Universidade de Aveiro DETI - Departamento de Electrónica Telecomunicações e Informática

2009

César Miguel da Costa e Veiga Esteves

SISTEMA DE INFORMAÇÃO PARA APOIO AO CORTE DE BOBINES DE CHAPA

César Miguel da Costa e Veiga Esteves

SISTEMA DE INFORMAÇÃO PARA APOIO AO CORTE DE BOBINES DE CHAPA

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e de Telecomunicações, realizada sob a orientação científica do Dr. Joaquim Arnaldo Martins, Professor Catedrático do Departamento de Electrónica Telecomunicações e Informática da Universidade de Aveiro

Agradecimentos

Dedico este trabalho aos meus amigos
e familiares que me acompanharam
ao longo deste percurso.

Ao Professor Doutor Joaquim Arnaldo Martins
pelo seu acompanhamento neste trabalho

À Universidade e cidade de Aveiro por tudo o
que transmitiu e ofereceu ao longo destes anos

O júri

Presidente

Professor Doutor José Luís Guimarães Oliveira
Universidade de Aveiro

Doutor Fernando Joaquim Lopes Moreira
Departamento de Inovação Ciência e Tecnologia da Universidade Portucalense

Professor Doutor Joaquim Arnaldo Carvalho Martins
Universidade de Aveiro

Objectivo

O presente trabalho pretende divulgar diversas técnicas de resolução de problemas de optimização, assim como classificar e encontrar uma resolução para o presente problema apresentado pelo grupo Plafesa.

Palavras – chave Cutting Stock Problem, aço, bobina, optimização

Resumo

O propósito desta pesquisa é o de estudar modelos de optimização relacionados com a indústria metalúrgica envolvendo o *"cutting stock problem"* (CSP) com o objectivo de melhorar o método usado no grupo Plafesa. O objectivo é melhorar o tempo de processamento e diminuir as perdas de material.

Há inúmeros métodos que pretendem resolver o CSP genericamente, contudo ao longo do trabalho abordaremos processos mais específicos para resolver o problema em causa. A mais pequena melhoria no processo provoca grandes resultados nesta indústria.

Através de um modelo algorítmico criado, foi possível atingir o objectivo proposto, conseguindo resultados optimizados num curto espaço de tempo.

Keywords Cutting Stock Problem, steel, coils, optimization

Resume

The purpose of this research is to study optimization models of *"cutting stock problem"* (CSP) in order to improve the method used on Plafesa. The objective is to improve the processing times and to reduce the trim loss.

There are a numerous methods for solving the generic cutting stock problem, but along this article we study specific algorithms to solve our problem. Even a small improvement will generate great results in this industry.

Through an algorithmic method it was possible to achieve the objective proposed with "optimal" results and a good temporal performance.

Índice Geral

Índice de figuras

Índice de tabelas

Índice de gráficos

Índice de equações

Acrónimos

ACO	Ant Colony Optimization
BB	Branch & Bound
CSP	Cutting stock problem
DCG	Delayed column generation
ECP	Extended Cutting Plane
GRASP	Greedy Randomized Adaptive Search Procedure
LP	Linear program
MILP	Mixed Integer Nonlinear Programming
SHP	Sequencial Heuristic Procedure
TS	Tabu Search
TSP	Traveling Salesman Problem

Nomenclatura

b	*bobinas usadas*
j	índice de cada corte padrão
i	índice de cada pedido
A*ij*	número de peças cortadas de um determinado pedido
xj	número de vezes que cada padrão *j* é usado
L	conjunto das larguras dos pedidos
P	conjunto dos pesos dos pedidos
LB	conjunto das larguras das bobinas
LP	conjunto dos pesos das bobinas

Capítulo 1

Introdução

1. Problema exposto.

Os sistemas informáticos de gestão para apoio à decisão são desenvolvidos e enquadram-se numa determinada área de negócio com o objectivo de serem uma ajuda ao elemento humano. Na grande maioria dos casos os sistemas de informação e módulos de produção visam a integração de produtos, ou seja, a junção de vários módulos para que no fim resulte apenas um produto. O que se pretende no âmbito deste trabalho é precisamente o contrário, partir de um único produto e deste obter subprodutos diferenciados consoante a especificidade de cada sector, usando para isso um sistema de informação para apoio à decisão que tenta obter a melhor solução de forma a optimizar o processo produtivo.

1.1. Preâmbulo

O problema geral de optimização aplicado ao corte de material toma o nome de *"Cutting Stock Problem"* (CSP). O CSP pode dividir-se em dois outros problemas que surgem quando se tenta cortar objectos a partir de uma peça maior. Temos então o que se chama de "*assortment problem"*, ou seja, problema de escolher as dimensões correctas nos objectos a serem cortados e o "*trim loss problem"* que tenta resolver a questão de como cortar as pequenas peças de modo a que se desperdice o menos possível. A nível industrial, especialmente na indústria do papel e do aço, as peças mais

pequenas são conhecidas como "*pedidos*" e os objectos originais têm a designação de "*stock*". Ao juntarmos estes dois problemas de produção atingimos o problema anteriormente referido, "*cutting stock problem*".

1.2. Âmbito

A pesquisa de soluções para o CSP começou há praticamente meio século e desde então inúmeros métodos foram desenvolvidos uns melhor que outros, mas sempre chegando à conclusão que não existe uma solução óptima mas sim várias soluções.

É objectivo deste trabalho encontrar um módulo de apoio à decisão associado à indústria do aço de forma a obter a melhor decisão de produção em função de variáveis de informação fornecidas por uma base de dados ou por um módulo externo.

O processo de produção baseia-se no corte de bobines de chapa que serão divididas consoante os requisitos de cada cliente. Pretende-se chegar a um resultado o mais óptimo possível, de forma automatizada ganhando assim tempo de processamento e diminuindo os gastos.

O objectivo final do trabalho é integrar este módulo no módulo de gestão do grupo empresarial PLAFESA.

Capítulo 2

Cutting Stock Problem (CSP)

Depois de conhecido o objectivo do trabalho é importante enquadrar o problema e situá-lo no contexto científico actual. Nas seguintes páginas será apresentado mais em detalhe o CSP e seus problemas derivados, bem como algumas soluções existentes e seus respectivos algoritmos. Sempre que possível tentar-se-á adaptar cada solução a uma situação real para que se torne mais claro e perceptível para o leitor.

2. Cutting Stock Problem (CSP)

O problema do corte de material, conhecido como *"Cutting Stock Problem"*, é um problema clássico de optimização aplicado nas mais diversas indústrias, desde a indústria do papel, à cinematográfica passando pela indústria metalúrgica. Onde quer que seja que se apliquem técnicas de optimização estas terão sempre como objectivo minimizar os excessos aproveitando assim o máximo possível de material e aumentando os lucros finais.

No dia-a-dia somos confrontados com inúmeros problemas de optimização que vamos resolvendo sem grandes cálculos mas que por vezes levam bastante tempo até atingirem níveis satisfatórios. Como exemplo quotidiano de optimização de espaço temos um caso tão simples como ir viajar. Quando pretendemos arrumar a mala de viagem com tempo e paciência para experimentar diversas posições e sequências, levamos tudo bem arrumado chegando por vezes a sobrar espaço. Esta poderá não ser a melhor distribuição da roupa na mala mas se o resultado obtido for satisfatório teremos o problema solucionado. O contrário não se verifica pois, quando regressamos de férias

com as mesmas roupas e a mesma mala, constatamos que o espaço destinado à carga é insuficiente. Isto ocorre apenas porque não dispusemos as coisas da mesma maneira e certamente não fizemos as experiências suficientes para obter o mesmo resultado obtido anteriormente, diminuindo assim a probabilidade de acertar no modelo correcto. Seguramente nestas alturas seria bom recordarmo-nos da disposição anterior das peças de roupa para que as colocássemos nos seus sítios à primeira tentativa, coisa que raramente acontece. Ainda assim, esta recordação visual da anterior disposição apenas nos iria servir para a situação pontual em que viajássemos sempre com as mesmas peças e objectos o que de igual modo não acontece. Todos estes problemas desapareceriam se possuíssemos alguma ferramenta que nos permitisse medir o espaço disponível, que contabilizasse cada uma das peças de roupa e nos orientasse para uma reposição perfeita destas no saco de viagem e depois dos vários sacos de viagem no porta bagagens do carro. Se viajássemos todos os dias com malas diferentes, roupas diferentes e em carros diferentes, seguramente que nos tentaríamos mecanizar para que o processo de arrumação fosse cada vez mais rápido e instintivo. A nível industrial é este o problema com que nos deparamos diariamente, embora a consequência da falta de optimização sejam custos financeiros. Assim uma empresa de corte de papel ou corte de metal que tem diariamente de fazer a gestão de diversos pedidos dos seus clientes tenta rentabilizar ao máximo a matéria-prima de modo a não desperdiçar, ficando assim mais competitiva no seu mercado. Esta optimização de espaço é como um “puzzle” em que tentamos colocar o maior número de peças dentro de uma determinada área. Hoje em dia apesar de o ‘Homem’ ainda ter muita influência na construção deste “puzzle”, o resultado que obtém muito dificilmente é o “mais óptimo” recorrendo por isso a métodos computacionais e algoritmos lineares para esse processo.

