UNIVERSIDADE ESTADUAL DO CEARÁ

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DO CEARÁ

EMANUEL FERREIRA COUTINHO

ALGORITMOS DE ESCALA E ROTEAMENTO DE VEÍCULOS PARA APLICAÇÃO EM SERVIÇOS SISTEMÁTICOS DE REGIÕES URBANAS

Fortaleza – Ceará
2003

UNIVERSIDADE ESTADUAL DO CEARÁ

CENTRO FEDERAL DE EDUCAÇÂO TECNOLÓGICA DO CEARÁ

EMANUEL FERREIRA COUTINHO

ALGORITMOS DE ESCALA E ROTEAMENTO DE VEÍCULOS PARA APLICAÇÃO EM SERVIÇOS SISTEMÁTICOS DE REGIÕES URBANAS

Dissertação apresentada ao curso de Mestrado Integrado Profissional em Computação, da Universidade Estadual do Ceará e do Centro Federal de Educação Tecnológica do Ceará, como requisito parcial para obtenção do grau de Mestre em Computação.

Área Temática: Sistemas de Apoio a Decisão.
Sub-Área Profissional: Sistemas de Otimização

Orientador: Prof. DSc. Marcos José Negreiros Gomes

Fortaleza – Ceará
2003

COUTINHO, EMANUEL FERREIRA

Algoritmos de Escala e Roteamento de Veículos para Aplicação em Serviços Sistemáticos de Regiões Urbanas [Fortaleza] 2003

XII, 132 p. 29,7 cm (MPCOMP – UECE/CEFET-CE, MSc. em Computaçao - Sistemas de Apoio a Decisão), 2003

Dissertação – Universidade Estadual do Ceará e Centro Federal de Ensino Tecnológico do Ceará, MPCOMP
1. Meta-heurísticas
2. Problema de Escala Periódica de Veículos
3. Problema de Roteamento em Arcos

I.MPCOMP/UECE-CEFET-CE II. Título (Série)

A meus pais, Inêz e Milton, minha esposa Ana Paula, e meu filho Lucas.

Agradecimentos

Agradeço primeiramente a Deus, por ter me dado forças para sempre seguir na linha.

Também agradeço aos meus amigos da Graphvs, Augusto e Pedro, que me ajudaram no desenvolvimento do trabalho.

Agradeço ao meu orientador, o professor Marcos Negreiros, por ter me dado este trabalho que tanto me fez feliz, e por ter me ajudado nesta caminhada.

Aos companheiros do Instituto Atlântico, que também me ajudaram, os meus agradecimentos.

E agradeço finalmente a meus pais, Inêz e Milton, que me deram condições para chegar aonde estou, e que tanto me ajudaram nesta vida.

Resumo

Aplicações de escalonamento e roteamento de veículos em regiões urbanas de serviços sistemáticos, ou seja, que acontecem com uma certa freqüência, e atendem a certas regiões, são bastante utilizadas nos dias atuais. Problemas de distribuição de recursos ocorrem muitas vezes em tarefas que são realizadas manualmente, e sem muita precisão e alocação correta. Qualquer alteração que seja feita implica em muito re-trabalho. Na maioria dos casos onde estes problemas ocorrem, não existem pessoas ou softwares adequados para a construção de zonas, escalas e rotas, além da necessidade da constante alteração de informações, pois são problemas muito dinâmicos. Apesar de existirem softwares que realizam estas atividades, geralmente eles são muito complexos e exigem um nível de especialização do usuário bastante elevado.

Como as cidades vêm crescendo rapidamente e inúmeros tipos de serviços são criados e utilizados, um processo de automação do planejamento viria agilizar e facilitar muito a realização de tarefas que são muito difíceis para um homem realizá-las.

O objetivo principal deste trabalho é propor novos métodos para solucionar alguns problemas encontrados em regiões urbanas que utilizam serviços de entrega/coleta/controle domiciliar sistemáticos, cujas aplicações são comumente denominadas de escalonamento e roteamento de veículos.

Para o problema de roteamento, será proposta uma nova solução para o Problema do Carteiro Rural Misto, com aplicação de algoritmos de percurso e fluxo mínimo.

Finalmente, serão mostradas duas aplicações adaptadas para a resolução destes problemas.

Abstract

Applications of scheduling and vehicle routing in urban regions of systematic services, that happen with certain frequency, and attend some regions, are commonly used in the present days. Problems related to distribution resources are tasks that in many times are realized manually or semi-automatically, and without precision considering the right/best allocation. Any changes in the process may implicate in more work to do. In the most case, there is no people suitable the work with this problem, once the necessity of constant changes in the information because they are dynamic problems. There are some software that solves these kind of problem, but generally they are too complex and require a great level of knowledge of the user.

Once cities are growing fast, and many kinds of services are created and used, an automatic process considering optimization criteria based resolution will make agile the resolutions and will facilitate the realization of activities that are too difficult for a man make it.

The principal objective of this work is to propose new methods to resolve some problems founded in urban regions that use systematic service of resource delivery, gathering and control, and in this case scheduling and route problems.

To the scheduling problem, we propose a mathematical model for the vehicle allocation to days of service and regions, and the use of meta-heuristics to generate solutions of scales.

For the routing problem, we propose a solution for the Mixed Rural Postman Problem, with the applicattion of route algorithms and minimum flux algorithm.

Finally, two applications were adapted to be test instances to these problems.

Sumário

Índice de Figuras

Índice de Tabelas

Introdução

Aplicações em regiões urbanas de escalonamento e roteamento de veículos para serviços sistemáticos, ou seja, que acontecem com uma certa freqüência, e atendem a certas regiões, são bastante utilizadas nos dias atuais. Problemas de distribuição de recursos são tarefas que muitas vezes são realizadas manualmente, e sem muita precisão e alocação correta. Qualquer alteração que seja feita implica em muito retrabalho. Na maioria dos casos onde estes problemas ocorrem, não existem pessoas ou programas adequados para a construção de zonas, escalas e rotas, além da necessidade da constante alteração de informações, pois são problemas dinâmicos. Apesar de existirem programas (a maioria de outros países) que realizam estas atividades, geralmente eles são muito complexos e exigem um nível de especialização do usuário bastante elevado.

Como as cidades vêm crescendo rapidamente e inúmeros tipos de serviços são criados e utilizados, um processo de automação do planejamento viria agilizar e facilitar muito a realização de tarefas que são muito difíceis para um homem realizá-las. Além disto, uma vez disponibilizada uma ferramenta com este grau de sofisticação, haverá um ganho expressivo para diversos setores como, por exemplo, a área de saúde (combate de doenças tropicais, como dengue, febre amarela, calazar, e outras), coleta/entrega de correspondências, leitura de medidores (água e luz), inspeção de redes (telefonia, dutos, etc), distribuição do gás engarrafado, coleta do lixo, e outros.

Vários problemas de otimização combinatória podem ser aplicados na prática. Muitas vezes estes problemas possuem soluções viáveis e exatas na teoria, mas quando aplicadas à prática, muitas vezes não compensam devido às condições reais de operacionalização, e infra-estrutura de equipamento para solução. Em certos casos, uma solução exata pode ser muito complicada de se aplicar, e uma solução aproximada poderia ser bem mais vantajosa.

Problemas de escalonamento consistem basicamente em associar recursos para a execução de atividades ao longo do tempo, [BEPSW96].

