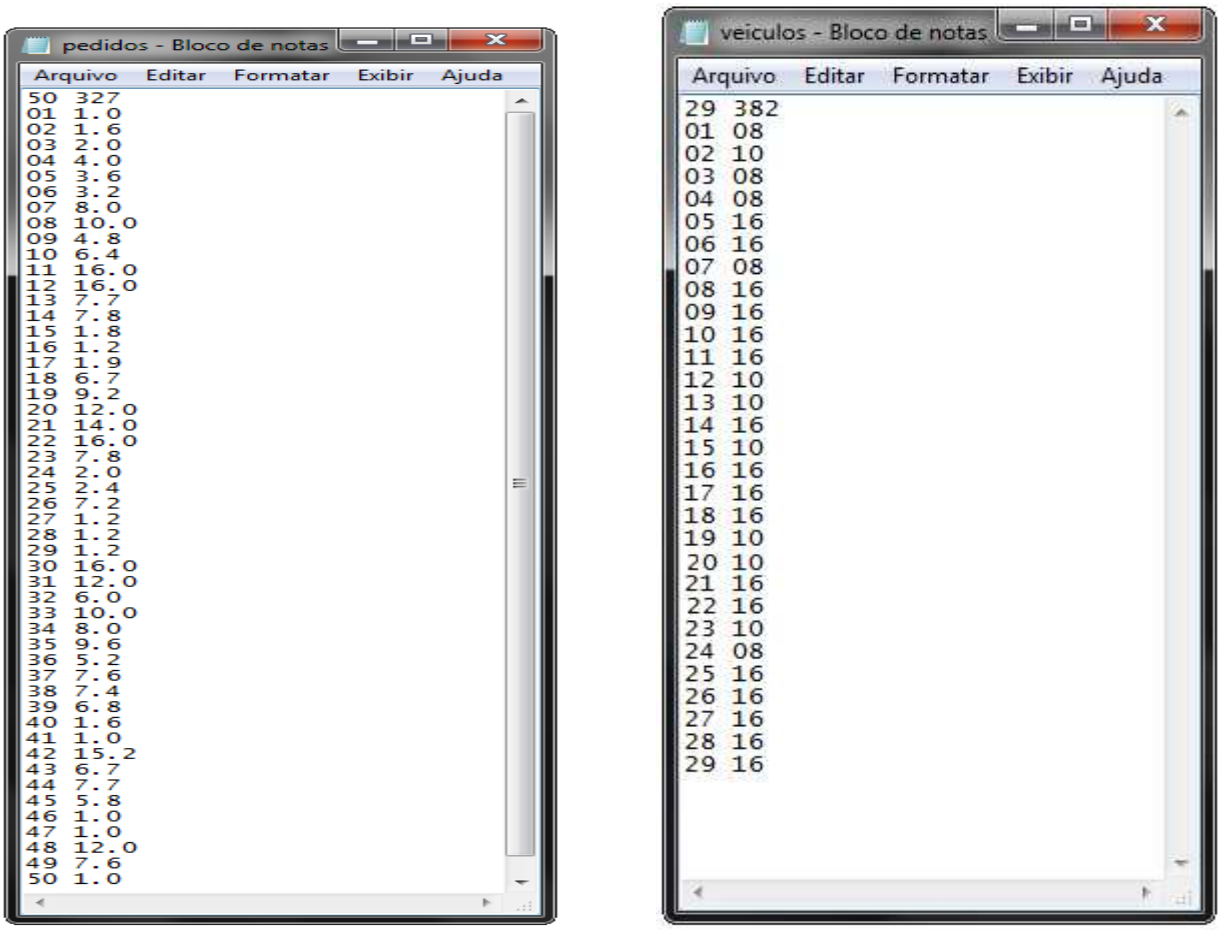


Figura 5: Modelo das entradas para a solução.

Estes dados foram usados para os testes na fase inicial de implementação da solução, são dados fictícios que reproduzem uma situação real, e estão organizados de maneira a ajudar na leitura das principais informações do problema, onde na primeira linha de cada arquivo aparece o número total dos veículos e pedidos, em seguida na mesma linha estão a capacidade total dos veículos e o peso total dos pedidos. Nas linhas subseqüentes encontram-se os identificadores e capacidades dos veículos e os números e pesos dos pedidos individualmente.

Esta entrada permite delimitar as estruturas de repetição do programa a partir de um estudo da entrada, já que ela nos fornece as quantidades de veículos e pedidos e

também fazer análises individuais que são imprescindíveis através dos identificadores individuais tanto dos veículos como dos pedidos bem como suas capacidades e pesos.

Outros dados importantes encontrados na entrada são o peso total e a capacidade total, onde através destes dois dados faz-se uma análise determinando se os veículos contratados são suficientes para transportar todos os pedidos.

Quando se tem em mãos um problema de otimização entende-se que a melhor solução é sempre a solução ótima.

No caso estudado a solução ótima é aquela que nos retorne a distribuição dos pedidos entre veículos de maneira que o desperdício de espaço seja mínimo utilizando, portanto, um número menor de veículos no transporte.

Como já foi discutido anteriormente, este problema é compatível com o Problema da Mochila, que por sua vez é um problema NP-Completo. Estas informações nos remetem ao fato de que para se garantir a solução ótima, deve-se testar todas as possíveis soluções e assim descobrir com certeza qual é a melhor solução. É hábito denominar-se esta técnica como Força Bruta.

Os algoritmos Força Bruta tem esse nome porque cercam todas as possibilidades, testando-as e comparando-as com o fim de descobrirmos qual é a melhor. Normalmente se implementa uma solução que gere todas as respostas possíveis, em seguida cria-se uma outra parte da solução, testa todas as respostas conseguidas e nos diz qual foi a melhor.

Este algoritmo normalmente é muito simples de ser implementado, porém, ele não é muito eficiente no que se trata de tempo computacional por ter uma complexidade

exponencial em relação à entrada, à medida que a entrada aumenta sua execução vai consumindo cada vez mais tempo e recursos da máquina.

Em teoria é muito simples criar uma lista de todas as possíveis soluções e depois encontrar uma solução que atenda às restrições, porém, neste caso, não é tão simples assim. Como temos vários veículos (“Mochilas”) os quais os carregamentos dependem uns dos outros, isto faz com que seja necessária uma atenção especial quanto ao encontro da melhor solução. Este fato também eleva bastante a complexidade do algoritmo e aumenta muito o seu tempo computacional

Apesar disso é importante que se implemente um algoritmo Força Bruta para se ter conhecimento da solução ótima e posteriormente poder avaliar sua viabilidade, assim como compará-lo com outras soluções.

Para tal propõe-se:

ALGORITMO 7: FORÇA BRUTA:

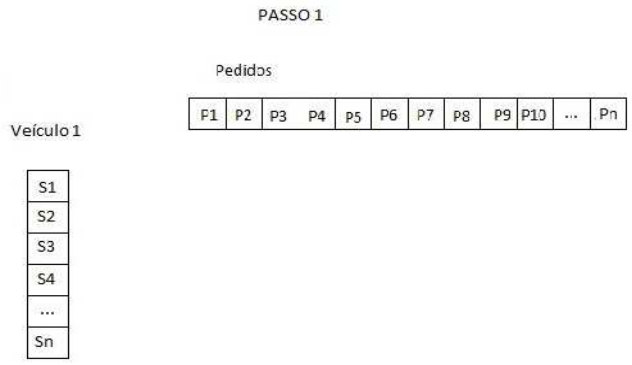
```
Algoritmo Força bruta Gera Soluções
1      for (i=0; i<v; i++){
2           inicio = 0
3              while (inicio<n){
4                carga = 0
5                for (s=0; s<n ; s++) {
6                    veículos[i].resposta[j].carga=[p]=pedido[s].num
7                   p++
8                   carga = carga+pedidos[s].peso
9                   if carga >= veículos[i].capacidade
10                        j++
11               }
12               inicio ++
13                }
14     }

Algoritmo Força bruta Encontra Melhor solução
Criar uma lista com as PossíveisSoluções preenchida com:
1 while (i<v) // v é o número de veículos
2        PossiveisSoluções[i].NúmeroVeículos++
```

```
3       PossiveisSoluções[i].veículo[cont] = veículo[i].id
4          for (aux=0; aux<p; aux++) //p é o tamanho do vetor com os pedidos
           da solução
5           PossiveisSoluções[i].carga[p] = veículos[i].resposta[j].carga=[p]
6       for (cont=0; cont<v; cont++)
7         for (c=0; c<j; c++)
8              { for (a=0; a<p; a++) //p é o tamanho do vetor com        os
             pedidos da solução
9            for (b=0; b<p; b++)
10           {                    if      (veículos[i].resposta[j].carga=[p]!=
             PossiveisSoluções[i].carga[p])
11                 resposa = "Sim"
12               else
13                 resposta = "Não"
14                }
15           if (resposta== sim)
16           PossiveisSoluções[i].veículo = veículo[i].id
17           for (aux=0; aux<p; aux++)
18            PossiveisSoluções[i].carga[p] =veículos[i].resposta[j].carga=[p
19           }
20       cont++
21       for (z=0; z<n; z++)
22      {    if (pedidos[z].carga !=0)
23                 I++
24           else
25                 I=v
26    }
27 Fim do while
28 num=PossiveisSoluções[0].NúmeroVeículos
29 for (i=0; i< I; i++)
30    if (PossiveisSoluções [i]<num)
31    {      num=PossiveisSoluções[0].NúmeroVeículos
32           solução = PossiveisSoluções[i]
33    }
34 Return solução;
```

Este algoritmo divide-se em duas partes sendo que a primeira é responsável por gerar uma lista contendo todas as possíveis soluções para o carregamento de cada veículo, e a segunda encontra a solução que atende às necessidades do problema.

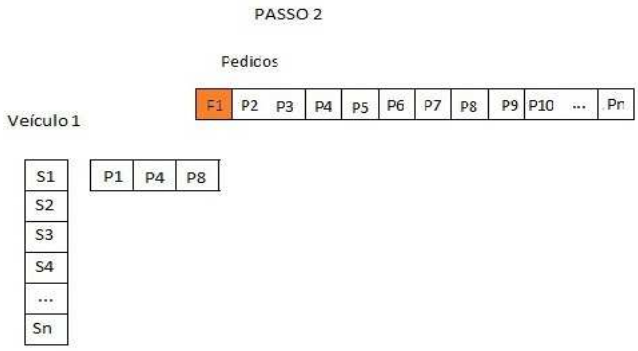
A primeira parte atua no problema de forma a criar uma lista de soluções possíveis para cada veículo da seguinte forma: em cada veículo cria-se a primeira das possíveis soluções através de um ***método guloso***, percorrendo o vetor de pedidos a partir do início e gerando uma solução, em seguida repete o processo começando do

próximo pedido e assim sucessivamente, sempre comparando as soluções com as soluções já existentes e gerando apenas novas soluções para o veículo em questão.