A rentabilidade de um bom processo de optimização é enorme, sendo que uma diferença mínima de 1% para uma produção não optimizada pode fazer a diferença de milhões de euros ao final de apenas um ano.

2.1. Classificação de um problema

Uma das primeiras formulações conhecidas deste tipo de problemas foi feita por um economista russo, Kantorovich em 1939, embora só fosse publicado muitos anos mais

tarde. De seguida apresentamos uma maneira de classificar este tipo de problemas usando um esquema de classificação desenvolvido por Dyckhoff [1].

A classificação de problemas de CSP pode ser feita segundo quatro características, sendo elas:

- Dimensionalidade
  - N – número de dimensões
- Tipo de disposição
  - B – Todo o objecto e uma selecção dos pedidos
  - V – Uma selecção do objecto e todos os pedidos
- Disposição das peças de stock
  - O – Apenas um objecto
  - I – Objectos parecidos
  - D – Objectos diferentes
- Disposição dos pedidos
  - F – Poucos pedidos de diferentes dimensões
  - M – Muitos pedidos de diferentes dimensões
  - R – Muitos pedidos relativamente parecidos
  - C – Muitos pedidos parecidos

De todas estas características a mais importante é a relativa à dimensão do problema. Segundo Dyckhoff [1] existem quatro níveis de dimensões. Problemas de dimensão um, dois, três e multidimensionais. No entanto existe um número considerável de problemas que se posicionam entre uma dimensão e duas dimensões chamando-se assim de problemas de dimensão 1,5 (Hinxman, 1979; Haessler [2]);

Os problemas de apenas uma dimensão – 1D, incluem corte de papel, cortes de tubos, etc. Estes são problemas clássicos de optimização em que vários objectos de diversos comprimentos devem ser cortados de material em stock que tem dimensões fixas.

Podemos ainda tratar de problemas a duas dimensões – 2D, sendo estes problemas relacionados com mobiliário, roupas e cortes de vidros. Nestas situações os pedidos podem ser "colocados" no stock em duas dimensões livres.

Por fim existem problemas menos comuns mas igualmente interessantes que são problemas de três dimensões – 3D, isto é, problemas de empacotamento sejam eles colocar objectos na mala de um carro ou preencher os contentores de um barco.

Outro factor importante é o tipo de disposição. Existem duas categorias a considerar. Na primeira todo o material é usado mas os pedidos não vão ser todos atendidos. Na segunda caso, todos os pedidos devem ser atendidos mas nem todo o material vai ser usado.

A terceira característica (disposição das peças em stock) divide-se em três categorias. A primeira acontece quando apenas existe uma peça em stock para cortar, a segunda quando existem várias peças de dimensão idêntica e o terceiro caso existe para quando em stock existem diversos objectos de dimensões distintas.

A quarta característica (disposição dos pedidos) é semelhante à anterior mas aplicada para a gestão dos pedidos. Dividindo-se em quatro categorias pois temos casos distintos consoante o número e dimensão dos pedidos.

Este trabalho vai-se focar na resolução de problemas de apenas uma dimensão – 1D, pois é sobre o corte de chapas de aço que incide. No entanto não deixamos de fazer referência aos demais aspectos relevantes dos restantes tipos de problemas.

De seguida ilustraremos dois exemplos gráficos de problemas unidimensionais.

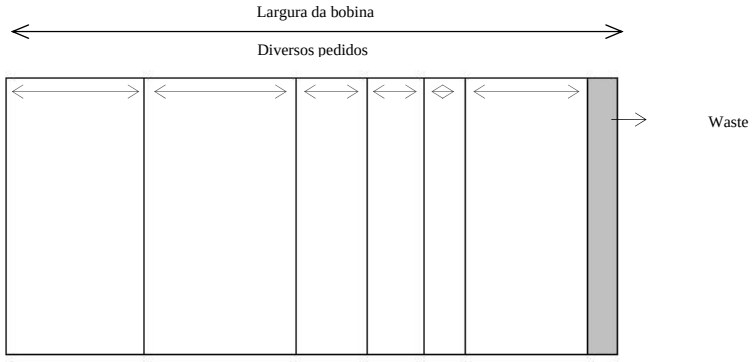

Figura 1 – Exemplo de uma folha de aço pronta a ser cortada

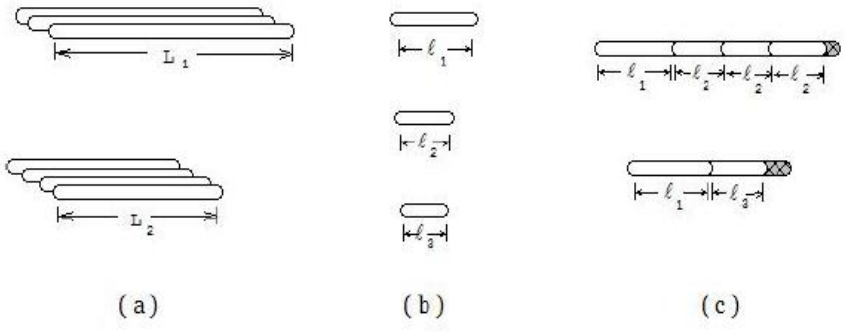

Figura 2 – Processo de optimização: (a) objectos em stock para corte; (b) Pedidos; (c) disposição dos pedidos na peça original

2.2. Soluções para o CSP

Existem três métodos tipo para resolver CSP's.

- Métodos heurísticos – Não obtêm os melhores resultados mas conseguem rapidamente atingir uma solução aceitável. Este método tem a desvantagem de ser demasiado dependente do problema em questão, ou seja, usa informação acerca do problema particular para o qual está a ser construido.

- Métodos meta heurísticos – É um método heurístico para resolver de forma genérica problemas de optimização. Aplicados geralmente a problemas sem um algoritmo eficiente específico satisfatório. Alguns dos métodos mais conhecidos são o GRASP, o Tabu Search e o Ant Colony Optimization (ACP)

- Métodos algorítmicos – Garantem uma solução o mais óptima possível no entanto a sua complexidade pode traduzir-se em longo tempo de processamento.

2.2.1. Métodos Heurísticos

Dentro da heurística existem três variantes. Podemos começar por apresentar a dificuldade existente em resolver problemas de programação linear (LP – Linear programming) no CSP. A linearização é um passo complexo e que recorre a algoritmos matemáticos sendo na verdade uma junção dos métodos algorítmico e heurístico. Na verdade Hinxman [2] classifica este método (LP) como um método algorítmico apesar de este não apresentar uma solução óptima para o CSP.

2.2.1.1. Programação linear (LP)

Para facilitar a compreensão vamos usar como exemplo demonstrativo de um problema de programação linear (LP) o caso de um agricultor. Suponhamos que um determinado agricultor tem uma vinha com A km^2 em que pode cultivar uva branca ou preta ou ainda uma mistura das duas qualidades de uva. A quantidade de fertilizante F e pesticida P que o agricultor tem é limitada sendo que cada um deles terá de ser usado em quantidades diferentes para a uva branca ou para a uva preta, ($F1,P1$) para uva branca, e, ($F2,P2$) para uva preta. O preço de venda também é distinto, sendo o preço da uva branca igual a $E1$ e da preta igual a $E2$. A área plantada por uva branca vai ser $x1$ e por uva preta de $x2$, então o número ideal de km^2 de uva branca e preta pode ser expressado através de um problema de programação linear.

Pretende-se maximizar $S1x_1 + S2x_2$. A esta função chamamos de função objectivo.

Esta maximização está sujeita a :

$$x_1+x_2 \leq A$$
$$F1x_1+F2x_2 \leq F$$
$$P1x_1+P2x_2 \leq P$$
$$x_1 \geq 0,\ x_2 \geq 0$$

De seguida é apresentado um exemplo de um problema de uma dimensão, um caso de aplicação na indústria do aço, precisamente o objectivo de estudo desta dissertação.