O escalonamento pode ser aplicado a diversas áreas, mas a forma de escalonamento que abordamos neste trabalho está voltada especificamente para a área de transportes ou logística. Em outras palavras, esta dissertação tratará o escalonamento como sendo uma distribuição de veículos num determinado período de tempo, onde visitas serão executadas a determinadas regiões, realizando algum tipo de serviço. Estes serviços podem ser: entrega de correspondências, coleta de lixo, leitura de medidores de água ou energia elétrica, etc.

Normalmente, as empresas fazem a construção das escalas de trabalho manualmente ou de forma semi-automática. Estas maneiras de construção são demoradas e estão sujeitas a erros de cálculo ou imprecisões quanto à tomada da decisão. Qualquer que seja a modificação na escala implica num grande retrabalho, ainda mais que diversas variáveis estão envolvidas na geração de uma escala, como freqüência, quantidade de recursos, tempo de execução, quantidade de regiões a serem visitadas, folgas, etc.

Existem diversas aplicações para problemas de escalonamento, e em particular podemos citar: alocação de caminhões a rotas ou regiões, distribuição dos funcionários a horários de trabalho, melhor distribuição dos recursos da empresa (frota, capacidade, empregados, horários de trabalho), escala de funcionários e serviços, escalas de trabalho para tripulação ferroviária, aérea e rodoviária, e escalas de trabalho para motoristas de ônibus, dentre outras, [BG83].

O problema de roteamento de veículos (geração de percursos) é um dos mais importantes problemas de otimização, podendo ser aplicado a várias situações reais. Muitos trabalhos foram desenvolvidos nesta área ao longo dos últimos 50 anos, e é considerado um dos problemas de maior sucesso do ponto de vista da aplicação na Pesquisa Operacional, [BG83].

Roteamento de veículos consiste em descrever o percurso que um elemento (veículo, pessoa, etc) deve seguir, passando por vários locais, que podem ser obrigatórios ou não, e retornar à origem ou seguir para um local de término definido.

Muitos pesquisadores já desenvolveram trabalhos tanto teóricos quanto práticos sobre problemas de roteamento, utilizando diversas técnicas para sua solução. Alguns pacotes prontos também existem, mas a particularidade de cada problema, de cada empresa, são os mais variados, o que faz com que os problemas de roteirização sejam muito difíceis de se resolver mesmo computacionalmente.

O enfoque desta dissertação é no roteamento de veículos em regiões urbanas, onde existem vias de sentido único e sentido duplo. Considerando a literatura, estaremos tratando do problema de roteirização em arcos, [EU36] e [HE73].

Existem diversas aplicações para problemas de roteirização em arcos, onde citamos: percurso de leituristas de água e luz, varrição de ruas, distribuição de garrafões de água, botijões de gás, remoção de neve das ruas, distribuição de produtos atacadistas a clientes (bebidas, alimentos, etc.), coleta de lixo, transporte escolar, serviços de entregas postais, deslocamento de ambulâncias, bombeiros e rádio-patrulhas e percurso de Veículos, [NEG96].

O objetivo principal deste trabalho é propor novos métodos para solucionar alguns problemas encontrados em regiões urbanas que utilizam serviços de entrega/coleta/controle domiciliar sistemáticos, cujas aplicações são comumente denominadas de escalonamento e roteamento de veículos.

Basicamente, como já dissemos, serão tratados dois problemas: o escalonamento e o roteamento de veículos, com particularidades discriminadas e propostas de soluções.

Durante o desenvolvimento do trabalho, alguns problemas que foram estudados e implementações realizadas voltadas para situações reais serão descritos. Algumas destas soluções podem ser ótimas na teoria, mas para a prática podem não ter uma operacionalização tão simples, ou necessitarem de muitas alterações devido a várias dificuldades de implantação.

A maioria das empresas que possuem estes problemas não têm profissionais qualificados ou algum tipo de software adequado para a construção de zonas, escalas e rotas, além da necessidade da constante alteração nestes valores e testes de novas situações, pois estes problemas são muito dinâmicos.

No mercado existem vários softwares desenvolvidos para o auxílio da confecção de escalas de serviços e rotas (http://mapinfo.com), mas normalmente são programas muito caros e complexos de se utilizar, exigindo um nível de especialização do usuário muito elevado, inviabilizando a aquisição e manuseio pela maioria das empresas.

Neste trabalho, avaliamos os problemas de escalonamento e roteamento de veículos através do estado da arte dos algoritmos e literatura da área de Otimização Combinatória. Muitos artigos ainda não publicados em revistas internacionais são também incluídos em nosso acervo de pesquisa, de modo a enriquecer o conhecimento e dar segmento aos caminhos que serão empreendidos no futuro dos assuntos abordados.

Para avaliar os procedimentos que construímos, faremos uso de implementações que já são consagradas, e outras que foram cedidas pela empresa GRAPHVS Cons. Com.e Rep. Ltda, (www.graphvs.com.br), as quais são baseadas em trabalhos desta empresa em cooperação com a UECE e UFES. O sistema principal utilizado para o desenvolvimento dos algoritmos do Carteiro Rural Misto, foi o sistema XNÊS, já parte do trabalho de escalas, está baseado no sistema SisRot® TRANSLIX, de propriedade da GRAPHVS.

O presente trabalho está dividido em sete capítulos, incluindo esta introdução, com uma breve introdução aos problemas estudados, com suas descrições e aplicações.

No capítulo I é feita uma revisão bibliográfica sobre as técnicas utilizadas na implementação dos problemas. Estas técnicas são compostas por meta-heurísticas, algumas das quais serão utilizadas para melhoria das soluções dos problemas, algoritmos de percurso, e técnicas para resolução de problemas.

No capítulo II é feito um desenvolvimento temático sobre o problema de escalonamento, voltado para a área de transportes, onde serão descritos diversos problemas e algoritmos, além de uma formulação matemática e meta-heurísticas para melhoria, e a apresentação dos resultados obtidos.

O capítulo III propõe uma solução para o Problema do Carteiro Rural Misto, através de manipulações em grafos e aplicação de algoritmos do Problema do Caixeiro Viajante Generalizado e de fluxo de custo mínimo com restrições canalizadas, e apresentação dos resultados obtidos.

Finalmente terminamos o trabalho com uma conclusão e sugestões para trabalhos futuros que possam vir a ser estudados ou derivados desta dissertação, seguido de uma bibliografia, composta por artigos, livros, dissertações e teses de doutorado com uma relativa amplitude sobre o estado da arte nestes dois assuntos da Pesquisa Operacional.

1. Revisão bibliográfica

1.1 – Introdução

Neste capítulo, é feita uma revisão bibliográfica sobre os diversos assuntos abordados no desenvolvimento deste trabalho. Apresenta-se uma breve explicação sobre as meta-heurísticas, problemas de caminho mínimo, o problema do caixeiro viajante e uma variação deste – o problema do caixeiro generalizado, o problema de fluxo em redes de custo mínimo com restrições canalizadas, e o problema do carteiro rural, assim como os respectivos algoritmos que são utilizados durante o desenvolvimento do trabalho.