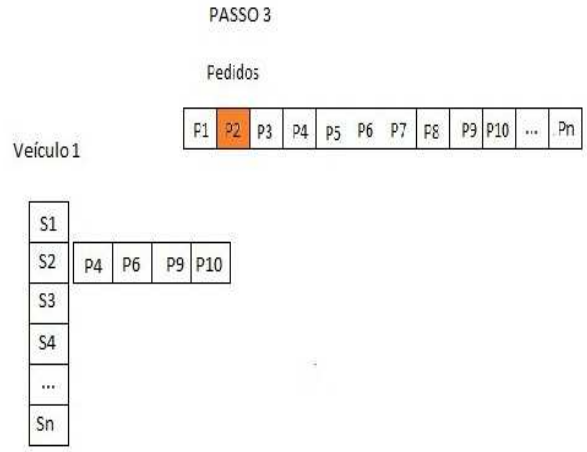
O primeiro passo o algoritmo começa a partir do primeiro item da lista de pedidos a fim de formar a primeira solução

Figura 6: Primeiro passo do algoritmo força bruta



É criada uma lista de pedidos que compatível à um possível carregamento do veículo este pedidos são alocados até que sua capacidade seja atingida

Figura 7: Segundo passo do algoritmo força bruta

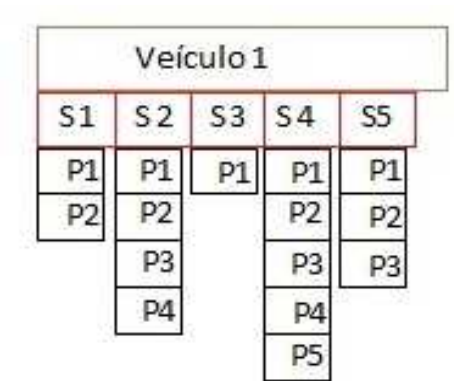


Em seguida a próxima possível solução é obtida a partir do mesmo método porém tendo um outro pedido como ponto de partida. Conseguindo assim outro resultado para um possível carregamento.

Este método garante que todas as possíveis soluções para cada veículo possam ser analisadas

Figura 8: Terceiro passo do algoritmo força bruta

Desta forma ele gera todas as soluções possíveis para um veículo. Ao repetir o processo para os outros veículos o algoritmo desconsidera as soluções encontradas nos veículos já analisados de maneira que sejam encontradas todas as soluções possíveis para cada veículo.



Cada veículo possui uma lista de possíveis soluções onde cada item desta lista é composto por um grupo de pedidos que forma um carregamento para cada veículo analisado.

Figura 9: Lista de possíveis carregamentos para um veículos.

A segunda parte do algoritmo encontra a melhor solução. E esta é a parte mais complicada, pois existem vários fatores a se observar: todos os pedidos tem de estar carregados nos veículos, deve ser utilizado o menor número de veículos possíveis, e, para isso deve-se haver o menor desperdício de espaço em cada veículo analisado. Também é imprescindível que a solução encontrada seja realmente a solução ótima, e, garantimos isso analisando todas as soluções e escolhendo a melhor entre elas.

A segunda parte da solução proposta cria uma lista de possíveis soluções ótimas para todos os veículos a partir das listas de soluções obtidas individualmente por veículo, da seguinte forma:

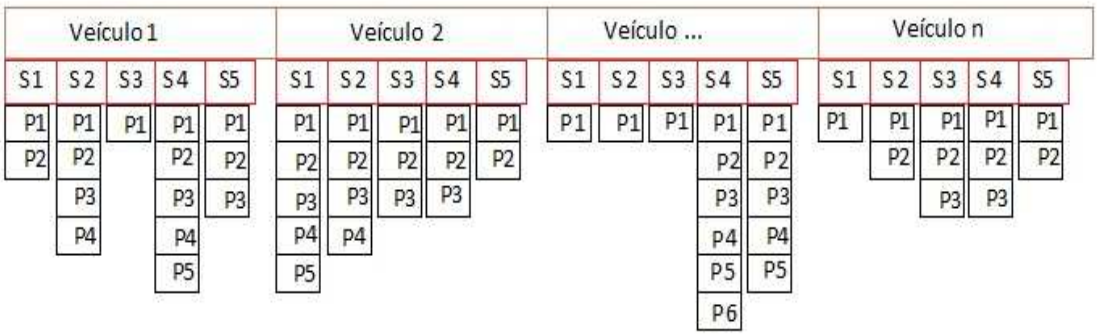
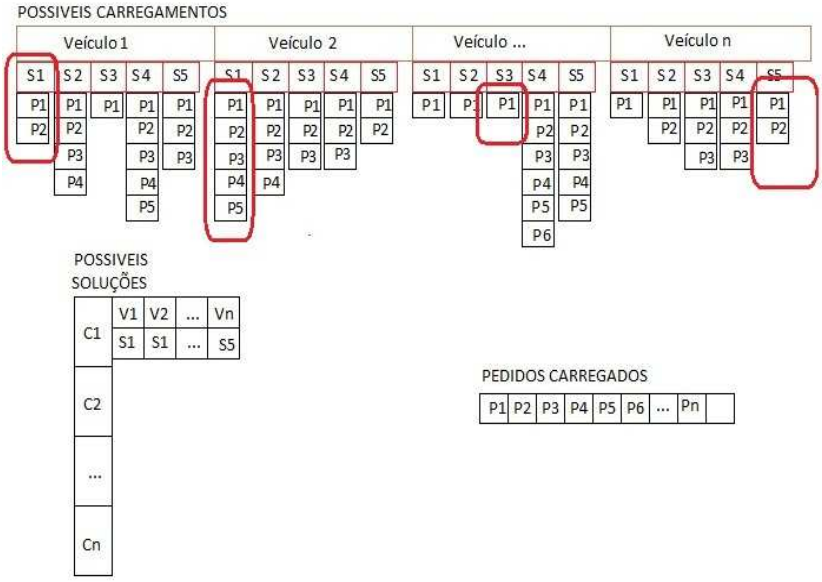


Figura 10: Lista de veículos e seus possíveis carregamentos.

Para cada veículo da lista aloca-se a primeira solução de sua lista individual de soluções, seus pedidos são gravados em uma lista de pedidos carregados. Em seguida segue-se para o segundo veículo e aloca-se a primeira solução que não coincida com os pedidos encontrados na lista de pedidos carregados, os pedidos da solução encontrada são gravados na lista de pedidos. Repete-se a operação até que todos os pedidos estejam na lista de pedidos carregados.

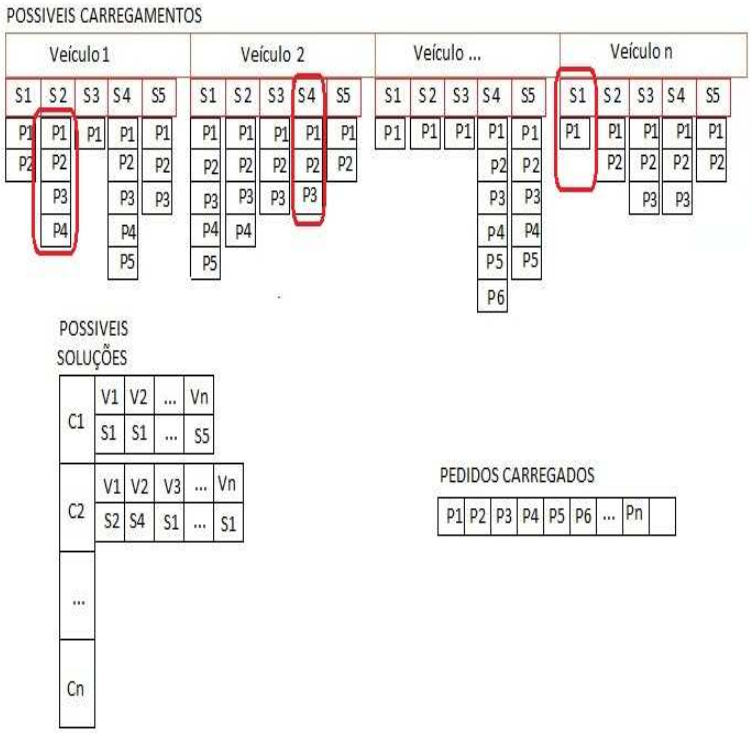


Construção da lista de possíveis soluções finais.

Note que as soluções contém o veículo e o número da solução na lista individual de soluções.

Figura 11: Modelo de formação das listas com os veículos carregados

O próximo item da lista de possíveis soluções é obtido da mesma forma, porém, o primeiro item alocado é a segunda solução possível do primeiro veículo. Isso se repete sucessivamente até que a lista seja começada pela última solução do último veículo. Isso garante que todas as combinações de carregamentos sejam analisadas pelo algoritmo. Como todas as combinações de carregamentos ficam gravadas em uma lista de possíveis soluções o algoritmo termina procurando nesta lista a combinação que minimiza a quantidade de veículos para o carregamento.



A cada solução alocada na lista de possíveis soluções os seus respectivos pedidos são gravados em uma lista de pedidos carregados e quando todos os pedidos realizados se encontram nesta lista a mesma é zerada e começa a análise de uma nova solução.

Figura 12: Modelo de formação das listas com os veículos carregados em outro estágio

Para garantir o resultado ótimo foi preciso criar uma lista individual para cada veículo das possíveis soluções, e em seguida criar outra lista com as possíveis

combinações destas soluções individuais e só depois encontrar a combinação ótima dos carregamentos individuais. Neste caso a solução ótima é a que minimiza a utilização de veículos

Esta solução sem dúvida nenhuma tem um custo computacional muito alto, uma vez que o mesmo possui mecanismos de otimização que como já dito anteriormente possui uma complexidade muito elevada.

O algoritmo gera uma lista de possíveis carregamentos para cada veículo, para isso utiliza-se de um método guloso que repetido "N" (N é o número de total de pedidos) vezes gera todas as combinações de pedidos, este é um algoritmo de combinação, e sua formula é descrita por:

$$\sum_{k=1}^{n} \frac{n!}{k!(n-k)!}$$ *N é o número total dos pedidos*

Como este algoritmo é repetido para todos dos veículos tem-se então:

$$v\left(\sum_{k=1}^{n} \frac{n!}{k!(n-k)!}\right)$$ *v é o número de veículos*

Isso sem tratar da otimização, que além de se utilizar do processo de combinação das soluções geradas pelo algoritmo anterior o faz atuando sobre uma entrada que já foi exponencialmente aumentada devido a atuação do algoritmo que gera os possíveis carregamentos por veículo.