Consideremos que possuímos bobinas com comprimentos diferentes e larguras semelhantes. Vamos agora tentar encontrar quais os cortes padrão que melhor

satisfazem os pedidos para um valor mínimo de desperdício, ou sucata como é designado no meio industrial. (trim loss). O problema pode ser formulado da seguinte maneira (equações (1) e (2)) (Eisemann [3], Haessler,1975 [11], Haessler et al., 1991 [12]):)

$$\min \sum_{j=1}^{J} \left(B - \sum_{i=1}^{I} b_i n_{i,j} \right) m_j, \qquad \text{(equação 1)}$$

$$N_{i,min} \leq \sum_{j=1}^{J} m_j n_{i,j} \leq N_{i,max}, \qquad m_j \in \mathbf{Z}^+ \qquad \text{(equação 2)}$$

Sendo N_i o número de bandas com comprimento b_i a serem cortadas de uma bobina de comprimento *B* e considerando que todos os pedidos têm a mesma largura.

Tal como no exemplo do papel vamos atribuir a variável *j* para o corte padrão em que m_j representa um vector coluna e $n_{i,j}$ o número de sub-bobinas de comprimento b_i produzidas por esse corte padrão. Desde já verificamos que o problema se torna demasiado complexo devido ao número elevado de cortes padrão possíveis, aumentando o processamento. Uma maneira de ultrapassar esta dificuldade é resolver o problema sem enumerar todas as situações viáveis. A partir daqui surge um novo problema ao qual é dado o nome de *"knapsack problem"*.

O *"Knapsack problem"* é mais um dos problemas clássicos de optimização. O seu objectivo é ocupar o espaço de uma mochila com a maior quantidade de objectos possíveis, podendo ou não tratar-se de objectos de dimensão variável. Este problema pode ser resolvido com um algoritmo conhecido como *branch and bound algorithm* (BB). Este algoritmo divide o conjunto de soluções em subconjuntos. O que acontece é que sendo cada subconjunto da mesma natureza que o conjunto mãe, estes irão também ser divididos em subconjuntos. Através do uso de limites superiores e inferiores, alguns subconjuntos vão sendo eliminados. O procedimento BB requer dois passos, o primeiro é um processo de *splitting*, em que dado um conjunto *S,* este se divide em subconjuntos $S_{1,}$ $S_{2...}$. Supondo *f(x)* como sendo uma função dos custos de material, é nosso objectivo minimizar esta função. O mínimo de *f(x)* sobre *S* é ***min{v1,v2,…}*** onde v_i é o mínimo de *f(x)* sobre S_i. A este passo chamamos de branching e leva à construção de uma árvore de soluções. Ao segundo passo, descoberta dos limites é dado o nome de bounding. Há

ainda um terceiro passo que descreve o conceito chave do processo BB. Se o limite inferior para um nó *A* for maior que o limite superior de um nó *B* então *A* pode ser descartado da busca. Este passo denomina-se por *pruning*. O algoritmo BB teve a sua origem em A. H. Land and A. G. Doig em 1960[5], em baixo (Figura (3)) podemos observar um exemplo da divisão do conjunto *S* em subconjuntos, cada um com os seus limites inferior e superiores.

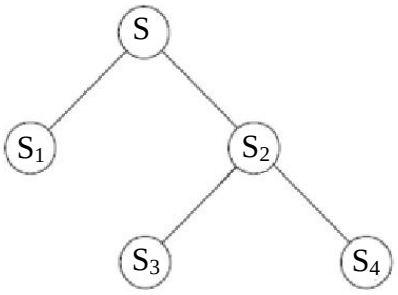

Figura 3 – Árvore de Soluções (search tree) em Branch and Bound.

Apesar de eficiente, muitos estudiosos crêem existir alguns inconvenientes no uso da programação linear. É um método pouco ambicioso, pois o seu único objectivo é a redução de desperdício (trim loss), e o knapsack problem apenas se resolve se algumas restrições forem colocadas e respeitadas.

Um outro método para solucionar o CPS é usar o SHP (sequencial heuristic procedure).

2.2.1.2. SHP – Sequencial Heuristic Procedure

Com o SHP os cortes padrão são gerados à medida que os pedidos vão sendo despachados.

São seleccionados os cortes padrão com pouco *trim loss* ou seja, com grande taxa de usabilidade.

Este método usa lógica combinatória sobre conjuntos de soluções.

Apresenta um conjunto solução *S* e encontra um resultado *S'* pertencente a *S* em que $z(S') < z$, isto é, em que o custo seja menor. Caso se verifique volta a comparar o custo

deste resultado com o custo de outro e assim sucessivamente até que encontre o resultado óptimo.

2.2.1.3. Soluções híbridas

Um outro método de obter soluções para o CSP é optar por uma junção dos dois métodos apresentados anteriormente. Este método tem a vantagem de unir a versatilidade e capacidade de adaptação do SHP com o procedimento de LP para minimização de desperdício e tudo isto num único procedimento

Este procedimento pode realizar-se de modo SHP ou LP puros, ou ainda de maneiras híbridas, isto é, iniciar o processo por usar o SHP e depois com um conjunto de soluções óptimas inicializar o processo LP, ou por outro lado, inicializar usando o processo linear e depois então aplicar o SHP.

2.2.2. Métodos meta-heurísticos

Um novo modelo de métodos heurísticos chamados de meta-heurísticos começou a emergir em finais do século passado. Novos algoritmos surgiram rapidamente. Muitos deles são aplicados ao CSP entre os quais destacaremos o *Tabu Search* (TS), *Greedy Randomized adaptive Search procedure* (GRASP) e *Ant Colony Optimization* (ACO).

A breve descrição de cada um destes métodos segue abaixo.

2.2.2.1. Tabu Search (TS)

Este método de optimização matemática foi atribuído a Fred Glover

O método de optimização TS parte de uma solução possível e avança para outras soluções cada vez melhores na sua vizinhança.

Um exemplo bastante usado para ilustrar o funcionamento do TS é o *Traveling Salesman Problem* (TSP). Este problema envolve viajar entre cidades e procurar que a distância total *p*ercorrida visitando todas as cidades seja minimizada. Se tivermos três cidades, *A*, *B* e *C*, e se supusermos que as cidades *B* e *C* estão perto uma da outra, torna-

se óbvio que a distância será mínima se estas forem visitadas uma depois da outra antes ou depois de visitar a cidade *A*. Neste exemplo o TS começa por uma solução inicial que satisfaça a condição de visitar as três cidades, de seguida para buscar novas soluções as ordens são trocadas e assim as distâncias totais são comparadas umas sobre as outras. De modo a prevenir ciclos e *local optima solutions* um resultado óptimo é guardado na *Tabu List* (TL) se for uma solução na vizinhança. Este procedimento é executado até se satisfazer um determinado critério que pode ser de limite de iterações ou limite de tempo decorrido.

2.2.2.2. GRASP (Greedy randomized adaptive search procedure)

GRASP é um algoritmo primeiramente apresentado por Feo e Resende [6] nos anos 80, a partir dos quais este método tem vindo a ser aplicado em várias aplicações quer de pesquisa quer a nível industrial. Basicamente consiste na criação de uma solução inicial e posterior busca local de um melhoramento da qualidade da solução. De seguida é apresentado um procedimento básico do greedy randomized adaptive search procedure (GRASP) para um problema de minimização, sendo x^* a melhor solução encontrada e $f^*=f(x^*)$ [19]

Pseudo código:

```
While ( critério não satisfeito) {
    Solução = criar solução de forma construtiva aleatoriamente ();
    If (Solução = melhor encontrada){
        Gravar Solução;
    }
}
```

Figura 4 – Pseudo-código GRASP

2.2.2.3. Ant Colony Optimization (ACO)

Este método foi introduzido por Marco Dorigo [7] e foi inspirado no comportamento das formigas e na sua forma de encontrar caminhos para a sobrevivência da colónia.

As formigas possuem uma estrutura de comunidade invejável e na sua busca por alimento deixam rastos de odor para que outras formigas possam seguir o mesmo rasto não tendo assim a necessidade de voltar a procurar. No entanto num percurso grande o rasto facilmente se apaga e quando uma outra formiga encontra um rasto alternativo o ciclo repete-se, visto que num percurso mais pequeno o rasto de odor é reforçado de forma mais frequente estes acabam por prevalecer sobre os outros. Este método pretende simular uma colónia de formigas percorrendo todas as possibilidades para assim encontrar uma solução óptima. Na figura (5) podemos observar como tendencialmente os percursos mais longos deixam de ser utilizados.