1.2 – Métodos de Busca em Vizinhança

Dado um conjunto finito **P**={1,...,*n*}, pesos c_j para cada $j \in \mathbf{P}$, e um conjunto $\Im$ de subconjuntos (soluções) viáveis de **P**. O problema de encontrar um subconjunto (uma solução) viável de peso mínimo é dito ser um Problema de Otimização Combinatória (POC), e pode ser escrito como segue, [WOL98]:

$$(POC)\ \min_{S \subseteq P} \{\sum_{j \in S} c_j : S \in \Im\} \qquad [1]$$

Uma vizinhança $\boldsymbol{N}(\boldsymbol{x},\boldsymbol{\sigma})$ de uma solução $\boldsymbol{x}$ de um problema de Otimização Combinatória, é um conjunto de soluções viáveis que podem ser alcançadas a partir de $\boldsymbol{x}$ através da operação $\boldsymbol{\sigma}$. Esta operação $\boldsymbol{\sigma}$ pode ser a remoção ou a adição de um objeto da solução $\boldsymbol{x}$, ou a troca entre dois objetos da solução, [CRR93].

Conforme Aarts e Lenstra, dada uma instância (uma forma em que os dados de um problema se apresenta) de um problema, cada solução $\boldsymbol{x}$ que ela gerar é definida como sendo uma vizinhança, [AL97].

O termo vizinhança sugere uma idéia de proximidade entre soluções viáveis, já que soluções vizinhas são obtidas as partir de pequenas perturbações de uma solução de partida. A função de vizinhança que descreve o processo pode ser muito complicada, podendo ser assimétrica, ou seja, a instância $\boldsymbol{S_i}$ pode possuir $\boldsymbol{S_j}$ como vizinho, mas $\boldsymbol{S_j}$ pode não possuir $\boldsymbol{S_i}$ como vizinho.

Um exemplo disso é o problema de particionamento de grafos. Este problema divide um grafo em partes, de modo que em cada uma existam vértices, sendo estas disjuntas entre si. Todas as partições geradas a partir de trocas entre os grupos (de partes) são ditas serem as vizinhanças.

Em geral, quanto maior a vizinhança, mais difícil de explorá-la e de achar um ótimo local, porém em geral presume-se que o ótimo local tenha boa qualidade. A maior vizinhança é na vizinhança exata, uma em que cada ótimo local também é global. Neste caso, o problema de busca local coincide com o problema de otimização global.

Uma visão sistêmica de um método de exploração de vizinhança pode ser colocada, segundo Reeves (1993), como mostra a figura 1.1, [CRR93]:

[01] **Procedure** Método de Busca em Vizinhanças
[02] Passo 1 : Inicialização
[03] (A) Selecione uma solução inicial $\boldsymbol{x}^{\mathbf{now}} \in \boldsymbol{S}$
[04] (B) Armazene a melhor solução corrente $\boldsymbol{x}^{\mathbf{best}} = \boldsymbol{x}^{\mathbf{now}}$
[05] Faça **bestCost** = $\mathbf{c}(\boldsymbol{x}^{\mathbf{best}})$
[06] Passo 2 : Escolha e terminação
[07] Escolha uma solução $\boldsymbol{x}^{\mathbf{next}} \in \mathbf{N}(\boldsymbol{x}^{\mathbf{now}})$
[08] Se o critério de escolha não é satisfeito por nenhum membro de $\boldsymbol{x}^{\mathbf{now}}$ ou se outro critério de terminação se aplicar, então o método pára
[09] Passo 3 : Atualização
[10] Faça $\boldsymbol{x}^{\mathbf{now}} = \boldsymbol{x}^{\mathbf{next}}$
[11] Se $\mathbf{c}(\boldsymbol{x}^{\mathbf{now}}) <$ **bestCost** então execute Passo 1B e volte ao Passo 2
[12] **end** {MBV}

Figura 1.1: Método Genérico de Busca em Vizinhanças.

O custo da função é definido pela variável c, e S é o conjunto de soluções, e cada solução $x \in S$ tem associado um grupo de soluções, $N(x) \subset S$, chamado de vizinhança de x.

1.3 – Meta-heurísticas

Em função do grau de dificuldade de muitos problemas de otimização combinatória, tais como problemas de escalonamento, roteamento, zoneamento, cortes, etc, que têm como característica muitas restrições e variáveis inteiras, instâncias muito grandes poderiam se tornar computacionalmente intratáveis.

Um resultado exato poderia não ser tão interessante se gastasse muitos recursos de máquina ou tempo para a sua obtenção, e na prática ainda poderia ser difícil de realizá-lo. Neste caso, uma aproximação poderia ajudar muito, evitando desperdício de recursos e tempo, e sendo bem mais interessante na prática.

Diversos algoritmos heurísticos de baixo custo computacional têm sido desenvolvidos com a finalidade de se encontrar boas soluções na prática, e que não necessariamente sejam soluções ótimas.

Entretanto, dada uma instância de um problema, quando este é resolvido por uma heurística gulosa ou por qualquer heurística determinística, somente uma solução poderia ser obtida, e que poderá ser ou não ótima, [EVN99]. Porém, se a heurística for de busca local, várias soluções poderão ser geradas.

A figura 1.2 mostra a representação de somente uma solução encontrada.

Figura 1.2: Representação de uma solução para uma instância de problema.

De acordo com o conceito de que quanto mais soluções viáveis são conhecidas para uma mesma instância de um problema, maior será a probabilidade de se encontrar uma solução ótima. Esta idéia pode ser vista na figura 1.3.

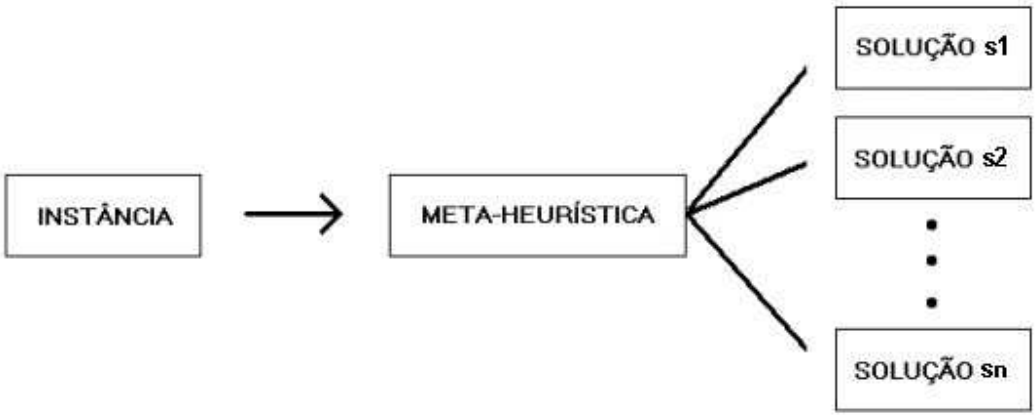


Figura 1.3: Representação de soluções obtidas da resolução por meta-heurística de uma instância de um problema

Normalmente, as meta-heurísticas obedecem a um determinado critério de parada, que geralmente é especificado por quem as utiliza. Estes critérios podem ser: tempo máximo de processamento, número máximo de iterações ou número máximo de iterações sem melhoria.

As meta-heurísticas são em geral consideradas como métodos de busca probabilístico de exploração de soluções, encaixando-se aí uma forte relação aos métodos Monte Carlo e Las Vegas, [CRR93].

Se o Passo 2 do método de busca de vizinhanças apresentado na figura 1.1 for alterado pelas seguintes linhas da figura 1.4., e supondo que é um problema de minimização, ele se tornará um método chamado de Monte Carlo.