Logo se assume que o resultado do algoritmo anterior fosse obtido em função de uma variável "*C*" ter-se-ia:

$$C = \left[v\left(\sum_{k=1}^{n} \frac{n!}{k!(n-k)!} \right) \right]$$

E como situação final:

$$v\left(\sum_{k=1}^{c} \frac{C!}{k!(C-k)!} \right)$$

Não é necessário dizer que tal complexidade seria impraticável.

Como visto o algoritmo de Força Bruta não é viável em um ambiente prático, para este fim tem-se que recorrer a soluções mais simples que obtenham um bom desempenho, mesmo que não garantam o melhor resultado.

Historicamente problemas compatíveis ao Problema da mochila têm sido resolvidos com técnicas como heurísticas de aproximação, programação linear, programação dinâmica, entre outras. Técnicas essas que mesmo não garantindo o melhor resultado garantem um bom desempenho, até mesmo em entradas com um grande volume de dados.

Para isso propôs-se uma abordagem baseada em Programação Dinâmica, que define que os problemas devem ser divididos em subproblemas menores e depois agrupados em uma solução final. Esta se baseia no Princípio da Otimalidade de Bellman, que diz que uma solução ótima pode ser conseguida través de sub-soluções ótimas extraídas de sub-problemas do problema principal.

Adotando esse princípio propõe-se o seguinte:

ALGORITMO 8 – PROPOSTA DE SOLUÇÃO (Primeira Abordagem)

```
1 void carrega()
2 {     for( i=0; i<=v; i++)
3    { cond =1;
4         p=0;
5         capacidade = veículos[i].cap;
6         carga = 0;
7         while (cond>0)
8         {  for ( j=0; j<=n; j++)
9             if (carga < veículos[i].cap)
10              if (veículos[i].carga[p] == 0)
11                if (pedidos[j].carga == 0)
12                  if (capacidade == pedidos[j].pes)
13                  {pedidos[j].carga = veículos[i].num;
14                   veículos[i].carga[p] = pedidos[j].id;
15                   carga = carga + pedidos[j].pes;
16                   p++;
17                   capacidade = capacidade -carga;
18                            }
19                  if (carga >= veículos[i].cap)
20                           cond = 0;
21                  else
22                      if (capacidade < 1)  cond = 0;
23                      else capacidade = capacidade /2;
24                 veículos[i].cartot = carga;
25         }
26     }
27 }
```

O algoritmo acima é baseado em Programação Dinâmica, e, é apenas uma abordagem inicial para o problema. Ele procura na lista de pedidos (itens para a mochila) itens cujos valores sejam iguais ao valor total da capacidade do veículo a ser carregado, se este valor for encontrado o veículo é carregado em sua totalidade pelo pedido em questão; senão a capacidade do veículo é dividida em duas, e repete-se a operação tentando achar os valores para as capacidades divididas (mochilas menores). Caso existam pedidos que coincidam com as capacidades divididas dos veículos estes

são alocados e somados a um valor total da carga. O programa só para quando o valor total da carga for igual ou maior que a capacidade do veículo.

Como exposto anteriormente esta é apenas uma abordagem inicial para o problema, pois, há vários aspectos para serem observados, há limitações claras, pois, os valores resultantes da divisão das capacidades dos veículos são limitados pelos fatores usados para este fim, quando se divide uma capacidade ao meio, apenas valores resultantes da divisão por 2 serão encontrados e o mesmo ocorrerá com qualquer fator. Ha também a situação verificada quando não se encontram valores exatos para os pedidos mesmo depois de divididas as capacidades dos veículos, nestes dois casos haverão pedidos que não serão alocados, além de muito espaço de sobra nos veículos.

Dividir o problema em subproblemas menores não significa necessariamente que estes subproblemas sejam absolutamente iguais, pode-se abranger um número maior de possibilidades ao se enxergar o problema com uma visão menos linear, dividindo as capacidades dos veículos em partes que se adéqüem aos pedidos, já que os mesmos são indivisíveis para o caso específico do problema estudado.

Porém apenas ao dimensionar parcialmente as mochilas de acordo com o peso dos itens cria-se um algoritmo guloso, que, apesar de que ao descrever-lo nos termos acima ele parece com um método de programação dinâmica mas ainda continuaria sendo um método guloso.

A fim de evitar isso, será utilizado este método guloso como uma abordagem auxiliar, alocando gulosamente os itens no caso da falha em alocá-los dinamicamente, isso trará a solução uma maneira de relaxar a divisão exata proposta acima.

Tanto no caso da divisão linear apresentada algoritmo anterior quanto na divisão não linear quando se aloca um item na mochila a capacidade da mesma é igual a sua capacidade inicial menos os itens já alocados. Então a cada item alocado temos uma nova mochila para ser analisada.

Assim cria-se uma mochila nova a cada interação e as suas divisões podem ser melhor aproveitadas.

Para tal, propõe-se:

ALGORITMO 9: SEGUNDA ABORDAGEM DA SOLUÇÃO PROPOSTA

```
1 void carrega()
2 {   for (int i=1; i<v; i++)
3     {   c = d = 0;
4         cond = 1;
5         p=0;
6         capacidade = veículos[i].cap;
7         carga =0;
8         while (cond > 0)
9         { j = AlocaIgual(i,p,capacidade);
10          if (j>0)
11          { pedidos[j].carga = veículos[i].num;
12            veículos[i].carga[p] = pedidos[j].id;
13            veículos[i].cargat= veículos[i].cargat + pedidos[j].pes;
14            p++;
15            capacidade = veículos[i].cap-veículos[i].cargat;
16            if (capacidade <=0)
17               cond = 0;
18            if (veículos[i].cargat >= veículos[i].cap)
19              cond=0;
20            else
21            { j = AlocaMenor(i,p,capacidade);
22              if (j>0)
23              { pedidos[j].carga = veículos[i].num;
24                veículos[i].carga[p] = pedidos[j].id;
25                veículos[i].cargat=veículos[i].cargat+pedidos[j].pes
26                p++;
27              }
28            }
29             capacidade = veículos[i].cap-veículos[i].cargat;
30             if ((c>=n)&&(d>=n))
31                cond=0;
32        }
33    }
34 }
```

```
1 int AlocaIgual(int i, int p, float cap)
2 {       while (cond>0)
3         {       c++;
4                  for (int j=1; j<=n; j++)
5                  {     if (veículos[i].carga[p] == 0)
6                           if (pedidos[j].carga == 0)
7                              if (cap == pedidos[j].pes)
8                                 return j;
```

```
9                   }
10                   cap = cap/2;
11                  if (cap <1)
12                      cond = 0;
13          }
14          return 0;
15 }
```

```
1 int AlocaMenor(int i, int p, float capacidade)
2 {  d++;
3    for (int j=1; j<=n; j++)
4    {     if (veículos[i].carga[p] == 0)
5               if (pedidos[j].carga == 0)
6                   if (cap >= pedidos[j].pes)
7                                       return j;
8     }
9          return 0;
10 }
```

A estratégia de alocar um valor menor quando a “Mochila” não for completada por inteiro na tentativa inicial alocando valores exatos para seu tamanho seja total ou dividido, traz a possibilidade de novas “mochilas” com tamanhos não lineares aos da “Mochila” original.

O algoritmo apresentado acima traz duas novidades em relação ao algoritmo anterior: a primeira é o desmembramento da função que calcula se existe um pedido igual ao valor total da capacidade do veículo analisado, ou a alguma fração do mesmo. E a novidade maior é o acréscimo de mais uma função que aloca o valor do menor pedido no veículo caso a primeira tentativa não tenha sido bem sucedida.

Logo no inicio do algoritmo existe uma chamada a função **AlocaIgual()** “`j = AlocaIgual(i,p,capacidade);`” esta função percorre o vetor buscando algum pedido que seja igual a capacidade atual do veículo, caso esta seja encontrada a função retorna a posição do pedido na lista para que seja alocada na solução através da função **Carrega()**. Caso não seja encontrado pedidos cujo peso seja igual a capacidade atual do veículo, esta última será dividida e a operação se repetirá, procurando valores iguais e dividindo as capacidades até que a condição de parada seja atingida.

Existem duas condições de parada para esta função: a primeira acontece quando é encontrado um pedido que seja igual à capacidade total atual da mochila, onde é retornada a posição do pedido; a segunda é quando a capacidade após divida atinge um valor menor que 1 que é referente a 1 tonelada, valor mínimo para os pedidos, neste caso a função retorna 0.

Em seguida a função **Carrega()** testa o retorno da função **AlocaIgual()** contido na variável "**j**" caso esta contenha um valor maior que 0 "zero", indica que foi encontrado um valor a ser alocado, proceder-se então a alocação do pedido no respectivo veículo e o abatimento do valor do pedido na capacidade total do veículo. Caso o valor encontrado seja 0 "zero", é chamada a função **AlocaMenor()**. E, esta última, percorre o vetor de pedidos e ao encontrar o menor pedido retorna seu valor para a função **Carrega()**.

A decisão pelo menor valor no para a alocação de pedidos não iguais ao valor total dos pedidos, deve-se do fato de ao se alocar um valor menor a probabilidade de um valor maior igual a capacidade restante ser encontrado e muito maior que no caso da opção pelo contrario. Além disso, com uma capacidade restante maior existe uma margem para que a busca através do método de programação obtenha sucesso, de maneira a minimizar a atuação do método guloso na solução.

Este processo é repetido diversas vezes até que uma das condições de parada seja a atingida, condições estas que são: a capacidade do veículo ser igual a 0 "zero" ou sua carga for maior ou igual a capacidade, esta por razões obvias; ou a soma das interações for igual ao tamanho da lista de pedidos, pois após tal quantidades interações as mesmas estariam se repetindo sem sucesso.

A solução utiliza uma estrutura construída de maneira a minimizar as possíveis falhas do algoritmo e aumentar o controle sobre suas interações, bem como possibilitar alternativas à construção de outras estruturas para a apresentação dos resultados:

```
1 struct StPedidos{
2         int id;
3         float pes;
4         int carga;};
```

```
1 struct StVeículos{
2         int num;
3         float cap,cargat;
4         int carga[8];};
```

Sob a ótica do controle sobre as interações do programa, note que na estrutura dos veículos existe uma variável ***cargat***, que é referente à carga total do veículo, esta recebe o valor carregado a cada interação do programa e é comparada com a capacidade do veículo para se encontrar a condição de parada para o programa.