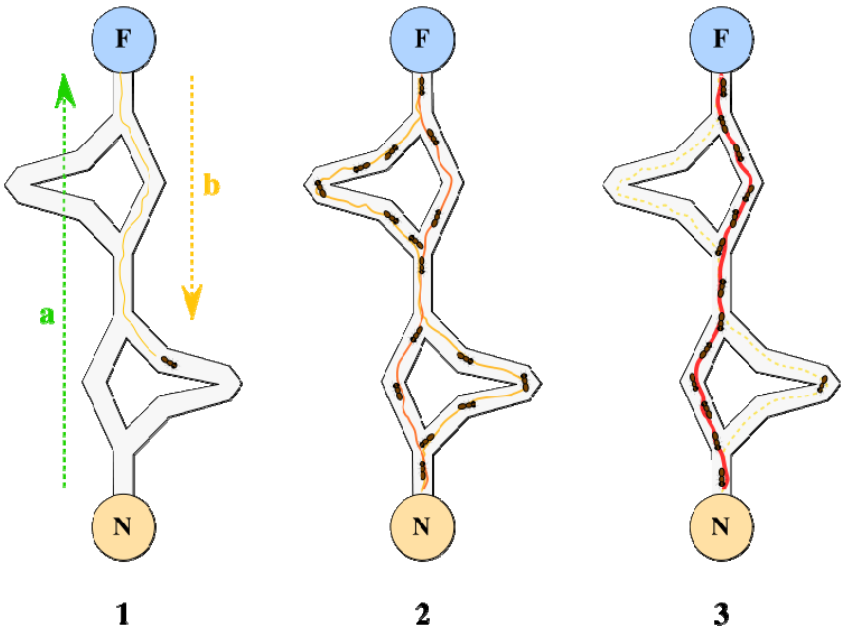

Figura 5 – Ant colony optimization

Pseudo código:

```
procedure ACO_MetaHeuristic
                  while(not_termination)
                        generateSolutions()
                         pheromoneUpdate()
                         daemonActions()
                  end while
                 end procedure
```

Figura 6 – Pseudo-código ACO

Na resolução de problemas de optimização, como já foi possível constatar é difícil obter uma solução óptima, pois é necessário conseguir um equilíbrio entre tempo de processamento, resultado final e custos associados. Os métodos apresentados anteriormente conseguem um tempo de processamento relativamente rápido embora os seus resultados não sejam os mais "óptimos" possíveis. Para conseguirmos uma optimização mais eficaz há que recorrer a métodos algorítmicos para a resolução do CSP. No entanto o custo de um resultado melhor é um aumento significativo no tempo de processamento dos dados.

Dois caminhos diferentes podem ser seguidos em ordem a executar métodos algorítmicos. MINLP - mixed integer nonlinear programming e ECP - extended cutting plane. Ambos baseados na ideia de transformar problemas MILP não convexos em problemas de possível resolução.

Em baixo é apresentada uma tabela resumo na qual constam os métodos apresentados e suas vantagens e inconvenientes.

Método	Vantagens	Desvantagens
SHP	-Rápido. -Permite considerar diversos objectivos	-Em problemas mais complicados não conduz à melhor solução.
Hybrid	-Produz soluções inteiras	-Mais lento que métodos heurísticos puros.
TS	-Consegue melhores resultados do que o GRASP	-Mais lento que o GRASP
GRASP	-Obtém resultados satisfatórios e é mais rápido que o TS	-Resultados menos precisos quando comparado com o TS
ACO	-Resultados satisfatórios	-Um pouco lento

Tabela 1- vantagens e desvantagens das diferentes soluções

Capítulo 3

Caso de estudo: Grupo Plafesa

3. Caso em estudo

O presente problema para a especificidade do grupo Plafesa será abordado de seguida assim como será apresentada uma formulação matemática acerca das suas limitações e objectivos. A identificação matemática do problema é um processo importante para que se possa pensar em aplicação computorizada.

3.1. Apresentação do problema.

O problema em mãos diz respeito a uma adaptação do CSP à realidade existente no grupo Plafesa. O que se pretende é optimizar o processo de corte de bobinas de chapa de aço e assim minimizar o desperdício. A figura (7) mostra o processo de uma linha de corte.

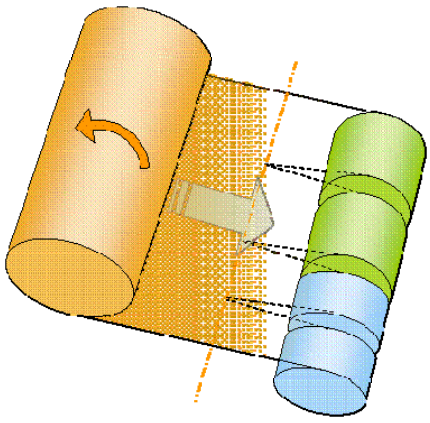

Figura 7 – Ilustração do processo de corte de uma bobina e sua divisão

Este processo inicia-se com o alinhamento e posicionamento dos objectos de corte de acordo com as diversas ordens de corte dos clientes. O que se tenta fazer neste momento é colocar o máximo de ordens ou pedidos numa determinada bobina diminuindo assim o número de bobinas usadas e a quantidade de desperdício/sucata. Depois de iniciar o processo, este não pode ser interrompido, a não ser que algum problema ocorra e a maquinaria pare, pelo que quando iniciado o corte de uma peça este tem de percorrer todo o seu comprimento. As peças resultantes são automaticamente enroladas numa nova bobina, bobina esta que corresponde a um dos pedidos satisfeitos.

As encomendas feitas pelos clientes são especificadas tendo em conta o peso e a largura da banda a ser cortada. Como exemplo podemos ter um pedido de 100 x 5000_{Kg} o que significaria ter uma banda de bobina de 100_{mm} com um peso de 5000_{Kg}. Se ao cortar uma banda de 100_{mm} ao longo de toda a bobina não satisfizermos a segunda condição, 5000_{Kg}, há a necessidade de cortar uma segunda banda de 100_{mm} e assim sucessivamente até que as duas condições estejam satisfeitas. Como se pode já concluir, poderá existir a situação em que o total excede o peso especificado, ou que por exemplo no final de um corte o peso não seja atingido por uma margem muito pequena. Nestes casos pontuais há uma margem de tolerância para o corte, sendo que o peso total que o cliente leva é um valor aproximado do que foi requisitado. Concluindo, para uma determinada medida solicitada o número de bandas cortadas da bobina mãe é aquele necessário para satisfazer o peso demandado.

3.1.1. Requisitos do problema

O corte obedece a determinados requisitos, sendo as bobinas especificadas em termos do deu peso bruto, do seu peso relativo, do seu raio interno, raio externo e espessura.

As bobinas em stock têm dimensões pré-definidas de 1000_{mm} 1250_{mm} e 1500_{mm}, sendo que o peso destas poderá variar.

Não são aceites encomendas com menos de 70_{mm}

A sucata mínima devido a pequenas imperfeições nas extremidades da chapa é de 3_{mm} em cada extremidade, logo 6_{mm} no total.

Partindo do problema colocado e conhecendo um pouco do seu enquadramento podemos tentar formalizar uma das muitas soluções possíveis. Vamos dar a conhecer a solução implementada e sua apresentação matemática. Posteriormente é feita uma análise da solução encontrada o que nos permitirá observar tempos de resposta e estabelecer um "Capacity Planning" de acordo com o crescimento do número de pedidos.

3.2. Apresentação matemática

A apresentação matemática do problema é de extrema importância para de forma abreviada explicarmos qual o objectivo proposto.

Sendo *b* um conjunto de bobinas usadas, e *n* um conjunto de pedidos, temos que x_{ij} é o número de cortes de largura *i* na bobina *j,* e que, p_{ij} é o peso do corte de largura *i* na bobina *j*, sendo *j* a bobina que esta a ser usada.

Sendo,

L – o conjunto das larguras dos pedidos

P – o conjunto dos pesos dos pedidos

LB – o conjunto das larguras das bobinas

LP – o conjunto dos pesos das bobinas

Temos como objectivo minimizar *b*, ou seja o número de bobinas usadas. Desta maneira estamos a maximizar o uso de cada uma das bobinas para um maior aproveitamento de espaço.