[1] **Passo 2** : Escolha e terminação
[2] (A) Escolha aleatoriamente $\boldsymbol{x}^{\mathbf{next}}$ a partir de $\mathbf{N}(\boldsymbol{x}^{\mathbf{now}})$
[3] (B) Se $\mathbf{c}(\boldsymbol{x}^{\mathbf{next}}) \leq c(\boldsymbol{x}^{\mathbf{now}})$ então aceite $\boldsymbol{x}^{\mathbf{next}}$ e vá ao Passo 3
[4] (C) Se $\mathbf{c}(\boldsymbol{x}^{\mathbf{next}}) > c(\boldsymbol{x}^{\mathbf{now}})$ então aceite $\boldsymbol{x}^{\mathbf{next}}$ com uma probabilidade que diminui com o aumento da diferença $\mathbf{c}(\boldsymbol{x}^{\mathbf{next}}) - c(\boldsymbol{x}^{\mathbf{now}})$
[5] Se $\boldsymbol{x}^{\mathbf{next}}$ não for aceito durante a tentativa atual por este critério, retorne a (A)
[6] (D) Termine através de uma regra de parada escolhida

Figura 1.4: Método de Busca em Vizinhanças.

1.3.1 – GRASP

Dentre as meta-heurísticas, o método GRASP (*Greedy Randomized Adaptative Search Procedure*) foi desenvolvido na década de 80 por Feo & Resende. Ele se baseia em relaxar uma solução originada de uma heurística gulosa, que é uma solução para cada iteração, de forma a obter um conjunto de elementos para a construção da solução. Ele utiliza escolhas aleatórias de decisão dos conjuntos de possibilidades de solução. Construções gulosas conduzem a soluções perto de ótimos locais, acelerando a busca local. Esta meta-heurística é bastante detalhada em Feo & Resende (1995) e pode ser colocada genericamente como na figura 1.5, [FR95], [RIAR98], [PRARIB99].

```
[1] procedure GRASP;
[2]   f (S*) ← +∞
[3]   for i: = 1,…,N do
[4]         Construir uma solução S usando um algoritmo guloso aleatorizado adaptativo e
iterativo
[5]      Aplicar um procedimento de busca local a partir de S, obtendo a solução S'
[6]      if f (S') < f (S*) then S* ← S'
[7]   end-for
[8] end {GRASP}
```

Figura 1.5: Algoritmo genérico para o GRASP.

Onde:

$f(\boldsymbol{S}^*)$ é o valor da função objetivo, de minimização, sobre a solução **s***;

$f(\boldsymbol{S}')$ é o valor da solução corrente obtida;

N é o número de iterações do GRASP.

Na linha [4] do procedimento meta-heurístico, uma solução é construída a partir de um procedimento guloso, onde a etapa de seleção construtiva deste algoritmo é aleatorizada, ou seja, a próxima escolha gulosa é randomizada entre possíveis escolhas, e também adaptativo e iterativo.

A fase de construção tem como objetivo a construção de uma solução viável, utilizando um elemento por vez. Cada iteração da fase de construção utiliza uma função gulosa para avaliar o benefício de incluir cada elemento na solução. Deve-se criar uma lista restrita de

candidatos, constituída pelos elementos que tiveram a melhor avaliação. A partir desta lista, seleciona-se aleatoriamente um elemento, e adaptam-se as informações do problema de acordo com o elemento incluído, ou seja, adaptar a função gulosa baseada na decisão do elemento que foi incluído. Informações sobre esta fase podem ser encontradas em [FR95] e [RIAR98].

A escolha do elemento a ser incluído na próxima etapa do método construtivo é determinada colocando-se os elementos ainda não escolhidos em uma lista ordenada de acordo com a função gulosa.

Restrições podem ser aplicadas à lista de candidatos, tais com o número máximo de elementos da lista ou a qualidade dos elementos da lista, em relação à escolha gulosa. É feita uma seleção na lista de candidatos dos melhores elementos, não necessariamente o elemento do topo da lista, onde a qualidade média da solução depende da qualidade dos elementos da lista, e a diversidade da solução depende da cardinalidade da lista. A diversificação é baseada na aleatorização de soluções, pois a cada iteração do procedimento GRASP, novas soluções são geradas.

A fase de construção não necessariamente termina em um ótimo local, mas o fato de ficar próximo dele acelera a busca local, e quase sempre a busca pode melhorar a solução achada.

Na linha [5], uma melhoria é testada dentro da solução ***S'*** construída, utilizando procedimentos de troca ou refinamentos genéricos de solução os quais podem ser procedimentos específicos relativos ao problema em análise, ou mesmo uma meta-heurística (como o VNS, por exemplo), [BMN99].

Para que a busca local seja eficiente, é preciso tomar cuidado com a escolha da vizinhança apropriada, de uma estrutura de dados que acelere o processo de busca local, e de boas soluções iniciais para acelerar a busca local. A utilização de um algoritmo guloso na fase de construção pode permitir que as buscas locais sejam mais rápidas.

Por fim, na etapa [6], guarda-se a melhor solução atingida até o momento e continua-se o processo enquanto não for atingido o número de iterações ou outro critério de parada. Detalhamentos e resultados de implementações encontram-se em [FR95], [RIAR98], [PRARIB99] e [BMN99].

A qualidade da solução encontrada é intrínseca à técnica de amostragem repetitiva no espaço de busca. Cada iteração GRASP age como se estivesse obtendo uma amostra de uma

distribuição desconhecida, onde a média e variância da amostragem dependem das restrições impostas na criação da lista restrita de candidatos. Quando uma solução não é aleatorizada no passo [2], ou simplesmente a seleção é sempre constante (como no método guloso convencional), temos um algoritmo GRASP puramente guloso, que encontra a mesma solução em todas as iterações. Já quando se faz uma amostragem de candidatos à medida que se evolui o procedimento, os resultados obtidos podem ser ruins em algumas iterações, porém em outras muito boas devido às muitas possibilidades de amostragem do espaço de vizinhança de soluções.

A nosso modo de ver, e concordando com os autores da meta-heurística, a grande vantagem do GRASP está justamente no aproveitamento do desempenho das heurísticas gulosas, geralmente de complexidade polinomial de baixo índice. A facilidade de paralelização e a pouca dependência a parâmetros de amostragem e ajuste (espaço de aleatorização e número de iterações) desta meta-heurística são outros importantes pontos de vantagem, os quais podem ser explorados para atingir desempenhos melhores.

O GRASP é simples de se implementar, tanto os algoritmos de construção quanto os de busca local, e heurísticas gulosas são simples de se projetar e implementar. Poucos parâmetros devem ser ajustados, ou seja: restrições da lista de candidatos (tamanho e qualidade) e o número de iterações. Ele depende de boas soluções iniciais, pois como é baseado em aleatorização de uma iteração para outra, cada iteração se beneficia da qualidade da solução inicial.

Tem como desvantagem a falta de memória das informações obtidas durante a iteração. A utilização de filtros pode acelerar a busca local, se aplicada a melhor solução construída ao longo de uma seqüência de aplicações do algoritmo guloso aleatorizado ou às soluções construídas que satisfazem a algum limiar de aceitação. Uma técnica de filtragem é conhecida como *Path Relinking*, sendo uma forma de geração de percurso (memória) entre duas soluções de boa qualidade, mantendo uma memória de trocas importantes entre duas soluções, [RIAR98].