Como alternativas para a apresentação da saída vê-se também na estrutura referente aos veículos existe um vetor com o nome de ***carga[]***, onde são marcados os identificadores de cada pedido alocado no veículo analisado. Na estrutura dos pedidos há também uma variável com o nome de ***carga***, esta se encontra ali para que quando o pedido for alocado em um veículo, a mesma receberá o número do veículo ao qual este pedido pertencerá. Isto serve tanto para controle de iterações, já que quando todos os pedidos tiverem valores diferentes de 0 “zero” para esta variável o programa terminará; quanto para apresentação na saída do programa.

É de suma importância lembrar que a capacidade que é dividida nas interações da função **AlocaIgual()** não se refere à capacidade útil restante para o carregamento do veículo, esta última só é alterada quando da alocação efetiva de um pedido. Sendo, portanto, dividida apenas uma variável auxiliar, gerando assim uma sub-solução, que é imediatamente incrementada à solução final.

Este algoritmo aloca todos os pedidos nos veículos da lista, e, além disto, consegue fazê-lo com bastante eficiência seja em relação tempo e custo computacional, ou seja, em relação à quantidade de veículos utilizados. Porém foram detectadas algumas melhorias que podem ser de muita importância para a solução final.

Em primeiro lugar foi verificado que quando a lista de pedidos está ordenada é necessário um menor esforço por parte da solução, pois sempre é necessário encontrar o menor pedido disponível, trabalho este, que é extremamente facilitado quando uma lista está ordenada encontrar o menor é uma tarefa óbvia, e muito mais fácil do que quando realizada em uma lista desordenada.

Organizou-se também a lista de veículos, pois através de testes foi verificado que ao começar pelos veículos com maior capacidade obteve-se um melhor resultado quanto ao número de veículos utilizados.

Para tal propõe-se:

ALGORITMO 10: ORDENAÇÃO DOS VEÍCULOS

```
1 void organizavei()
2  {  int i, j, auxnum;
3     float auxcap;
4     for (i=0; i<=v-1; i++)
5       for (j=i; j<=v; j++)
6           if (veículos[i].cap > 0)
7           { if (veículos[i].cap < veículos[j].cap)
8              { auxcap = veículos[i].cap;
9                auxnum = veículos[i].num;
10               veículos[i].cap = veículos[j].cap;
11               veículos[i].num = veículos[j].num;
12               veículos[j].cap = auxcap;
13               veículos[j].num = auxnum;
14           }
15       }
16 }
```

ALGORITMO 11: ORDENAÇÃO DOS PEDIDOS

```
1  void organizaped()
2  {  int i, j, auxnum;
3      float auxcap;
4      for (i=0; i<=n-1; i++)
```

```
5         for (j=i; j<=n; j++)
6            if (pedidos[i].pes > 0)
7            { if (pedidos[i].pes < pedidos[j].pes)
8               { auxcap = pedidos[i].pes;
9                 auxnum = pedidos[i].id;
10                pedidos[i].pes = pedidos[j].pes;
11                pedidos[i].id = pedidos[j].id;
12                pedidos[j].pes= auxcap;
13                pedidos[j].id = auxnum;
14             }
15          }
16  }
```

Os dois algoritmos (algoritmos 10 e 11) são simples de ordenação que utilizam o método de seleção para realizar esta tarefa, e possuem complexidade polinomial, o que não aumenta o custo computacional do programa, e ajudam bastante em outras partes da solução. Como existem características particulares no problema há também necessidades especiais, dado a isso, a necessidade da ordenação das entradas se torna crítica.

Com relação aos resultados obtidos com esta técnica, verifica-se um ótimo desempenho tendo como ponto de vista principal a otimização do número de veículos. Porém há casos em que esta estratégia tem algumas falhas. Um exemplo é o caso descrito abaixo onde os pedidos foram alocados em dois veículos, de capacidade 10 ton e poderiam ser alocados em veiculos de capacidades menores diminuído assim o custo do transporte.

Para ilustrar este raciocino temos:

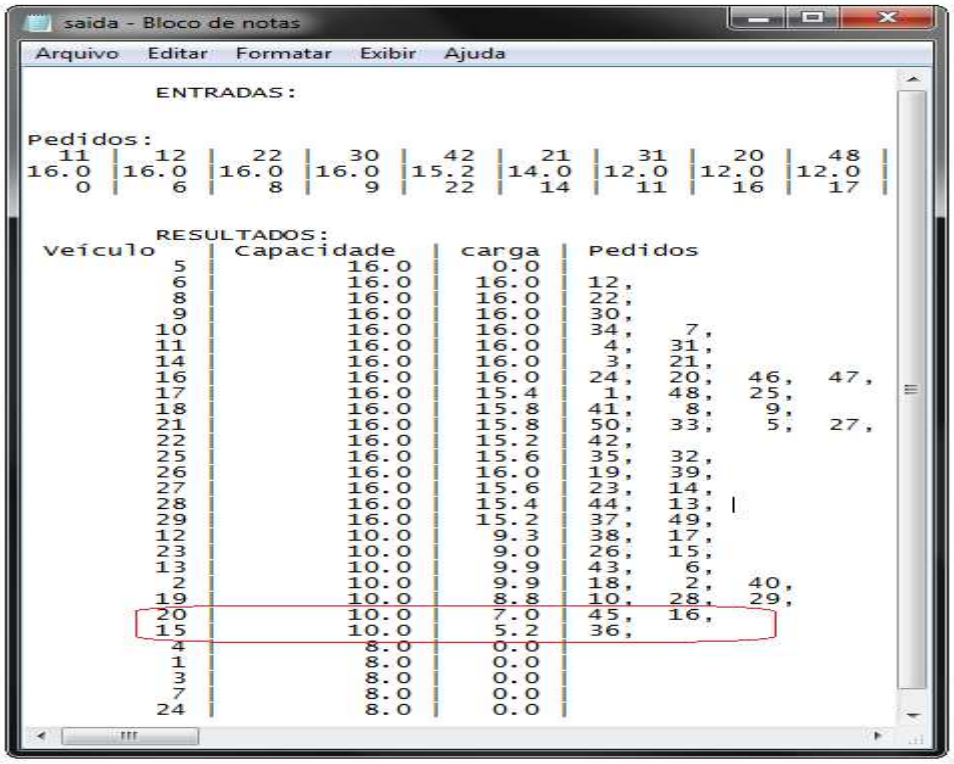


Figura 12: Falha de otimização na alocação dos pedidos nos veículos

Este resultado demonstrou que a solução apresenta um bom resultado quando a questão é apenas a quantidade de veículos utilizados, mas em relação ao desperdício relativo de espaço nos veículos torna-se um fator de grande relevância.

Os veículos de número 20 e 15 são veículos que possuem 10 toneladas de capacidade, capacidade esta que está pouco aproveitada com 7,0 e 5,2 toneladas de carga em cada veículos respectivamente, provocando assim um desperdício de quase 50% da capacidade dos veículos.

Se for levado em consideração que há ainda vários veículos de 8 toneladas que não estão sendo aproveitados, e que o transporte em um veículo de 10 toneladas possui um custo maior o mesmo transporte em um veículo de 8 toneladas, percebe-se que há uma possibilidade de melhoria neste processo.

Visando este fim a proposta é uma função que faz um relaxamento da solução final e verificam-se estes detalhes importantes na solução final, e realizam-se as principais correções, aproximando-se mais da solução ótima que é o objetivo final.

A verificação da solução se dá assim:

ALGORITMO 12: VERIFICAÇÃO DAS SOLUÇÕES

```
1 void corrige_carga()
2 {   int cont;
3     int cont1=0;
4     for (int i=0; i<v; i++)
5       if (veículos[i].cargat < veículos[i].cap)
6       {for (int j=0; j<v; j++)
7             if ((veículos[j].cargat == 0) && (veículos[i].cap >
        veículos[j].cap) && (veículos[i].cargat <= veículos[j].cap))
8        {  cont=0;
9           for (int l=0; l<8; l++)
10          {  veículos[j].carga[l] = veículos[i].carga[l];
11             veículos[i].carga[l] = 0;
13          }
13          veículos[j].cargat = veículos[i].cargat;
14          veículos[i].cargat = 0;
15          cont++;
16       }
17       cont1++;
18      }
19 }
```

No âmbito da solução esta correção elimina a possibilidade de ocorrer o problema citado anteriormente, situação esta, pode ocorrer tanto com veículos de menor quanto de maior capacidade.

A correção apresentada atua verificando para cada veículo, se a carga total é menor que a carga carregada neste veículo. Em seguida se o teste for verdadeiro, percorre-se a lista de veículos procurando um veículo vazio cuja capacidade seja menor que a do veículo analisado e que tenha uma capacidade maior que a carga a ser movida.

Em seguida caso estes testes se confirmem a carga do veículo analisada é transferida para o outro veículo, deixando o veículo de maior capacidade disponível e aproveitando melhor a relação carga/capacidade dos veículos menores.

Com o acréscimo desta última função a saída para o problema anterior se apresenta da seguinte forma:

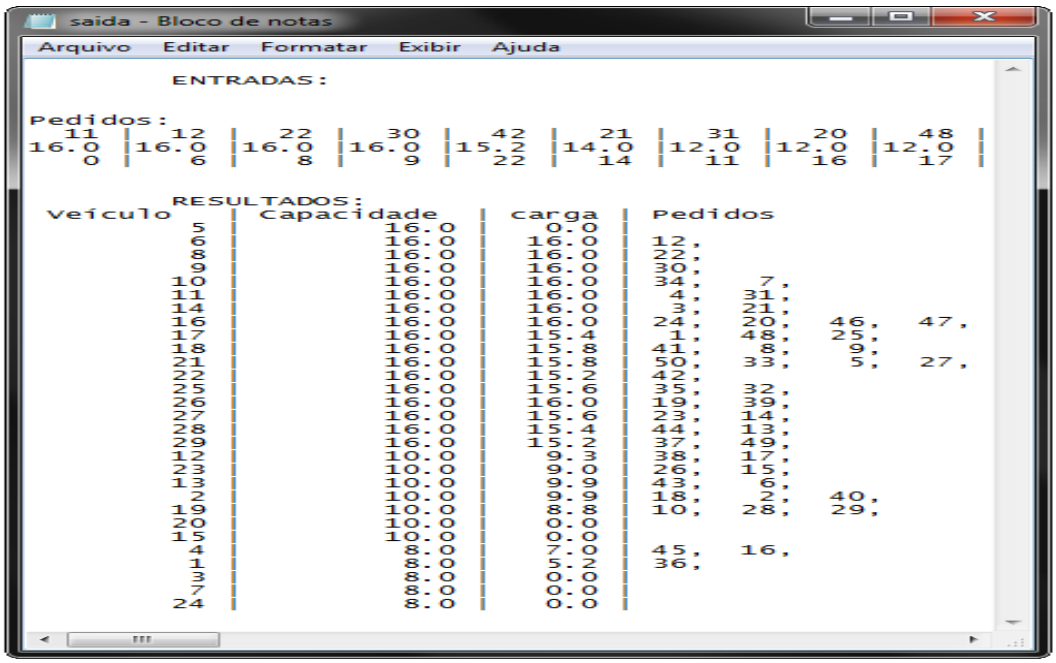


Figura 13: Visualização do arquivo de saída da versão final da solução proposta

Note que após a adição da correção dos carregamentos os carregamentos que poderiam ser aproveitados em veículos menores foram transferidos para estes veículos, reduzindo, portanto, o desperdício de espaço, aumentado assim a eficiência da solução.