De modo a atingir o objectivo há condições a respeitar, assim:

$$\sum_{i=1}^{n} x_{ji}.L_i \leq LBj \qquad \text{(equação 3)}$$

Como é fácil de entender pela equação (3), o somatório da multiplicação de uma determinada largura *Li* pelo número de vezes que essa largura é cortada na bobina *j* não pode ser superior à largura total da bobina *LBj*.

$$\sum_{j=1}^{b} x_{ji} . p_{ij} \geq Pi \qquad \text{(equação 4)}$$

Do mesmo modo temos na equação (4) que o resultado da multiplicação do peso de determinado corte *i*, na bobina *j* pelo número de vezes que esse corte está presente na mesma bobina *j* não pode ser inferior ao peso total do pedido, *Pi,* para assim satisfazer o pedido do cliente.

3.3. Possível solução

Com o problema descrito matematicamente é mais simples entender o seu modo de funcionamento.

Para o caso concreto do grupo Plafesa em que o número de pedidos em espera simultaneamente não ultrapassa os 10 (dez), é possível usar um método de optimização mais simples conceptualmente embora mais exigente a nível de processamento. Foi optado usar um algoritmo que percorresse todas as combinações possíveis e que para cada caso confirmasse a percentagem de ocupação da bobina. Este método parte de uma solução não óptima e percorre as várias combinações possíveis de encaixar os diversos pedidos num rolo de bobina até encontrar uma solução melhor. Apenas termina quando não houver mais hipóteses a testar, e aí apresentará a melhor solução encontrada. Vamos de seguida apresentar um exemplo do uso deste método, e o que fazer quando o volume de pedidos se tornar demasiado grande.

No capitulo anterior foram apresentados alguns métodos existentes para a resolução de CSP's. No ponto seguinte será exposto uma possível resolução do problema de optimização de bobinas, para isso usaremos um exemplo encontrado na bibliografia.

Mais à frente será apresentada a solução proposta para a resolução do problema.

3.3.1. Exemplo da fábrica de papel

Uma indústria em que a optimização é fundamental é na indústria do papel. Imaginemos que estamos responsáveis pelo processo de corte numa fábrica de papel. Na nossa situação temos à disposição rolos de papel de tamanhos bem definidos para serem cortados, no entanto os diversos clientes da empresa querem rolos de tamanhos distintos uns dos outros. A pergunta que imediatamente surge é: "Como vamos cortar os rolos de modo a minimizar o desperdício de papel?" Verificamos que temos entre mãos um problema de optimização, que se traduz a nível matemático por um problema de integração linear.

Consideremos então o seguinte exemplo de um livro de Vasek Chvatal[8]:

Temos em stock rolos de 100_{cm} e quatro pedidos de comprimentos diferentes, sejam eles.

Quantidade	Comprimento do pedido
97	45
610	36
395	31
211	14

Tabela 2 – Pedidos e sua quantidade

Antes de mais devemos apresentar todas as hipóteses possíveis de cortar um rolo de papel com os pedidos que dispomos. A cada uma destas hipóteses chamamos de corte Padrão. Todas estas hipóteses estão demonstradas na tabela 3.

Hipótese	Pedidos	Total (cm)
1	45	45
2	45 45	90
3	45 36	81
4	45 36 14	95
5	45 31	76

6	45 31 14	90
7	45 14	59
8	45 14 14	73
9	45 14 14 14	87
10	36	36
11	36 36	72
12	36 36 14	86
13	36 36 14 14	100
14	36 31	67
15	36 31 31	98
16	36 31 14	81
17	36 31 14 14	95
18	36 14	50
19	36 14 14	64
20	36 14 14 14	78
21	36 14 14 14 14	92
22	31	31
23	31 31	62
24	31 31 31	93
25	31 31 14	76
26	31 31 14 14	90
27	31 14	45
28	31 14 14	59
29	31 14 14 14	73
30	31 14 14 14 14	87
31	14	14
32	14 14	28
33	14 14 14	42
34	14 14 14 14	56
35	14 14 14 14 14	70
36	14 14 14 14 14 14	84
37	14 14 14 14 14 14 14	98

Tabela 3 – Hipóteses possíveis de cortar a bobina

Representando matematicamente o problema atribuindo variáveis aos diversos módulos que temos, podemos definir como *x* – O número de cortes de cada um dos cortes padrão. De seguida temos que para cada padrão *j*, A*ij* é o número de peças cortadas de um determinado pedido *i* se *xj* for o número de vezes que cada padrão *j* é usado, então A*ij* * *xj* tem de ser maior ou igual ao número de pedidos para um determinado comprimento. A soma de todos os *xj* dá-nos a conhecer a quantidade de rolos necessários para satisfazer todos os pedidos dos clientes.

Agora que temos o caso planificado resta-nos resolver matematicamente o problema através de uma linearização. A condição de integração é comum em problemas de corte de stock o que os torna de difícil resolução. Com a ajuda computorizada é possível resolver estas equações de várias formas. Um dos métodos de resolução possíveis é denominado por "Simplex Method". Este método executa um processo iterativo de busca sucessiva de melhores soluções e só termina quando todas as hipóteses iterativas forem experimentadas. No final a solução apresentada é a que melhores resultados obteve. No entanto, à medida que o número de pedidos diferentes aumenta, irá aumentar exponencialmente o número de cortes padrão, o que, para quantidades consideráveis poderá dificultar ou mesmo tornar impossível o processamento da função linear. Surge então a pergunta se será mesmo necessário expor e usar todos os cortes padrão possíveis?

A resposta a esta pergunta apresenta-se na forma de um novo método designado por "Delayed Column Generation method"(DCG). Com esta nova solução não é necessário usar todos os padrões. Inicia-se o processo com apenas alguns cortes padrão e mais serão gerados à medida que forem sendo necessários. Os novos padrões a serem usados aparecem pela resolução do *"Knapsack problem"*.

Resumindo o processo, inicialmente escolhemos um conjunto de cortes padrão, se este conjunto não corresponder ao resultado óptimo (o que é mais provável de acontecer), existe a necessidade de criar um novo padrão, padrão esse que irá ser gerado através do *"Knapsack problem"*, e assim sucessivamente alternando um e outro problema de optimização até que já não existam mais padrões possíveis a serem criados. Deste modo aliviamos o processamento e conseguimos igualmente um resultado óptimo.

O procedimento a ter quando se usa o método DCG é o seguinte:

1. Seleccionar um conjunto de padrões para a linearização;
2. Resolver a linearização;
3. Resolver o "Knapsack problem";
4. Se o valor obtido no "Knapsack problem" Z> 1, retomar o passo 2;
5. Caso contrário aplicar uma integração linear para obter a solução.

O resultado obtido com este método é bastante satisfatório e permite resolver o problema de optimização do corte de bobinas. No entanto o processo usado foi outro, obtendo melhores soluções à custa de um maior esforço a nível de processamento.

3.4. Solução usada para o Problema Plafesa e seu aprofundamento

Dado o número de pedidos médios, a solução mais óptima que se pode ter é usar todas as combinações possíveis e daí extrair a solução pretendida.

Para dar a conhecer um pouco mais em detalhe o problema e também o porquê da solução encontrada, de seguida serão apresentados os dados mais relevantes em relação ao algoritmo usado.

O processo usado para apresentar ao utilizador um conjunto de soluções foi baseado no algoritmo matemático apresentado mais em cima neste documento (equações (3) e (4))

Vamos agora apresentar e explicar o programa feito passo por passo percorrendo toda a cadeia de acontecimentos. Em qualquer ponto, desde que oportuno serão apresentados pequenos trechos do código se este servir para clarificar alguma ideia.

O programa começa por pedir ao utilizador diversas características da bobina, entre elas a sua Largura, Peso, Espessura…!

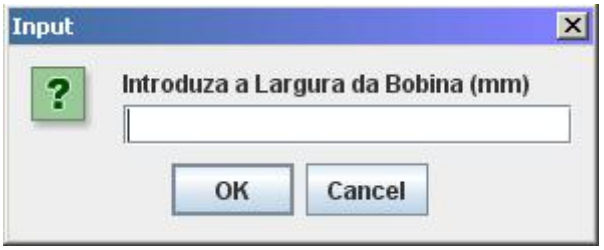

Figura 8 – Janela para introdução da Largura da bobina

Após a introdução dos valores característicos da bobina, o programa vai verificar se há pedidos.

Cada "pedido" é tratado como sendo uma sequência de dois números que correspondem ao peso e largura pretendidos. Estes valores estão armazenados num ficheiro externo (no presente caso pedidos.txt), que irá ser lido pelo bloco "PedidoInput". Como exemplo apresentamos uma possível lista de pedidos a tratar, retirada do ficheiro *pedidos.txt*. Esta lista é composta por três pedidos, sendo que o valor "banda" corresponde à largura máxima por banda e "peso" ao peso máximo total de cada pedido, ou seja, ao peso máximo que terá o pedido juntando todas as bandas que poderão ser necessárias.