Diversas aplicações podem ser resolvidas com o auxílio do GRASP, tais como: sistemas de Steiner, sistemas de manufatura flexível, *scheduling* de linhas aéreas, Produção e planejamento programando auxiliado por computador, problemas de localização, coloração

em grafos, sistemas de planejamento de transmissão de potência e roteamento de veículos, [NDSA00]. Uma aplicação em cortes unidimensionais se encontra em [EVN99].

1.3.2 – VNS

O VNS (*Variable Neighbourhood Search*), desenvolvido inicialmente por Mladnovic (1995), é uma meta-heurística baseada num princípio simples: trocas sistemáticas de vizinhanças durante a busca. Maiores detalhes, variações, aplicações e implementações se encontram em [HAM98], [HM98] e [HM00]. Um algoritmo genérico para o VNS está colocado na figura 1.6.

[1] **procedure** VNS
[2] Construir uma solução inicial **S** usando um algoritmo heurístico
[3] $k = 1$
[4] **while** uma condição de parada em **S** seja atendida $k = k_{max}$ **do**
[5] **begin**
[6] Gerar randomicamente S' a partir da k-ésima vizinhança de S ($S' \in N_k(S)$)
[7] Aplicar um procedimento de busca local com **s'** como solução inicial
[8] Ao final da busca em $N_k(S)$, obter S''
[9] **if** $c(S') < c(S')$ **then** $S' \leftarrow S''$ e continue a busca com $N_1(k \leftarrow 1)$
[10] **senão** $k = k + 1$
[11] **end-while**
[12] **end** {VNS}

Figura 1.6: Algoritmo genérico para o VNS.

Inicialmente, devemos encontrar para o problema uma solução inicial através de algum algoritmo guloso. Iniciamos também as vizinhanças a serem trabalhadas com $k = 1$. Após a inicialização, entramos em uma fase de repetição. Este laço se repete enquanto existirem vizinhanças a serem pesquisadas, ou algum outro critério de parada. Uma solução S' é gerada aleatoriamente a partir da vizinhança atual. $N_k(S)$. Um método de busca local é aplicado para se obter uma segunda solução S'', que deve ser um ótimo local. Então S'' é comparada com S'', e se S' for menor, então S será igual a S'', e a busca continua. Se não for menor, a vizinhança k é aumentada e o laço se repete.

Aplicações desta meta-heurística a problemas clássicos de otimização combinatória tais como: o problema do caixeiro viajante, problemas de localização (p-medianas), análise de *clusters* (particionamento), problemas com restrições bilineares, problemas de otimização global, dentre muitos outros foram desenvolvidas, [HAM98].

1.3.3 – Busca Tabu (BT)

A Busca Tabu (*Tabu-search*), proposta inicialmente por Glover (1989), é uma meta-heurística para resolução de problemas de otimização geral. Ela é baseada em princípios de técnicas inteligentes, [G89] e [GL96]. Segundo Reeves (1993), a BT é uma meta-heurística de melhoramento local que utiliza uma lista de movimentos proibidos para avançar em direção ao ótimo, e segundo Noronha et al (2000), podemos dizer que os métodos de busca tabu são técnicas que conduzem sub-métodos em direção a uma solução, [CRR93], [NDSA00].

A BT possui a habilidade de guiar o processo de busca dos métodos iterativos de melhoria, portanto fornecendo meios de explorar o espaço de soluções além dos pontos em que a heurística está guiando.

A BT parte de uma solução inicial e através de uma seqüência de movimentos caminha para outra solução. O conjunto de movimentos é chamado de lista de candidatos, e o método utilizado para comparar os movimentos é chamado de avaliador. A BT permite, através da lista de candidatos, que ao se encontrar um ótimo local, ela continue uma busca por ótimos locais melhores, e talvez até encontre um ótimo global.

Pode ocorrer que na busca de outras soluções, o movimento realizado chegue a uma solução já visitada. Para prevenir este ciclo, um número de atributos relacionados ao movimento são registrados numa lista, chamada lista tabu. Movimentos que revertam movimentos já realizados são chamados de tabu, e eles não serão mais aceitos por um certo número de iterações.

A BT seleciona o melhor movimento permitido, que não seja tabu. De posse destes atributos, é possível verificar se o movimento é tabu ou não, varrendo a lista tabu. Ela não permite que o algoritmo volte e re-visite uma solução pela qual já tenha passado, por *n* passos anteriores. Os atributos tabu são armazenados na lista por um pequeno número de iterações, e então são removidos. Atributos armazenados na lista restringem o espaço de busca, porém

uma diversificação é conseguida quando são considerados movimentos que não envolvem os atributos tabu, de forma a direcionar a busca para novas áreas de espaço de solução.

Esse procedimento de barramento de atributos e depois estrategicamente esquecê-los é chamado de memória de curto prazo. O conceito de memória de médio e longo prazo também é utilizado, e nela são armazenadas soluções completas e informações das outras vizinhanças. O uso destas memórias é para possibilitar estratégias de aprendizado as quais serão utilizadas para a realização de buscas mais minuciosas sobre uma região promissora, chamadas intensificações, e combinam movimentos e características históricas das soluções achadas. Ela analisa profundamente a vizinhança da resposta anterior. Elas também são utilizadas para guiar a busca para regiões inexploradas do espaço de soluções, chamado de diversificação.

Os critérios de aspiração são movimentos realizados entre os elementos e são utilizados de modo que movimentos interessantes que são tabus não sejam completamente rejeitados. A BT pode aceitar um movimento, mesmo se ele degradar o valor da função objetivo. Isto é que permite ao método de sair de ótimos locais. Os critérios de aspiração permitem aceitar certos movimentos tabu, que são julgados muito interessantes para o prosseguimento da pesquisa, deixando de ser tabu. O mais simples dos critérios de aspiração consiste em aceitar todo movimento em direção a uma solução que melhore o registro. Este critério é quase sempre adotado na maior parte das aplicações.

Um algoritmo para a BT pode ser colocado como na figura 1.7:

[01] **procedure** Busca-Tabu
[02] $\boldsymbol{k} = 1$;
[03] Gerar solução inicial $\boldsymbol{S}$
[04] **While** (critério de parada não for satisfeito) **do**
[05] **begin**
[06] Identifique $\mathbf{N}(\boldsymbol{S})$ // soluções vizinhas de $\boldsymbol{S}$
[07] Identifique $\mathbf{T}(\boldsymbol{S},\boldsymbol{k})$ // identifique a lista tabu **T**
[08] Identifique $\mathbf{A}(\boldsymbol{S},\boldsymbol{k})$ // verifique o critério de aspiração
[09] $\boldsymbol{S}_{\mathbf{best}}$ = (Selecione o melhor $\boldsymbol{S'} \in \mathbf{N}(\boldsymbol{S},\boldsymbol{k}) = \{\mathbf{N}(\boldsymbol{S}) / \mathbf{T}(\boldsymbol{S},\boldsymbol{k})\} \cup \mathbf{A}(\boldsymbol{S},\boldsymbol{k})$)
[10] Armazene *S*' se melhorar a melhor solução anterior $\boldsymbol{S} = \boldsymbol{S}_{\mathbf{best}}$
[11] $\boldsymbol{k} = \boldsymbol{k} + 1$
[12] **end-while**
[13] **end** { TS }

Figura 1.7: Algoritmo genérico para a Busca tabu.