Foi apresentada uma solução que se baseia nos conceitos de programação dinâmica que, juntamente com outras técnicas obteve um bom desempenho junto ao problema.

Na formatação da solução final apresentou-se um algoritmo que é formado pelas funções que “carregam” os veículos ajudadas pelas funções de ordenação e a função que corrige o carregamento.

Analisando cada função separadamente em ordem de complexidade tem-se:

A função **carrega()** é formada por loop's sucessivos e alinhados ente si um deles um loop "***for"*** (linhas 2 a 33) este com complexidade F(v), onde v é o número de veículos e um loop "***while"*** (linhas 8 a 32) tendo como condição de parada o comando da linha 30 "*if ((c>=n)&&(d>=n))*" que define o tamanho do loop como sendo F(2n) onde n é o tamanho da lista de pedidos

Até agora se tem F(v)+F(2n), porém nesta função temos a chamada de duas outras a função **AlocaIgual()** e a função **AlocaMenor().** Analisando a complexidade de cada uma vimos que a função **AlocaIgual()** é formada por um laço "***while"*** e um aço for alinhados, onde o laço "***for"*** tem o tamanho n (tamanho da lista de pedidos) e o laço ***"while"*** tem como condição de parada o número de divisões sucessivas sobre valor da capacidade do veículo analisado para tal temos como função de complexidade F((n)log(c)). Já na função **AlocaMenor()**, é formada apenas por um laço "***for"***, e tem como seu piro caso F(n).

Logo se chega à conclusão da função de complexidade seria dada por: F(v)+F(2n)+F((n)log(c))+F(n), que por sua vez pertence à ordem de complexidade polinomial O(vn).

A solução final usa ainda duas funções de ordenação que são igualmente formadas tendo como única diferença estrutural entre si a referencia aos veículos e pedidos ambas são formadas por dois laços "***for"*** alinhados entre si tendo cada um o tamanho da lista referente à respectiva função logo a complexidade de cada uma é dada por $F(n^2)$ para os pedidos e $F(v^2)$ para os veículos.

No caso da função **Corrige_Carga()** ela segue o mesmo padrão das funções de ordenação acima descritas, esta também é formada por dois laços "***for"*** alinhados que percorrem a lista de veículos afim de encontrar cargas que sejam incompatíveis com a capacidade do mesmo e tendo como função de complexidade $F(v^2)$

Por fim analisando-se a complexidade da solução final podemos dizer que temos:

$=F(v)+F(2n)+F((n)\log(c))+F(n)+ F(n^2)+F(v^2)+F(v^2)$

$=F(v+2n+(n)\log(c)+n+n^2+v^2+v^2)$

$=F(v+3n+n^2+2v^2+(n)\log(c))$

O que nos demonstra um algoritmo com complexidade na ordem: $\mathbf{O(v^2n^2)}$, algoritmo polinomial em função das entradas v e n ou seja em função do tamanho das listas de pedidos e veículos.

3.3. Metodologia da Analise dos Resultados.

A análise dos resultados realizada neste trabalho tem como fundamento dois métodos principais. Inicialmente, foi feita a comparação com situações e casos de usos que colocam o algoritmo frente não apenas a situações encontradas em nas condições reais de uso, comparando e analisando os resultados estaticamente e demonstrando-os por meio de tabelas e gráficos. Além desta abordagem, as comparações também serão feitas com outros algoritmos e com outras soluções, que através desta comparação, descreverão a solução em função de suas qualidades e falhas.

Em relação à comparação com outros algoritmos descritos na literatura, comparou-se a solução apresentada com um algoritmo baseado em um Método Guloso,

e com um algoritmo que utiliza o Método de força bruta, esse último já foi citado anteriormente.

Também foi comparada a solução com o método utilizado atualmente sem nenhuma intervenção automática. Neste caso a comparação não foi em função do desempenho, que obviamente é superior por parte do método automático, e sim, a comparação será em função dos resultados obtidos levando em conta a otimização do processo, da redução na utilização de recursos.

Pediu-se à empresa que realizasse o processo normal de alocação de cargas e nos disponibiliza-se o resultado, marcando o tempo de cada processo. Foram levantados alguns grupos de pedidos e veículos, e, as posteriores soluções, conseguidas através do método tradicional. Levando em consideração que cada entrada é relativa a uma região de atuação da empresa, consegui-se na ocasião um considerável grupo de dados, os quais foram submetidos aos testes com a solução proposta dos quais se extraiu resultados, que posteriormente foram comparados com os obtidos com o método tradicional.

Todos os testes foram feitos em uma máquina compatível com as máquinas disponíveis na empresa estudada. Para tal fim usou-se um computador com um processador AMD Athlon X2 64, com 2GHZ de freqüência de Clock e 3 Gbytes de memória, em um sistema operacional Windows XP SP2, esta opção foi feita tendo como objetivo uma maior aproximação à realidade com a qual a solução se deparará no dia a dia.

4, ANALISE E DISCUSSÃO DE RESULTADOS

4.1 Análise em relação a outros algoritmos

Quando uma solução é comparada com outro algoritmo deve-se ter em consideração o desempenho e a qualidade da solução. Em relação ao desempenho podem ser feitos tantos testes práticos quanto análises feitas em função da comparação de suas complexidades, e previsões em função das mesmas.

4.1.1 Relação com o Algoritmo Força Bruta

Este algoritmo tem função crucial na análise dos resultados quando se trata de problemas de otimização, pois, este garante sempre o resultado ótimo, servindo assim para análise da qualidade dos resultados obtidos.

Em contraponto este é um algoritmo que praticamente impossibilita que estas análises sejam realmente abrangentes, devido a sua “enorme” complexidade computacional. Esta é a razão que limita as ações de comparação, pois, como esta solução consome muitos recursos computacionais e necessita de muito tempo para o retorno da resposta apenas se torna minimamente viável em entradas muito pequenas, levando ainda em consideração que quanto maiores às entradas mais recursos computacionais e conseqüentemente tempo se fazem necessários.

Em relação ao desempenho na obtenção da resposta a solução proposta se mostrou extremamente mais satisfatória que a solução baseada no algoritmo de Força

Bruta, isso se deve à complexidade exponencial/combinatória do algoritmo de Força Bruta que em comparação com a complexidade polinomial da solução apresentada.

No tangente à qualidade da solução foram propostas cinco situações para a comparação da solução, estas situações são representadas por entradas que formam retiradas diretamente de a empresa estudada podendo-se assim avaliar os resultados em uma condição de trabalho real.

A tabela abaixo apresenta os resultados obtidos com as duas soluções:

Entradas		**Nº de veículos carregados**		**Tempo de execução - segundos**		**Carga total perdida**			
Nº de Veículos	**Nº de Pedidos**	**Força Bruta**	**Solução Proposta**	**Força Bruta**	**Solução Proposta**	**Força Bruta**		**Solução Proposta**	
						Ton.	**%**	**Ton.**	**%**
5	**8**	5	5	2,30	0,50	4,1	1,8	4,3	1,89
5	**14**	4	4	35,70	0,50	2,5	1,1	2,5	1,1
10	**20**	8	8	92,20	0,70	3,8	3,8	3,8	3,8
10	**35**	9	10	257,40	0,90	1	1	5	5
15	**32**	11	11	642,20	0,90	4,2	3,33	4,2	3,33

Tabela 1

Ao levar em consideração que o algoritmo Força bruta apresenta sempre a melhor solução podemos concluir que a proposta apresentada tem um bom desempenho, podemos ver também que a mesma possui falhas, algumas delas inconcebíveis.

Analisando os resultados mais minuciosamente pode-se chegar a conclusões importantes, em princípio se analisou a qualidade do resultado da proposta apresentada em função do algoritmo Força Bruta tendo sempre em mente que este último garante sempre a melhor solução, nos resultados apresentados as soluções foram muito aproximadas, coincidindo várias vezes, levando a uma conclusão inicial de que o resultado deste algoritmo é muito bom para as entradas propostas.

Porém esta primeira impressão é apenas um engano, pois, é sabido que apesar do Princípio da Otimalidade de Bellman ser válido, esta solução não aplica apenas sub-soluções exatas e lineares, há durante o processo uma abordagem gulosa o que compromete a otimalidade da solução. Além disso, também já foi discutido anteriormente a limitação desta solução em encontrar sub-soluções a partir da divisão exata da capacidade dos veículos, e como a solução visa minimizar o número de veículos contratados, ainda existem lacunas a serem preenchidas

No entanto, não é só de falhas e lacunas que este resultado se apresenta, a solução proposta teve um ótimo desempenho neste teste, obtendo um nível de otimalidade de aproximadamente 80%, conforme demonstrado no gráfico abaixo, o que é muito bom principalmente quando os resultados provêm de testes tão limitados.