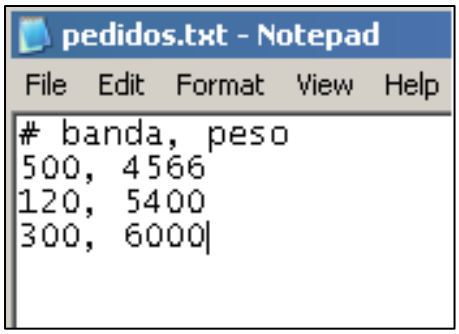

Figura 9 – Exemplo de um "pedido" efectuado

Como foi acima mencionado estes valores vão em seguida ser lidos pela classe *"PedidoInput"* que vai ler linha a linha o ficheiro *"pedidos.txt"* acima e guardar cada valor para que estes possam ser devidamente tratados. Em Java cada pedido está armazenado numa lista dinâmica de inteiros. O uso de listas dinâmicas tem como vantagem o facto de alocar espaço automaticamente caso novos valores sejam adicionados ao ficheiro "*pedidos.txt*". Assim deixa de ser necessário fazer à partida a reserva de espaço para uma determinada lista, acção esta que nos poderia conduzir à situação de num dado momento possuirmos mais pedidos do que aqueles que prevemos, logo necessitaríamos de mais espaço. Por outro lado o uso de listas dinâmicas permite que para o caso exposto de três pedidos, não seja necessário reservar tanto espaço, o que melhora o desempenho do programa.

Após o armazenamento na lista, cada pedido é devidamente tratado de forma a calcular, com base nas suas características, quantas bandas terão de ser cortadas para satisfazer este cliente. Para este cálculo é usada uma simples fórmula matemática. Sabendo quanto pesa a bobina e qual a sua largura, podemos chegar ao peso por metro, logo, dado que o corte não é interrompido, a partir da largura obtemos facilmente o peso de uma banda. Se com apenas uma banda não atingimos o peso ordenado teremos que cortar outra banda e assim sucessivamente até ao ponto em que o peso que falta ou o peso que temos a mais seja inferior a metade do peso de uma banda.

```
Pp= listaPedidos.get(pos).Peso;                                    // Peso do pedido a uma variavel
Pb= (bob.Peso/bob.Largura)*listaPedidos.get(pos).Banda;
PesoMetro = bob.Largura*bob.Espessura*bob.PesoEsp*1000/100000;
nbanda=1;
Pa=(nbanda*Pb);

if ( Pa < Pp ) {
        Pf = (Pp - Pa);                                    //O que falta para atingir o Peso do cliente
       //System.out.println( "Peso em falta = "+ Pf );
       While (Pf>(Pb/2))   {
              nbanda++;
              Pa = Pb*(nbanda);
              Pf = (Pp-Pa);
     }
          j = (Pa - Pp);                                   // o que o cliente leva a mais
           System.out.println( "Peso de cada banda = "+ Pb);
           System.out.println( "Número de bandas = "+ nbanda );
           System.out.print ( "Peso cortado = "+ Pa);
          System.out.print("   ->   ");
           System.out.println ( "Peso pedido = "+ Pp);
          //System.out.println("Peso Por metros das bandas: "+ PesoMetro);
           System.out.println("O cliente Leva " + j +" Kg de diferença.");
           System.out.println("");
}
```

```
else
        {
          j = (Pa - Pp);                                    // o que o cliente leva a mais
          System.out.println ("else Peso da banda = "+ Pb);
          System.out.println ( "else Numero de bandas = "+ nbanda);
          System.out.print ( "Peso cortado = "+ Pa);
          System.out.print("   ->   ");
          System.out.println ( "Peso pedido = "+ Pp);
         //System.out.println("Peso Por metros das bandas: "+ PesoMetro);
          System.out.println("O cliente Leva " + j +" Kg de diferença.");
          System.out.println("");
        }
```

Figura 10 – Código PedidoInput

Desta forma poderá acontecer que um determinado cliente leve mais ou menos material consoante o peso das bobinas e os pesos pretendidos. Seguidamente apresentaremos uma pequena ilustração (figura (11)) em que a título de exemplo usaremos uma bobina com uma largura de 1000_{mm} e um peso de 10000_{Kg} na qual vamos encaixar os pedidos listados acima.

```
Pedido 1: Banda 500 mm, Peso 4566 Kg
Peso da banda = 5000
Numero de bandas = 1
Peso cortado = 5000   ->   Peso pedido = 4566
O cliente Leva 434 Kg de diferença.

Pedido 2: Banda 120 mm, Peso 5400 Kg
Peso de cada banda = 1200
Número de bandas = 4
Peso cortado = 4800   ->   Peso pedido = 5400
O cliente Leva -600 Kg de diferença.

Pedido 3: Banda 300 mm, Peso 6000 Kg
Peso de cada banda = 3000
Número de bandas = 2
Peso cortado = 6000   ->   Peso pedido = 6000
O cliente Leva 0 Kg de diferença.
```

Figura 11 – Tratamento de cada pedido na bobina

Temos na figura acima o exemplo de um pedido em que se pretende obter rolos de chapa que tenham 500_{mm} de largura e que tenham um peso que no seu total iguale os 4566_{Kg}, podemos observar que para um só corte obtemos já o total de 5000_{Kg}, então este cliente irá levar 434_{Kg} a mais.

Já no segundo caso, o cliente necessita bandas de 120_{mm} e que pesem um total de 5400_{Kg}. Verificamos que cada banda tem o peso de 1200_{kg} e que com 4 cortes obtemos 4800_{kg}. Chegados a este ponto temos de decidir se cortamos mais uma banda e assim o cliente levaria mais 600_{kg} ou se pelo contrário o cliente leva menos 600_{kg}. Este é o tipo de gestão que terá obviamente que ser feito pela empresa tendo em conta múltiplos factores entre os quais factores humanos, para o programa gerado, uma vez que a quantia que falta é inferior a metade do peso de uma só banda admitimos que o cliente levará uma quantia em peso inferior ao inicialmente previsto.

Por fim temos o 3º e último caso em que o cliente quer bandas de 300_{mm} até perfazer um total de 6000_{kg} e para este cliente é possível inserir nesta bobina exactamente o que tinha inicialmente pedido. Nesta altura temos para cada um dos pedidos os respectivos números de cortes necessários para os satisfazer.

É neste ponto que é criada uma nova lista dinâmica que armazena o número de cortes necessários fazer para cada pedido, ou seja, armazenamos a largura total necessária para que um determinado corte seja satisfeito. Esta mesma lista será usada no algoritmo de combinação para que desta forma seja possível fazer uma combinação de todos os pedidos existentes.

O algoritmo combinatório começa então a tratar os valores os resultados.

No final da execução apenas os resultados considerados relevantes aparecem no ecrã. A cada combinação é atribuído um valor que representa uma taxa de ocupação na bobina. As várias combinações são comparadas e apenas as que têm uma taxa de ocupação superior a 98% são apresentadas no ecrã, podendo nesta fase o utilizador escolher qual a que mais lhe convêm e ordenar a sua execução. Recordemos rapidamente os valores obtidos a partir do exemplo que tem vindo a ser seguido.

Nesta bobina em particular os cortes podem ser feitos da seguinte maneira!

Vão ser cortadas 1 bandas de 500 mm de largura o que dá um total de: 500
Vão ser cortadas 4 bandas de 120 mm de largura o que dá um total de: 480
Vão ser cortadas 2 bandas de 300 mm de largura o que dá um total de: 600

Figura 12 – Encaixe de cada corte na bobina

Se para o exemplo acima exposto a apresentação de todas as hipóteses combinatórias seria bastante legível, não nos esqueçamos que matematicamente uma função de combinações é exponencial, rapidamente atingindo milhares de valores.

Combinações dos pedidos [500, 480, 600]

500
500, 480
500, 480, 600
500, 600
480
480, 600
600

Figura 13 – Diversas combinações dos pedidos

Como podemos observar na figura (13), algumas das combinações ultrapassam o valor da largura da bobina 1000_{mm}, pelo que estes podem ser imediatamente anulados poupando o utilizador e o próprio processamento.

Desta maneira, filtrando o resultado para que o utilizador tenha uma visão mais clara, com estes três pedidos obteríamos a seguinte saída:

Melhores resultados

****
500, 480, , score 98%
****

Fim de execução do programa!