Neste algoritmo, o melhor movimento admissível é aquele com melhor avaliação na vizinhança da solução corrente em termos de valor de função objetivo e restrições tabu. A função de avaliação escolhe o movimento que possui a melhor melhoria, ou a menor piora para a função objetivo. A lista tabu guarda informações dos movimentos realizados, para que os movimentos revistos sejam classificados como tabu. O critério de aspiração irá liberar alguns movimentos que são tabu, permitindo que se escape de ótimos locais. Como critério de parada poderemos ter: quantidade máxima de iterações, número de iterações sem melhorias, se a solução encontrada é uma solução ótima, ou se o número de vizinhanças acabou, ou ainda, se um determinado tempo limite foi atingido.

A literatura mostra diversas aplicações usando este método, tais como: escalonamento, roteamento de veículos, coloração de grafos, problema do máximo clique, problema da mochila, redes de telecomunicações, otimização global, redes neurais, etc, [GL96].

Descreveremos um exemplo de BT, onde teremos um vetor com números onde faremos permutações, e temos uma determinada solução inicial.

Permutações entre dois elementos aleatórios serão realizadas, gerando novas soluções, conforme pode ser visto na figura 1.8.

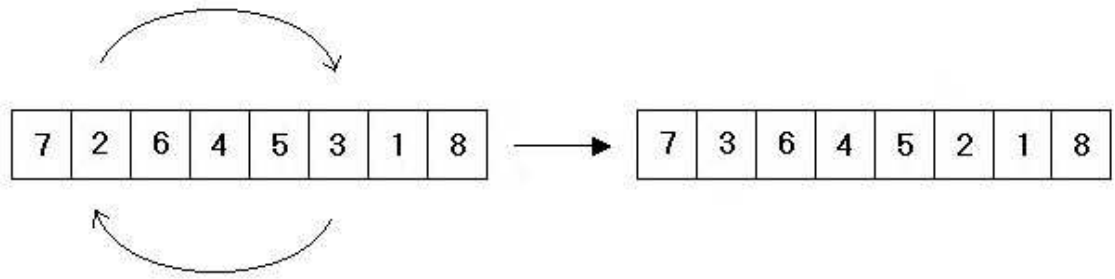


Figura 1.8: Troca entre dois elementos gerando uma nova solução.

Uma lista de candidatos será criada com movimentos que provocaram melhores resultados na solução. De acordo com a tabela 1.1, temos as permutações que geram os melhores ganhos na solução, que no caso será de maximização.

Tabela 1.1: Lista de candidatos.

Candidatos	Valor
2,6	7
5,8	4
3,7	3

Numa iteração da BT, o movimento do topo da lista de candidatos será escolhido para gerar uma nova solução. O movimento reverso será adicionado à lista tabu e será armazenado por apenas três iterações. Enquanto permanecer na lista tabu, o movimento será considerado proibido (critério de aspiração). A tabela 1.2 mostra a lista tabu e em cada célula pode-se verificar a quantidade de iterações que o movimento vai ficar proibido.

Tabela 1.2: Lista Tabu.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1								
2	2					3		
3							3	
4								
5		1						3
6								
7				2				
8								

O processo se repete, e um novo elemento da lista de candidatos será escolhido para gerar uma nova solução. Se melhorar a solução, então ela é atualizada. Caso a solução tenha um valor inferior, podemos permitir que alguns movimentos proibidos sejam utilizados, a fim de que saia de um ótimo local e explore novas regiões do espaço de soluções. Também é verificada a permanência dos movimentos da lista tabu. Se já duraram três iterações (critério da aspiração), então eles são retirados da lista, podendo ser utilizados novamente.

1.4 – Grafos

Um grafo simétrico **G** = (**V**,**A**) é definido pelo par de conjuntos **V** e **A**, onde:

V - conjunto não vazio: representa o conjunto dos vértices ou nós do grafo;

A - conjunto de pares (**v**,**w**), **v** e **w** ∈ **V**: as arestas do grafo.

Seja, por exemplo, o grafo **G** = (**V**,**A**) dado por:

V = { **p** | **p** é uma pessoa }

A = { (**v**,**w**) | < **v** é da familia de **w** > }

Esta definição de A representa pessoas da familia de outras pessoas. Um exemplo de elemento deste grafo, como pode ser visto na figura 1.9, é dado por:

V = { Maria, Pedro, Joana, Luiz }

A = { (Maria, Pedro) , (Joana, Maria) , (Pedro, Luiz) , (Joana, Pedro) }

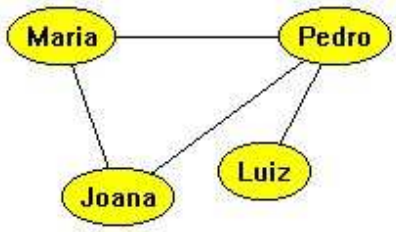


Figura 1.9: Exemplo de modelagem em um grafo.

Neste exemplo, estamos considerando que a relação <**v** é da familia de **w**> é uma relação simétrica, ou seja, se <**v** é da familia de **w**> então <**w** é da familia de **v**>. Como conseqüência, as arestas que ligam aos vértices não possuem qualquer orientação. Então, dizemos que este grafo é não orientado, ou seja, suas arestas não têm nenhuma orientação. Neste caso as arestas são chamadas de elos.

Considere, agora, o grafo definido por:

V = { **p** | **p** é uma pessoa da família Castro }

A = { (**v**,**w**) | < **v** é pai/mãe de **w** > }

Um exemplo deste grafo pode ser visto na figura 1.10, e é dado por:

V = { Emerson, Isadora, Renata, Antonio, Rosane, Cecília, Alfredo }

A = {(Isadora, Emerson), (Antonio, Renata), (Alfredo, Emerson), (Cecília, Antonio), (Alfredo, Antonio)}

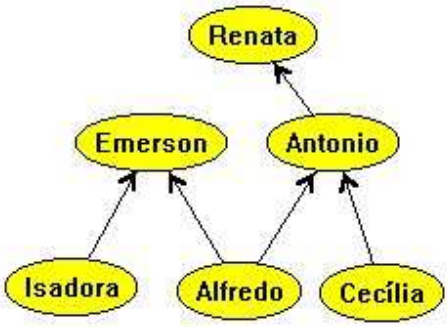


Figura 1.10: Exemplo de modelagem em um grafo orientado.

A relação definida por **A** não é simétrica, pois se <**v** é pai/mãe de **w**>, não é o caso de <**w** é pai/mãe de **v**>. Há, portanto, uma orientação na relação, com um correspondente efeito na representação gráfica de **G**.

O grafo acima é dito ser um grafo orientado (ou dígrafo) e acíclico, sendo que as conexões entre os vértices são chamadas de arcos.

Temos um outro tipo de grafo, chamado de grafo misto, que possui tanto elos quanto arcos nas suas ligações (arestas). Um exemplo de grafo misto pode ser visto na figura 1.11.

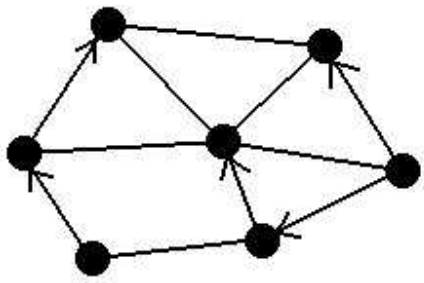


Figura 1.11: Exemplo de grafo misto.