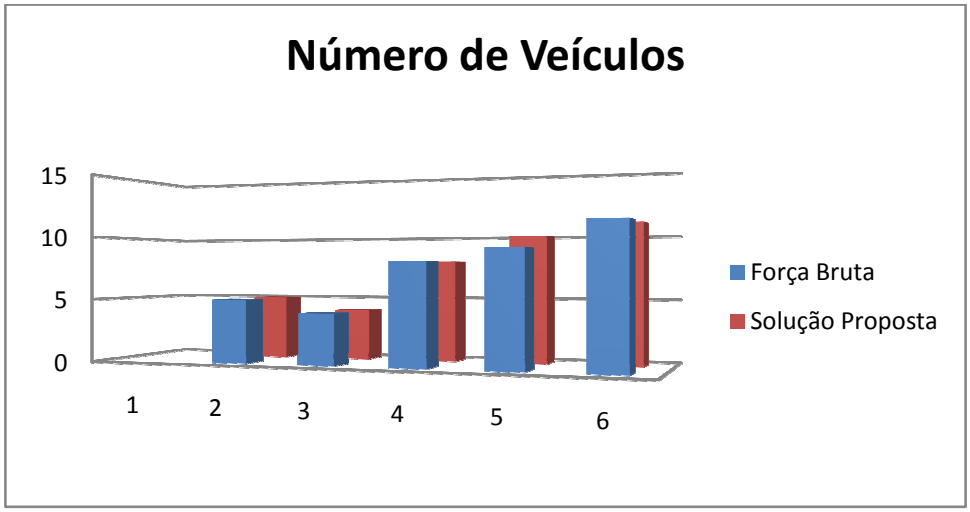


Gráfico 1

“Limitações”, esta é a expressão para definir esta bateria de testes, e, isso se deve principalmente ao altíssimo custo computacional do algoritmo de Força Bruta, pois, é praticamente impossível fazer testes nesse tipo de algoritmo utilizando entradas de um tamanho considerável. Na avaliação em relação ao tempo que cada algoritmo

utilizou na construção de sua resposta esta situação fica bem clara, pois, os tempos obtidos com a solução proposta foram do mínimo de 0,2 ao máximo 0,9 segundos, enquanto o algoritmo Força Bruta obteve um mínimo de 2,3 segundos e um máximo de 642,2. A diferença de desempenho é realmente gritante, e se levarmos em consideração a curva de crescimento do tempo consumido em relação à entrada, vemos que o algoritmo de Força bruta é extremamente ineficiente e literalmente impraticável em termos práticos.

É interessante também registrar que as entradas foram cuidadosamente escolhidas entre todas as opções de entradas reais, foram escolhidas entradas cujo número de veículos e de pedidos fossem inicialmente pequenos e depois fossem aumentado de forma a dar uma noção quantitativa do comportamento do algoritmo em relação a entradas de tamanhos variados e crescentes.

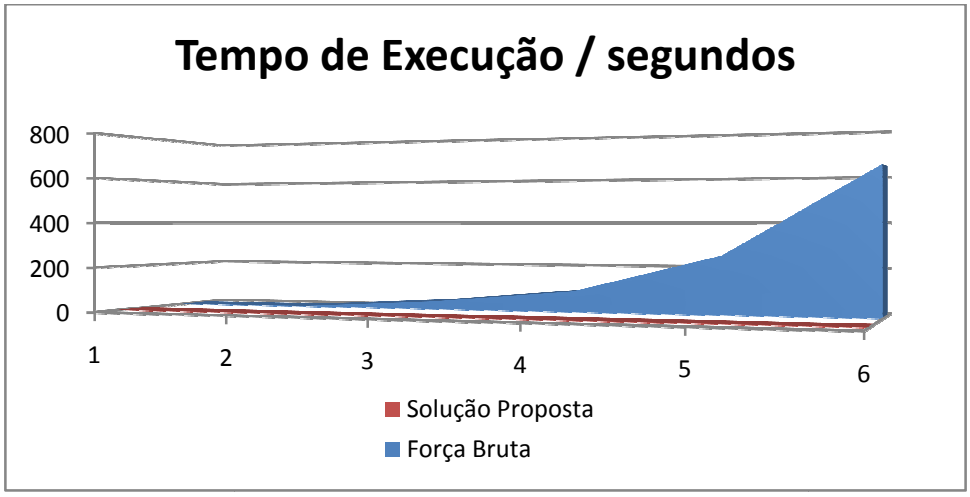


Gráfico 2

O gráfico acima mostra a enorme discrepância entre os tempos de execução do algoritmo da proposta de solução apresentada com complexidade computacional polinomial, comportando-se quase linearmente em relação às entradas crescentes e o algoritmo de Força bruta com complexidade de tempo combinatória.

Um dado interessante coletado neste teste é o desperdício de espaço total no carregamento dos veículos, visto que foi observado uma íntima relação deste dado com a otimização do número de veículos utilizados a qual será discutida posteriormente.

Notou-se que quando o desperdício é reduzido ao mínimo aceitável é também reduzido o número de veículos carregados, isso se deve, porque o desperdício de espaço também se coloca como um dado de otimização, já que aproveitando o máximo de espaço de um veículo tem-se uma quantidade menor de mercadoria para ser carregada nos próximos.

Apesar de óbvio, este raciocínio não é simples de ser executado e conseguido por meios automáticos, e, como já comentado, apenas o algoritmo de Força Bruta, que tem um custo muito auto, é capaz de garanti-lo.

4.1.2 Relação com o Método Guloso

O Método Guloso é também um clássico da análise de algoritmos, que é usado em quase todos os problemas, principalmente quando se trata de problemas cujas entradas são pequenas ou não existe a exigência de que se encontre uma solução ótima.

Este método baseia-se na idéia de escolher a melhor opção para o passo atual da solução, não se importando se a mesma é a melhor solução para o conjunto que formará a possível solução para o problema.

Fica óbvio que este método nem sempre fornece a melhor solução, porém, ela apresenta um ótimo desempenho em função da complexidade computacional é um dos melhores entre os métodos existentes para solução deste tipo de problema.

Levando em conta estas considerações, foi possível traçar um paralelo entre os resultados apresentados pela solução proposta e os conseguidos através de uma solução baseada em um método guloso.

Para tal analise temos:

Entradas		Nº de veículos carregados		Tempo de execução - segundos		Carga total perdida			
Nº de Veículos	Nº de Pedidos	Alg. Guloso	Solução Proposta	Alg. Guloso	Solução Proposta	Alg. Guloso		Solução Proposta	
						Ton.	%	Ton.	%
20	**37**	17	15	0,4	0,9	4,5	2,79	4,1	2,30
12	**29**	10	10	0,4	0,9	4,2	3,38	4,2	3,38
26	**43**	19	15	0,5	1,1	4,7	4,81	4,1	4,13
10	**20**	10	8	0,2	0,5	4,1	4,03	2,2	2,24
29	**46**	23	17	0,5	1,1	4,2	5,09	3,1	1,14

Tabela 2

Em relação à qualidade da solução apresentada, representada pelo nível de otimização da quantidade de veículos utilizados para um grupo de pedidos, podemos observar que a solução proposta se saiu melhor em praticamente todos os casos deste comparativo.

Este resultado tem uma íntima relação com a questão abordada a pouco, pois o Método Guloso, ao escolher a melhor solução com uma visão imediata não se preocupa com o problema como um todo, dando uma atenção especial a mínimos ou máximos locais, que, nem sempre são a melhor solução.

Outra questão que contribui para a má qualidade da resposta provida pelo algoritmo Guloso está relacionada à entrada, pois o problema é distribuir os pedidos entre os vários veículos, logo temos vários itens para várias mochilas, e neste caso

quando se aloca um pedido em um veículo, ele poderia ser de grande utilidade em outra carga. Passo que não é observado pelo algoritmo Guloso.

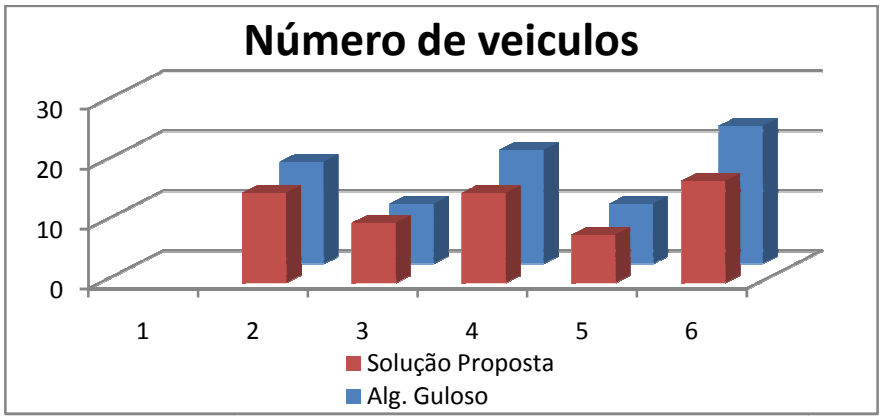


Gráfico 4

O gráfico acima demonstra a diferença já citada em relação à resposta do problema, diferença da quantidade de veículos carregados entre as duas soluções propostas se posicionou em uma média de 17,72% a favor da solução apresentada. Visto ainda que a solução proposta apresentou um melhor resultado em todas as instancias do teste.

Como dito acima o algoritmo guloso não é o mais indicado devido a configuração da entrada, pois, o problema apresenta várias mochilas para se carregar, fato que qualquer solução deve considerar.

Em relação ao desempenho, a o ponto mais relevante é o quão próximo a solução proposta chegou da solução construída a partir do Método Guloso, e para esta analise segue o gráfico:

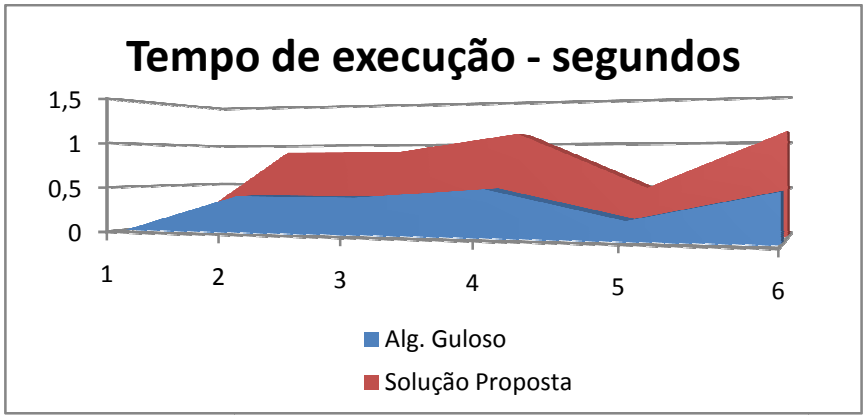


Gráfico 5

É visível a qualidade do desempenho do algoritmo guloso, é fácil também chegarmos à conclusão que existe uma relação ao tempo computacional e a qualidade do algoritmo. Isto na realidade é apenas uma impressão que se tem quando se analisa um grupo reduzido de algoritmos, pois existem casos em que os métodos gulosos são os mais apropriados, como é o caso do problema da Arvore Geradora Mínima (a definição do problema foge ao escopo do trabalho) onde o algoritmo de Cruscal e o algoritmo de Prim, Métodos Gulosos, fornecem sempre a solução ótima.