Figura 14 – Resultado final com percentagem de utilização

De modo resumido e simples podemos ilustrar o processamento geral de um pedido na figura (15).

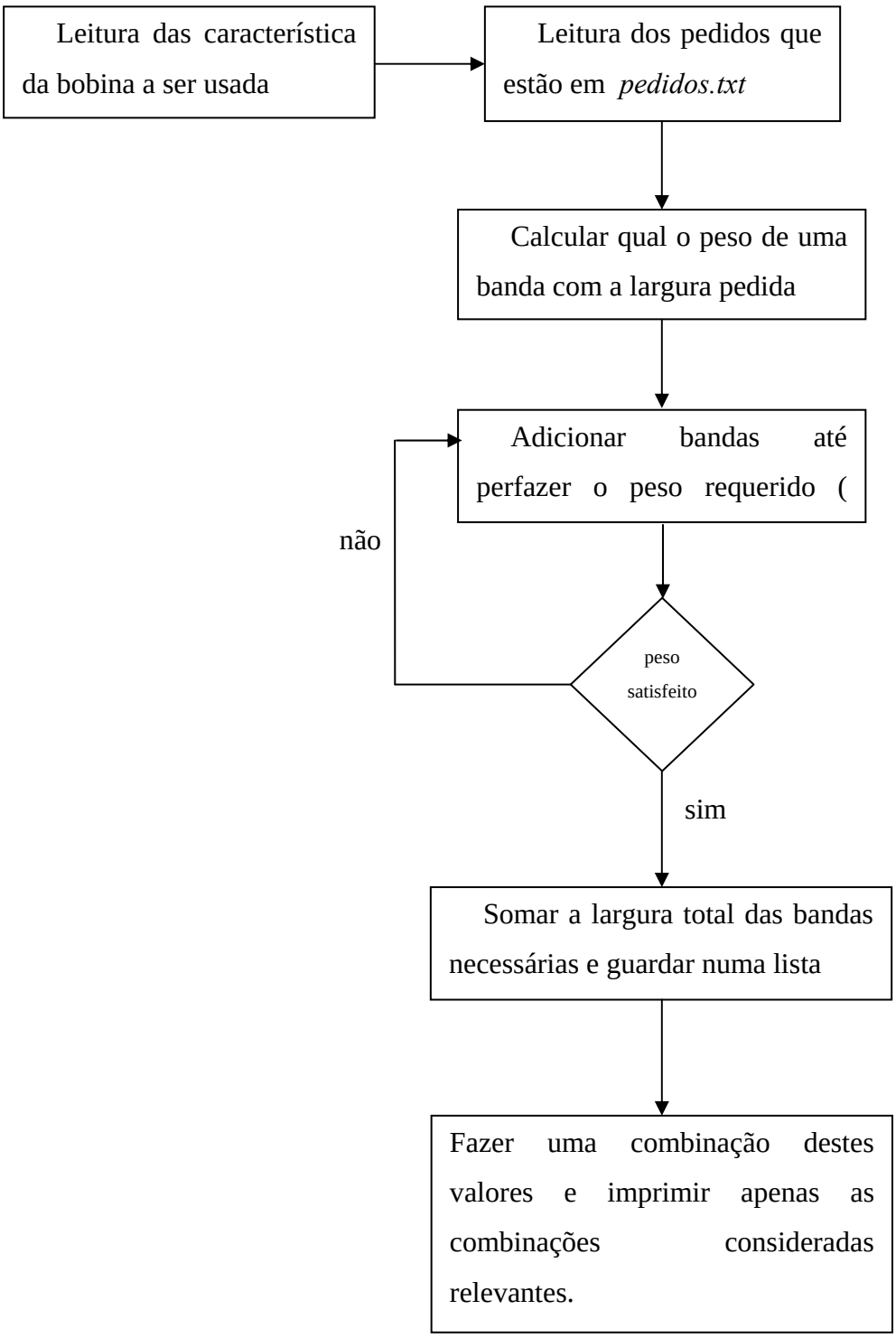

Figura 15 – percurso que um pedido faz, desde que é lido até ser inserido nas combinações.

Em baixo (figura (16)) pode-se observar um pequeno esquema ilustrativo do corpo global do programa, onde se percebe qual a interligação das diversas classes criadas e qual o seu papel e enquadramento geral no código.

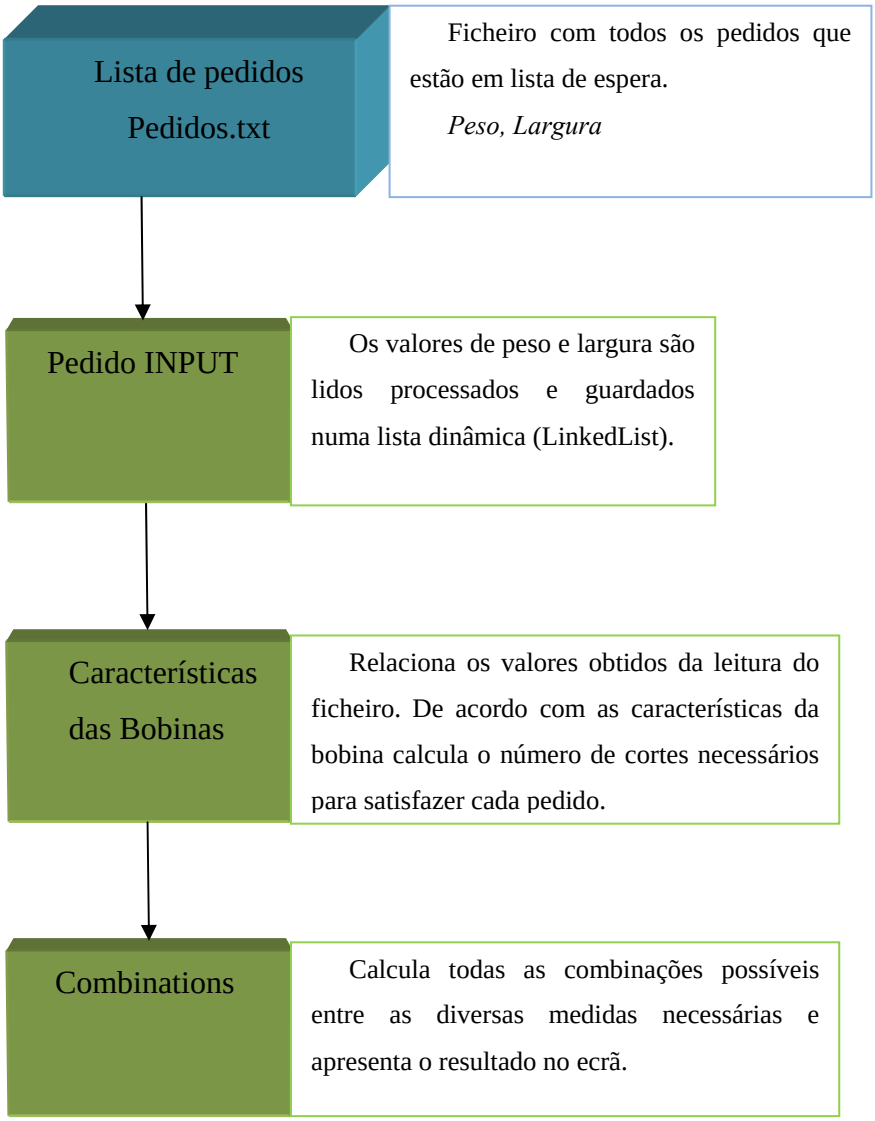

Figura 16 – Estrutura principal do programa Java

3.5. Análise de Desempenho

É extremamente importante efectuar um teste de esforço ao código feito. Avalia-se a capacidade do programa e conhecem-se os limites a partir dos quais será necessário usar outras metodologias. A nível empresarial este passo é fundamental, só assim se pode com alguma segurança projectar e gerir o crescimento de uma empresa.

De modo a ter uma visão empírica do programa elaborado, este foi testado com um incremento sucessivo de pedidos até ao ponto em que deixámos de conseguir bons resultados, ou seja, resultados em que o tempo dispendido para calcular as combinações possíveis se torna demasiado excessivo.

O resultado obtido deste “Capacity Planning” está apresentado na tabela e gráfico seguintes.