Um grafo **G** = (**V**,**A**) é dito valorado, quando são atribuídos pesos às suas ligações e/ou a seus vértices.

Um grafo **G** = (**V**,**A**) é dito ser um multigrafo quando existem múltiplas arestas entre pares de vértices de **G**. No grafo da figura 1.12, por exemplo, há duas arestas entre os vértices **A** e **C** e entre os vértices **A** e **B**, caracterizando-o como um multigrafo.

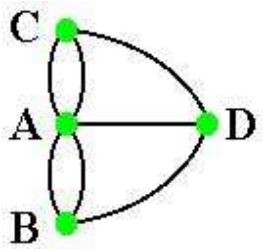


Figura 1.12: Exemplo de multigrafo.

1.5 – Problemas de Caminho Mínimo

Problemas de caminho mínimo são fundamentais e muito comumente encontrados em problemas que estudam aplicações no setor de transportes e redes de comunicação.

Existem diversos tipos de problemas de caminho mínimo. Por exemplo, nós podemos estar interessados em encontrar o menor caminho (ou mais econômico, ou mais rápido) a partir de um determinado vértice de um grafo valorado até outro, ou de um nó a todos os outros, ou até o menor caminho entre todos os pares de nós da rede (grafo). Podemos também querer o caminho mínimo passando por determinados vértices do grafo.

1.5.1 – O Algoritmo PDM

O PDM é um algoritmo exato para o cálculo do caminho mínimo em grafos orientados. Ele foi proposto inicialmente por Moore em 1957 e Bellman em 1958, e foi implementado de maneira eficiente por vários pesquisadores. A este método foi aplicada uma heurística proposta por d' Esopo e refinada por Pape em 1980. Esta heurística deixa o algoritmo mais rápido, ou de complexidade média mais baixa. Uma implementação do PDM se encontra em [SDK83].

Ao contrário do algoritmo de Dijkstra, o PDM funciona corretamente se a rede contiver elementos com peso negativo, desde que não forme circuitos negativos. Além disso, ele não somente calcula o caminho mínimo de um nó a outro da rede, e sim de um nó para todos os outros. Um esquema para o PDM se encontra na figura 1.13. Maiores detalhes sobre esta implementação, como o algoritmo passo a passo, encontram-se em [SDK83].

```
[01] procedure PDM
[02] // Inicialização - Q é uma fila
[03] for todos v ∈ V do
[04]   begin
[05]     dist(v) ← ∞
[06]     pred(v) ← -1
[07]   end;
[08] dist(s) ← 0;
[09] // Inicialize Q para conter s somente;
[10] head ← s;
[11] // Iteração
[12] while Q não estiver vazio do
[13] begin
[14]     retire o nó u de Q
[15]     para cada aresta (u,v) que começa em u faça
[16]     begin
[17]             newlabel ← dist(u) + w_u,v
[18]             if newlabel < dist(v) then
[19]             begin
[20]                     dist(v) ← newlabel
[21]                     pred(v) ← u;
[22]                     if v nunca esteve em Q then
[23]                             insira v no fim de Q
[24]                     else if v estava em Q mas não atualmente em Q then
[25]                             insira v no inicio de Q
[26]             end
[27]     end
[28] end { PDM }
```

Figura 1.13: Algoritmo genérico para o PDM.

O algoritmo acima procura os menores caminhos a partir do nó **s** para todos os outros nós do grafo. O vetor **dist** armazena a menor distância obtida até o momento do nó de origem para cada nó da rede, e **pred** armazena o nó predecessor de cada vértice do grafo. Durante a

inicialização, é armazenado o valor zero em **dist** na posição correspondente ao nó de origem, e um número muito alto nas demais. No passo iterativo, **dist** será sempre atualizado com a distância atual conhecida de **s** ao nó que estiver sendo examinado, e **pred** também será atualizado com o predecessor deste nó. A fila **Q**, que a cada iteração pode ter novos elementos adicionados, possui os vértices das arestas que se conectam aos nós examinados. Um nó será examinado se ele estiver na cabeça da fila. A cada iteração, o nó que estiver na cabeça da fila terá suas arestas identificadas e nós adjacentes entrarão na fila, e em seguida será apagado da fila. Certos nós já examinados poderão entrar novamente na fila, e neste caso, eles não entrarão no final, e sim no início, para evitar iterações desnecessárias. Isto ocorre se na análise do arco (**u**,**v**) se um caminho melhor que o corrente for encontrado, ou seja, se **dist** for reduzido.

Um exemplo gráfico de solução para o PDM pode ser visto na figura 1.14 e tabela 1.3. Note que o grafo possui uma aresta com peso negativo.

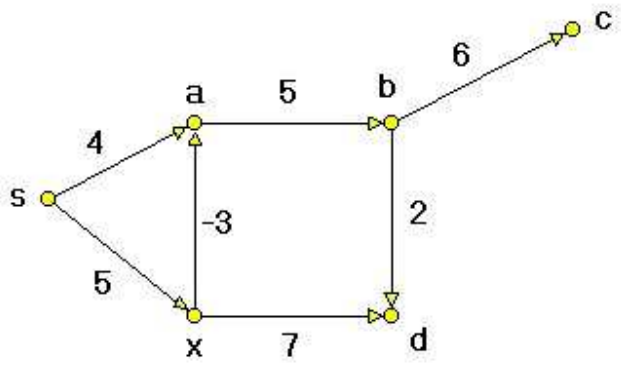


Figura 1.14: Grafo exemplo para o PDM.

Tabela 1.3: Descrição da movimentação na fila e os custos a cada iteração para solução do PDM.

Q ←	**s**	**a**	**b**	**c**	**d**	**X**
S	0	∞	∞	∞	∞	∞
A , X	0	4	∞	∞	∞	5
X , B	0	4	9	∞	∞	5
A , B , D	0	2	9	∞	12	5
B , D	0	2	7	∞	12	5

D , C	0	2	7	13	9	5
C	0	2	7	13	9	5
	0	2	7	13	9	5

A implementação do PDM utiliza como entrada a quantidade de nós e arestas da rede, e a lista de ligações entre pares de vértices e seus respectivos custos associados. É armazenado com uma estrutura *forward-star*. Ele tem como saída o caminho mínimo e o custo de um nó específico para todos os outros nós da rede. Sua complexidade pode ser de $O(n^2)$, se poucos vértices entrarem novamente na fila, ou $O(2^n)$ no pior caso, [SDK83].

Se a rede é grande (com muitos vértices) e esparsa, e precisa-se calcular o caminho mínimo de um nó para todos os outros, é preferível usar o PDM. Mas se a rede é pequena e densa, sugere-se o uso de outro procedimento o algoritmo de Dijkstra, estas observações foram consideradas em [SDK83].

1.6 – O Problema do Caixeiro Viajante - PCV

O Problema do Caixeiro Viajante (*Traveling Salesman Problem*) é um dos mais antigos problemas de otimização. Problemas matemáticos relatados sobre o PCV foram tratados no século XIV pelo matemático irlandês Sir William Rowan Hamilton e pelo matemático inglês Thomas Penyngton Kirkman. Hamilton inventou o jogo icosiano em 1857, e o vendeu para um comerciante em Londres, por 25 *pounds*. Depois dessa venda, o jogo foi comercializado em toda a Europa, sob diferentes formas. A figura 1.15 exibe o dodecaedro utilizado por Hamilton.