Em relação à comparação entre os tempos das duas soluções, é visível o melhor desempenho da proposta baseada no Método Guloso, que foi em média 55% mais eficiente. Porém este é um dado que fica minimizado se levarmos em conta os tempos reduzidos que foram observados nos dois casos, onde o pior resultado não passou de 1,1 segundos, um tempo muito pequeno principalmente em comparação com o método utilizado atualmente que é inteiramente manual o qual discutiremos a seguir.

Logo em relação ao tempo temos um melhor desempenho do algoritmo baseado em método guloso, e com relação à qualidade da solução temos uma clara vantagem por parte da solução proposta.

4.1.3 Comparativo com a Solução Manual.

Este é um comparativo onde o que realmente importa é exclusivamente a qualidade da solução, pois os tempos necessários para a obtenção da solução são muito discrepantes. O que já era de se esperar já que conseguir uma solução manualmente é algo muito complexo, pois, quem realiza esta árdua tarefa precisa de recorrentes e sucessivas consultas aos dados de entrada, este trabalho é além de desgastante e enfadonho, extremamente moroso, consumindo um grande tempo por parte dos responsáveis por sua execução.

Seguindo ainda a linha de raciocínio que descreve a análise das soluções em função da qualidade da solução obtida, percebemos que este é sim o dado mais importante do ponto de vista da empresa, pois para os principais interessados é este dado que define se a solução é ou não viável.

Para esse fim, preparamos também uma bateria de comparações baseadas em situações reais de definição das cargas, e chegou-se a estes dados:

Entradas		**Nº de veículos carregados**		**Tempo de execução - segundos**		**Carga total perdida**			
Nº de Veículos	**Nº de Pedidos**	**Solucionado manualmente**	**Solução Proposta**	**Solucionado manualmente**	**Solução Proposta**	**Solução Manual**		**Solução Proposta**	
						ton	**%**	**Ton**	**%**
20	**37**	16	15	1200	0,9	3	1,59	4,1	2,30
12	**29**	10	10	720	0,9	0,2	0,16	4,2	3,38
26	**43**	14	15	1500	1,1	0,1	0,05	4,1	4,13
10	**20**	8	8	900	0,5	2,2	2,24	2,2	2,24
29	**46**	19	17	1800	1,1	1,1	0,47	3,1	1,14

Tabela 3

Este comparativo nos mostra com clareza que para se conseguir o mesmo resultado conseguido pela solução proposta é necessário um tempo muito maior quando se utiliza a solução manual.

Isto já está certo, e confirmado. Por isso que o dado mais importante para este comparativo é com certeza o número de veículos utilizados.

O gráfico abaixo pode nos trazer uma perspectiva mais completa.

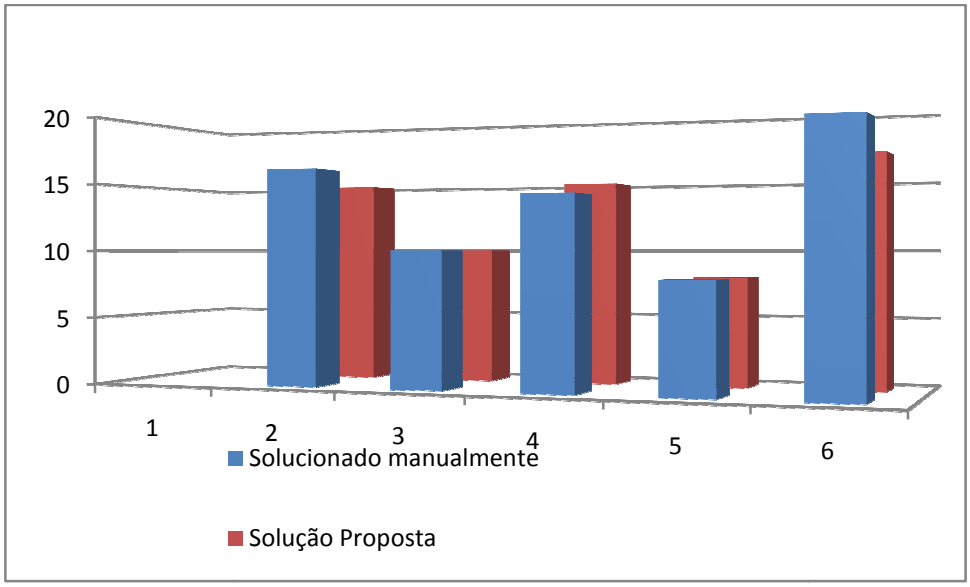


Gráfico 6

Nota-se que nas 5 situações analisadas tivemos duas em que a solução proposta teve um desempenho melhor, em outras duas situações, ambas as soluções obtiveram o mesmo resultado, sendo que a solução proposta apenas apresentou um resultado pior em um dos testes realizados.

O melhor desempenho por parte da solução automatizada não é de todo surpreendente, porém o que nos chama a atenção é justamente o contrario a

proximidade dos resultados com os resultados obtidos através da solução manual. Tivemos dois resultados iguais e em um caso a solução manual foi melhor.

Vários fatores podem ser usados para justificar esta situação, pois, os testes foram realizados por um profissional que conhece esta situação, realizando este tipo de trabalho a anos, isso da a ele possibilidade de usar vários métodos para chegar a uma solução melhor.

Após a ponderação de todos os fatos nós nos convencemos da capacidade de geração de soluções plausíveis que coincidam pelo menos com a realidade que atualmente a empresa atravessa e que atendam plenamente as necessidades de melhoria de tempo e ainda contribuindo na otimização da utilização dos veículos.

Durante a realização dos testes se vislumbrou um novo dado comparativo que se fez de muita importância para a análise da situação em questão este dado é a carga total perdida, que revela a quantidade de capacidade que não foi aproveitada nos carregamentos daquele grupo de veículos.

Percebeu uma relação entre a quantidade mínima de veículos necessária para cada grupo de pedidos e a quantidade máxima de desperdício aceitável para que o mínimo na relação veículos x pedidos fosse atingido. Quanto mais próximo o desperdício ficar de 0 “zero”, mais próximo da situação ótima o resultado será.

Para análise tem-se o seguinte caso hipotético: um valor total para os pedidos de 35 toneladas, se tivermos apenas veículos de 4, 8, 10 e 16 toneladas, qualquer combinação entre estes veículos terá como resultado de carga máxima um número par, logo teremos um desperdício de pelo menos 1 tonelada neste caso.

Porém apenas dizer que zerando o desperdício chega-se ao ótimo é exagerado, o correto é dizer que existe uma relação muito próxima entre estes dois fatores, mas, esta relação não é definitiva. A otimização para este caso depende muito mais de uma distribuição exata dos pedidos entre os veículos e em menor medida do aproveitamento da capacidade do veículo.

Imaginemos agora a seguinte situação temos 16 toneladas em pedidos a serem carregadas em 5 veículos com as seguintes capacidades 4, 4, 8, 10, 16. Para este caso pode-se propor uma situação onde a resposta seria 4, 4, 8, que teria como taxa de desperdício 0, e um cenário onde temos a resposta 16 também com desperdício 0.

Isso deixa bem claro que uma melhor organização é o objetivo final para a proposta de solução do problema.

Já em relação aos comparativos feitos viu-se que a solução proposta se comportou bem obtendo resultados que sempre ficaram na média entre os melhores e piores resultados em relação às outras situações propostas.

5. CONCLUSÃO E PROPOSTA DE TRABALHOS FUTUROS

Este trabalho tinha como proposta inicial a construção e análise de uma solução para um problema apresentado pela empresa de representações “Dinâmica Rural”, problema este que se apresentou como um problema de otimização onde se tinha o objetivo de minimizar a quantidades de veículos utilizados no transporte das mercadorias as quais a mesma representa junto aos clientes finais.

Visto que naturalmente quando se apresenta uma proposta de automatização de um processo os ganhos de produtividade são consideráveis as conclusões quanto a viabilidade da solução ficam no campo da obtenção da solução ótima, ou algo que pelo menos, resulte em um avanço nesta direção.

As pesquisas quanto ao tema indicaram que o melhor caminho a seguir era o de uma solução baseada em Programação Dinâmica, que não se apresentou como solução definitiva, sendo por tanto necessário a utilização de outras técnicas como complemento na obtenção da solução e relaxamento da mesma afim de se aproximar da solução ótima. O que trouxe bons frutos, pois, a solução se demonstrou muito eficiente.

A eficiência da solução foi comprovada em uma bateria de testes que demonstrou que a solução apresentada se aproxima em diversas situações do resultado ótimo, em muitos casos até mesmo alcançando este objetivo. E quando não foi possível atingir o melhor resultado, a solução proposta obteve melhores resultados que outras soluções.

Foram usados como parâmetros de comparação nos testes o tempo computacional e a qualidade da resposta obtida, e concluiu-se que os algoritmos utilizados para comparação possuíam um excelente desempenho em um dos parâmetros de comparação, deixando sempre algo desejar em relação ao outro aspecto da comparação. Já a proposta apresentada neste trabalho oscilou sempre em pontos medianos em relação ao tempo computacional, sendo melhor que o algoritmo de pior desempenho e pior que o algoritmo de melhor desempenho. No tangente a qualidade da resposta a solução apresentada obteve sempre um resultado muito bom, chegando inclusive a atingir o resultado ótimo por várias vezes.

Visto por uma ótica acadêmica é importante o estudo de mais uma variação do Problema da Mochila, um clássico da Pesquisa operacional, propondo uma abordagem que une várias técnicas e chegando a um resultado que apesar de não garantir o resultado ótimo, é claramente um resultado muito bom.

No ponto de vista da necessidade da empresa fica mais claro ainda a importância do estudo e desenvolvimento da solução visto que já foram comprovados os avanços que uma solução automática traz em termos de produtividade para qualquer empresa, a também que se considerar que os resultados obtidos superaram os resultados que são atualmente obtidos, o que marca ainda mais sua posição destaque em relação ao método atualmente utilizado.

Por fim resta salientar que este é um tema ainda em estudo, e que esta não é uma solução definitiva, tanto em relação ao estudo do tema abordado, quanto no desenvolvimento de uma solução tecnológica para os problemas estudados na empresa. Apenas foram dados alguns passos neste sentido, faltando ainda um logo caminho para se percorrer.

Para trabalhos futuros verificaram-se algumas necessidades no projeto de automação dos processos da empresa estudada, pois se tratou apenas da distribuição dos pedidos entre os veículos disponíveis, havendo ainda assuntos que necessitam de atenção.