Nº de pedidos	tempo de resposta (s)
1	0,1
2	0,1
3	0,1
4	0,1
5	0,3
6	0,3
7	0,3
8	0,35
9	0,35
10	0,4
11	0,5
12	0,9
13	1,4
14	1,76
15	2,7
16	5
17	10
18	19
19	45
20	87

Tabela 4 – valores do primeiro teste ao código

Gráfico 1 – Curva de tempo de resposta em função do número de pedidos

Como conclusão deste teste, podemos observar que a partir de vinte pedidos o programa se torna cada vez mais lento o que poderia inviabilizar o seu uso. No entanto a média de pedidos ronda os dez o que faz com que estejamos em condições de usar este programa na situação actual. De referir que o teste acima foi realizado para uma situação crítica, em que escrevemos no ecrã todas as combinações entre os diversos valores dos pedidos. Como tentativa de melhorar o rendimento do programa, algumas modificações foram efectuadas no sentido de diminuir o processamento de CPU e assim conseguir melhorias ao nível de tempo de execução.

Outro teste foi efectuado, agora escrevendo apenas os valores que potencialmente interessam ao nosso utilizador. Concluiu-se que deste modo o tempo de execução desce bruscamente, o que permite aumentar o número de pedidos a ser executado. Os valores obtidos para este segundo teste foram os apresentados na seguinte tabela:

Nº de pedidos	tempo de resposta (s)
1	0,1
2	0,1
3	0,1
4	0,1
5	0,2
6	0,2
7	0,2
8	0,2
9	0,2
10	0,3
11	0,3
12	0,3
13	0,3
14	0,4
15	0,5
16	0,5
17	0,6
18	0,6
19	1
20	2

Tabela 5 – valores do segundo teste ao código

Do mesmo modo que para o teste anterior, é apresentado um gráfico para melhor ilustrar os tempos de resposta obtidos com a reformulação efectuada no código.

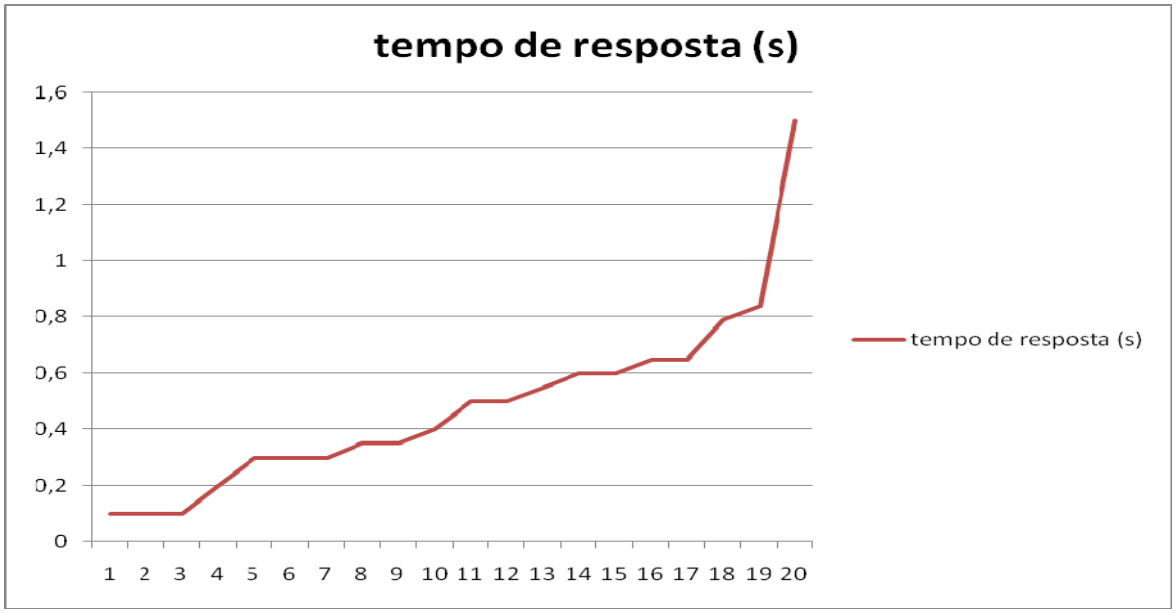

Gráfico 2 – Curva de tempo de resposta em função do número de pedidos

Como podemos observar, a diferença é notória. Executando até 26 valores de pedidos, foi obtido um tempo de execução de 1minuto e 20segundos.

Estes foram os testes executados na tentativa de medir o comportamento do programa quando temos mais ou menos o dobro da quantidade de pedidos média.

Ao usar este método de selecção podemos garantir tempos de resposta relativamente satisfatórios perante o elevado número de combinações processadas.

Capítulo 4

Conclusões e trabalho futuro

Com o objectivo de melhorar e aperfeiçoar o programa feito, alguns itens podem ser revistos.

Para um possível aperfeiçoamento, proponho que se crie uma interface gráfico com o utilizador. Desta maneira tornamos mais simples e intuitivo o uso deste programa.

Uma vez que o grupo Plafesa possui Bases de Dados Oracle, poderíamos integrar este programa com a Base de Dados o que alargaria as possibilidades de execução. Com este tipo de suporte poderíamos de forma também automática seleccionar que tipo de pedidos nos interessa que façam parte em cada execução do ficheiro "*pedidos.txt*", dependendo das características das bobinas disponíveis em cada ocasião. "Comunicando" com a base de dados relacional através de SQL, podemos extrair para um ficheiro externo pedidos acima de um determinado peso, com mais ou menos largura, de um ou outro cliente. Podemos ter todas as especificidades que correspondam à quantidade de informação que introduzimos na base de dados. Este ficheiro externo é o que será lido pelo programa de optimização. O utilizador tem assim bastante controlo sobre as operações efectuadas.

Com a integração com a Base de Dados, torna-se também muito mais fácil usar os dados de stock para o possível cálculo em diversas bobinas.

Tendo em vista resolver o problema encontrado com o teste de stress do programa, uma solução a ponderar seria a de estudar a possibilidade de usar multiprocessamento para o caso em que existam demasiadas ordens de clientes. No caso de ser viável mantínhamos a eficácia sem descurar a eficiência e disponibilidade do sistema.

Estes são alguns dos pontos que poderão ser úteis para um posterior seguimento deste trabalho.

Bibliografia

[1] - Dyckhoff, H. 1990. *A typology of cutting and packing problems*. European Journal of Operational Research.

[2] - Hinxman, A. I. 1979. *The trim-loss and assortment problems*: A survey. European Journal of Operational Research.

[3] - Eisemann, K. 1967. *The trim problem. Management Science*, 3: 279–284.

[4] - Gilmore, P. C., & Gomory, R. E. 1961. *A linear programming approach to the cutting stock problem*. Operations Research,

[5] - H. Land and AG Doig, "An Automatic Method of Solving Discrete Programming Problems," Econo met- rica, 28: 497-520,

[6] - Feo, T. A., & Resende, M. G. C. 1995. *Greedy randomized adaptive search procedures*. Journal of Global Optimization, 6:109–133, 1995.

[7] - Colorni, A., Dorigo, M., & Maniezzo, V. 1991. Distributed optimization by ant colonies.

[8] - Chvátal, V. (1983). *Linear Programming*. W.H. Freeman

[9] - Thomas Weise: *Global Optimization Algorithms – Theory and Application.*

[10] - Kantorovich, L. V. 1960. Mathematical methods of organizing and planning production. Management Science.

[11] - Haessler, R. W. 1975. Controlling cutting pattern changes in one-dimensional trim problems. Operations Research, 23: 483–493.

[12] - Haessler, R. W., & Sweeney P. E. 1991. Cutting stock problems and solution procedures. European Journal of Operational Reserach, 54: 141–150.

[13] - Gerhard Wa¨scher, Heike Haußner, Holger Schumann 2005, *An improved typology of cutting and packing problems*, European Journal of Operational Research 183, 1109-1130

[14] - Janne Karelahti, 2002, *Solving the cutting stock problem in the steel industry*, Master's thesis, Espoo, HELSINKI UNIVERSITY OF TECHNOLOGY

[15] - Bruce Eckel, 2000, Thinking in Java, 2nd Edition, release 11, Prentice-Hall

[16] – Alves, William Pereira, 2006, JAVA 2: Programação multiplataforma, 1. Ed, Érica.

[17] – Cay S. Horstmann e Gary Cornell, 2006, Core JAVA 2. Volumen I-Fundamentos, 7th Ed, PEARSON EDUCACIÒN

[18] - Cay S. Horstmann e Gary Cornell, 2006, Core JAVA 2. Volumen II-Fundamentos, 7th Ed, PEARSON EDUCACIÒN

[19] - L. Pitsoulis and M.G.C. Resende (2002) Greedy randomized adaptive search procedures. In P.M.Pardalos and M.G.C.Resende, editors, *Handbook of Applied Optimization*, pp. 168–181, Oxford University Press.