- Propõe-se um estudo sobre o roteamento das entregas dos pedidos,
- A criação de uma solução que aborde um paralelo entre a solução do roteamento com a solução para o carregamento, e posteriormente, servindo até mesmo como padrão na ora da distribuição da carga.
- Há também que se integrar a solução a um software que efetivamente faça o gerenciamento do processo.

Na realidade, muito ainda falta para ser feito visto que a empresa está em um processo de automação de seus processos, visto ainda que o tema abordado neste trabalho seja um tema que, apesar de muito estudado, possui um amplo espectro de variações que merecem uma atenção individualizada.

7. REFERENCIAS BIBLIOGRÁFICAS:

Arenales, M.; Morabito, R. & Yanasse, H. (1999). **Cutting and packing problems.** ***Pesquisa Operacional*****, 19**(2), 107-299

Armbruster, M. (2002). **A solution procedure for a pattern sequencing problem as part of a one-dimensional cutting stock problem in the steel industry**. *European Journal of Operational Research,* **141**, 328-340.

Arroyo, J.E.C. (2002). **Heurísticas e metaheurísticas para otimização combinatória multiobjetivo.** 256f. Tese (Doutorado em Engenharia Elétrica) – Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas.

Ashikaga, F.M. (2001). **Um método frugal para o problema de minimização de pilhas abertas. 91f. Dissertação** (Mestrado em Ciências no Curso de Engenharia Eletrônica e Computação na Área de Informática) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

Becceneri, J.C. (1999). **O problema de seqüenciamento de padrões para a minimização do número máximo de pilhas abertas em ambientes de cortes industriais.** 145f. Tese (Doutorado em Ciências no Curso de Engenharia Eletrônica e Computação na Área de Informática) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

Belluzzo, L. & Morabito, R. (2005). **Otimização nos padrões de corte de chapas de fibra de madeira reconstituída: um estudo de caso.** ***Pesquisa Operacional,*** **25**(3), 391-415.

CARMO, Gustavo S. B. do, GRINGS, Alexandre, NETO, Henrique de Castro, "**A resolução do "problema da mochila" por três diferentes abordagens**" Dissertação de Mestrado,UFU, 2007.

CORMEN, Thomas H., LEISERSON, Charles E., RIVEST, Ronald L., STEIN, Cliford, "**Algoritmos, Teoria e Prática**" Luis F., Editora Campus, 2003.

Cohon, J.L. (1978). ***Multiobjective programming & planning***. Academic Press, New York.

Dowsland, K. & Dowsland, W. (1992). **Packing problems**. *European Journal of Operational Research*, **56**, 2-14.

Dyckhoff, H. & Finke, U. (1992). ***Cutting and packing in production and distribution:*** Typology and bibliography. Springler-Verlag, Heidelberg.

Dyckhoff, H.; Scheithauer, G. & Terno, J. (1997). ***Cutting and packing*****. In:** ***Annoted bibliographies in combinatorial optimization*** [edited by M. Amico; F. Maffioli and S. Martello], John Wiley & Sons, New York, 393-414.

Dyckhoff, H. & Waescher, G. (Eds.) (1990). **Cutting and packing.** *European Journal of Operational Research*, **44**(2).

Dyson, R.G. & Gregory, A.S. (1974). **The cutting stock problem in the flat glass industry**. *Operational Research Quarterly*, **25**, 41-53.

Faggioli, E. & Bentivoglio, C.A. (1998). **Heuristic and exact methods for the cutting sequencing problem**. *European Journal of Operational Research*, **110**(3), 564-575.

Fink, A. & Vob, S. (1999). **Applications of modern heuristic search methods to pattern sequencing problems**. *Computer and Operations Research*, **26**(1), 17-34.

Foerster, H.E. & Waescher, G. (1998**). Simulated annealing for order spread minimization in sequencing cutting patterns**. *European Journal of Operational Research*, **110**, 272-281.

Gilmore, P. & Gomory, R. (1961). **A linear programming approach to the cutting stock problem.** *Operations Research*, **9**, 849-859.

Gilmore, P. & Gomory, R. (1963). **A linear programming approach to the cutting-stock problem II. *Operations Research,* 11**, 863-888.

GOODRICH, Michael T, TAMASSIA, Roberto, "**Projeto de Algoritmos – Fundamentos, analises e exemplos da intenet**" Editora BookMan, 2002.

Gramani, M.C.N. (2001). **Otimização do processo de cortagem acoplado ao planejamento da produção.** 118f. Tese (Doutorado em Engenharia Elétrica), Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas.

Haessler, R. (1980**). A note on computational modifications to the Gilmore-Gomory cutting stock algorithm**. *Operations Research*, **28**, 1001-1005.

Hendry, L.C.; Fok, K.K. & Shek, K.W. (1996). A cutting stock and scheduling problem in the copper industry. *Journal of the Operational Research Society*, **47**, 38-47.

Hinxman, A. (1980). The **trim-loss and assortment problems: a survey**. *European Journal of Operational Research*, **5**, 8-18.

Hifi, M. (2002). **Special issue: Cutting and packing problems**. *Studia Informatica Universalis*, **2**(1), 1-161.

Linhares, A. & Yanasse, H.H. (2002). **Connections between cutting-pattern sequencing, VLSI Design, and flexible machines**. *Computers & Operations Research*, **29**(12), 1759-1772.

Lins, S. (1989). Traversing **trees and scheduling tasks for duplex** corrugator machines. *Pesquisa Operacional*, **9**, 40-54.

Lodi, A.; Martello, S. & Monaci, M. (2002). **Two-dimensional packing problems: a survey.** *European Journal of Operational Research*, **141**, 241-252.

Madsen, O. (1979). **Glass cutting in a small firm.** *Mathematical Programming*, **17**, 85-90.

Madsen, O. (1988**). An application of travelling-salesman routines to solve pattern-allocation problems in the glass industry.** *Journal of the Operational Research Society*, **39**, 249-256.

MARQUES, Fabiano Prado, "**O Problema da Mochila Compartimentada**", Dissertação de Mestrado,USP, 2000.

Martello, S. & Toth, P. (1990). ***Knapsack problems:* algorithms and computer implementations.** John Wiley & Sons, Chichester.

Morabito, R. & Arenales, M. (1992). **Um exame dos problemas de corte e empacotamento. *Pesquisa Operacional,* 12**(1), 1-20.

Morabito, R. & Arenales, M. (2000**), Optimizing the cutting of stock plates in a furniture company.** *International Journal of Production Research*, **38**(12), 2725-2742.

Morabito, R. & Garcia, V. (1998a). **Uma abordagem para o problema de cortes de chapas de fibra de madeira reconstituída.** *Pesquisa Operacional*, **18**(1), 37-57.

Morabito, R. & Garcia, V. (1998b). **The cutting stock problem in a hardboard industry: A case study.** *Computers & Operations Research*, **25**(6), 469-485.

Oliveira, J.F. & Waescher, G. (2005). **Special Issue on Cutting and Packing.** *European Journal of Operational Research* (a aparecer).

Pileggi, G.C.F. (2002**). Abordagens para otimização integrada dos problemas de geração e seqüenciamento de padrões de corte**. 154f. Tese (Doutorado em Ciências da Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos.

Pinto, M.J. (1999**). O problema de corte de estoque inteiro**. 98f. Dissertação (Mestrado em Ciências da Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos.

Pinto, M.J. (2004). **Algumas contribuições à resolução do problema de corte integrado ao problema de seqüenciamento dos padrões.** Tese (Doutorado em Computação e Matemática Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos.

Pisinger, D. (2000). **A minimal algorithm for the problem. bounded knapsack** *INFORMS Journal on Computing*, **12**, 75-84.

Poldi, K.C. & Arenales, M. (2005**). Dealing with small demand in the integer cutting stock problem with limited different stock lenghts**. Notas do ICMC-USP, n. 85, São Carlos.

Poltroniere, S.C.; Arenales, M.; Toledo, F.M.G. & Poldi, K.C. (2005**). Coupling cutting stock and lot sizing problems in the paper industry.** Notas do ICMC-USP, n. 83, São Carlos.

Riehme, J.; Scheithauer, G. & Terno, J. (1996). **The solution of two-stage guilhotine cutting stock problems having extremely varying order demands**. *European Journal of Operational Research*, **91**, 543-552.

Silveira, R. & Morabito, R. (2002). **Um método heurístico baseado em programação dinâmica para o problema de corte bidimensional guilhotinado restrito**. *Gestão & Produção*, **9**(1), 78-92.

Soma, N.Y. & Toth, P. (2002). **An exact algorithm for the subset sum problem.** *European Journal of Operational Research*, **136**, 57-66.

*SILVEIRA, Rejane Joas, MORABITO Reinaldo***, "Um Método Heurístico baseado em Programação Dinâmica para o problema de corte bidimensional guilhotinado restrito",** Departamento de Engenharia de Produção, Universidade Federal de São Carlos, 2002

Steuer, R.E. (1986). ***Multiple criteria optimization*: theory, computation & application.** John Wiley, New York.

Sweeney, P. & Paternoster, E. (1992). **Cutting and packing problems: a categorized, application-oriented research bibliography**. *Journal of the Operational Research Society*, **43**, 691-706.

Vance, P.; Barnhart, C.; Johnson, E. & Nemhauser, G. (1994**). Solving binary cutting stock problems by column generation and branch-and-bound.** *Computational optimization and applications*, **3**, 111-130.

Yanasse, H.H. (1996a). **Minimization of open orders – polynomial algorithms for some special cases.** *Pesquisa Operacional*, **16**, 1-26.

Yanasse, H.H. (1996b). **A transformation for solving a pattern sequencing problem in the wood cut industry**. INPE/LAC Technical Report 6, 15p, São José dos Campos.

Yanasse, H. (1997). **On a pattern sequencing problem to minimize the maximum number of open stacks.** *European Journal of Operational Research*, **100**(3), 454-463.

Yuen, B. (1991**). Heuristics for sequencing cutting patterns**. *European Journal of Operational Research*, **55**, 183-190.

Yuen, B. (1995**). Improved heuristics for sequencing cutting patterns.** *European Journal of Operational Research*, **87**, 57-64.

Waescher, G. & Gau, T. (1996**). Heuristics for the integer one-dimensional cutting stock problem: a computational study**. *Operations Research Specktrum*, **18**, 131-144.

Wang, P.Y. & Waescher, G. (2002). **Cutting and packing.** *European Journal of Operational Research*, **41**(2), 239-469.

WEBER, Hans Hermann, "**Pesquisa Operacional**", Editora Universitária da UFPb, 1980.

ZIVIANE, Nivio, "**Projeto e Análise de Algoritmos: com implementações em pascal e C**", Edição 2, Editora Thomson Pioneira, 2009.