Figura 1.15: Dodecaedro pentagonal utilizado por Hamilton.

O jogo icosiano é um quebra-cabeça que consiste basicamente em passar por todos os vértices apenas uma vez. Hoje em dia, esses caminhos são conhecidos como Circuitos Hamiltonianos. Um circuito Hamiltoniano sobre um tablado do jogo icosiano original pode ser visto na figura 1.16.

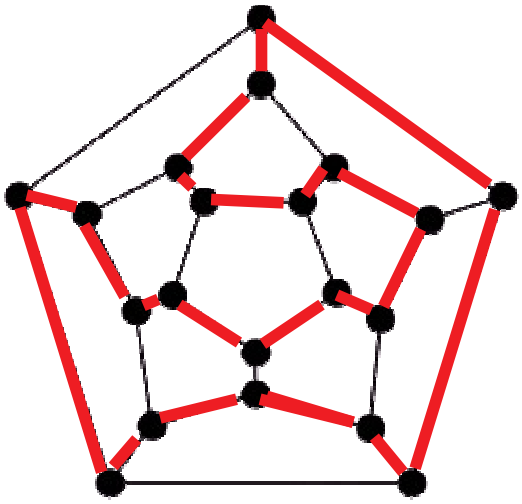


Figura 1.16: Um circuito Hamiltoniano sobre o tablado icosiano.

Antes mesmo dessa venda, em 1856, Hamilton já havia criado o Cálculo Icosiano, no qual ele usou para investigar caminhos em um dodecaedro, para passar por cada vértice uma única vez. Atualmente, existem diversas aplicações baseadas no jogo icosiano, como o Circuito Hamiltoniano e também o Problema do Caixeiro Viajante, estudado em teoria dos grafos.

A forma geral para o PCV foi estudada inicialmente por volta de 1930 por Karl Menger em Vienna e em Harvard. O problema foi estudado depois por Hassler Whitney e Merril Flood em Princeton. Um tratamento detalhado dos trabalhos de Menger e Whitney, e do crescimento do PCV como tópico de estudo pode ser encontrado num artigo publicado por Alexander Schrijver (2003), [SC03].

No Problema do Caixeiro Viajante, dado uma lista de cidades que um caixeiro precisa visitar, o caixeiro deve visitá-las exatamente uma vez, e então retornar a cidade de partida. Dado um custo entre cada par de cidades, como planejar o itinerário de forma que cada cidade seja visitada somente uma vez, e que o custo total do circuito seja o mínimo possível? Em outras palavras, este problema quer encontrar um circuito de custo mínimo num grafo completo (onde todo vértice está ligado a todo vértice) e valorado nas ligações.

Seja $\mathbf{G} = (\mathbf{V},\mathbf{L})$ um grafo onde **V** é o conjunto de vértices, e **L** é o conjunto de ligações, compostas por elos e arcos. Seja $\mathbf{C} = [\mathbf{c}_{ij}]_{n\times n}$ a matriz de custos associada a **L**. O PCV consiste em determinar o circuito de distância mínima passando por cada vértice somente uma vez. Este circuito é conhecido como ciclo ou circuito Hamiltoniano, desde que se considere **C** uma matriz completa. Um exemplo gráfico para o PCV pode ser visto na figura 1.17.

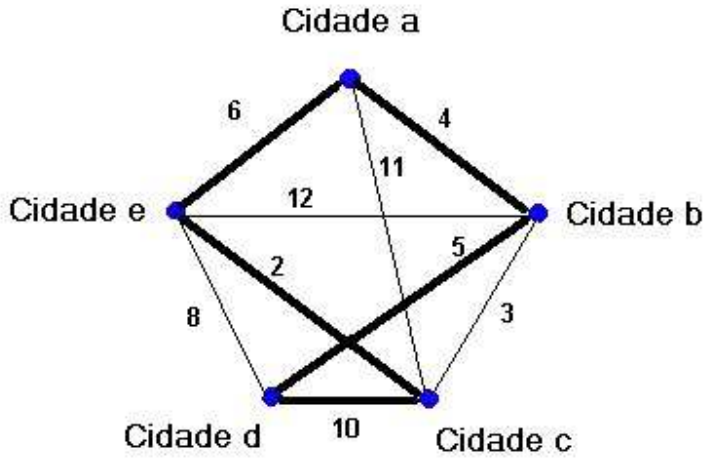


Figura 1.17: Exemplo de solução para o PCV.

Em algumas aplicações, **C** pode ser admitida como uma matriz de custos, distâncias ou tempos de percurso. É importante, porém, fazer a distinção entre os casos em que **C** é simétrica, ou seja, $\mathbf{c}_{ij} = \mathbf{c}_{ji}$ para todo $\boldsymbol{i}, \boldsymbol{j} \in \mathbf{V}$, ou assimétrica. Também se diz que **C** satisfaz à desigualdade triangular se e somente se $\mathbf{c}_{ij} + \mathbf{c}_{jk} \geq \mathbf{c}_{ik}$, para todo $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k} \in \mathbf{V}$, em geral nos problemas Euclidianos com grafos completos.

Existem diversas formulações para o PCV, que podem ser baseadas em atribuição, em 1-fluxo, fluxo com múltiplas comodidades, e outras.

Dentre as formulações existentes para o PCV, uma formulação baseada em fluxo introduzida por Gavish e Gravis (1978) pode ser colocada como segue, [GG78]:

$$[PCV] \quad Min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} . x_{ij}$$

$$[1] \quad \sum_{i=1}^{n} x_{ij} = 1 \qquad (j = 1,2,...,n)$$

$$[2] \quad \sum_{j=1}^{n} x_{ij} = 1 \qquad (i = 1,2,...,n)$$

$$[3] \quad \sum_{j=1}^{n} y_{ji} - \sum_{j=1}^{n} y_{ij} = -1 \qquad (i = 1,2,...,n)$$

$$[4] \quad y_{ij} \leq U x_{ij} \qquad (i = 1,2,...,n)$$

$$[5] \quad x_{ij} \in \{0,1\}, y_{ij} \geq 0 \qquad (i = 1,2,...,n)$$

Eles introduziram variáveis de fluxo $\mathbf{y_{ij}}$, e assumiram que (n-1) unidades de fluxo são supridas pela origem, o nó 1, e que todos os outros vértices da rede necessitam de apenas uma unidade de fluxo. Os dois conjuntos de restrições, [1] e [2] são de atribuição. As restrições [3] evita uma formação de sub-ciclos e as desigualdades [4] são restrições para forçar o custo, e faz com que o fluxo no arco $\mathbf{y_{ij}}$ sobre o arco (***i,j***) seja zero se ele não for selecionado na matriz de atribuição **X**. Esta formulação proíbe a formação de sub-ciclos, uma vez que o se o vértice ***i*** recair sobre um sub-ciclo que não contenha a origem, ele não poderia receber uma unidade de demanda da origem.

O PCV pode ser implementado de maneira exata ou heurística. De maneira exata, podemos aplicar métodos *branch-and-bound*, e para soluções heurísticas, podemos aplicar heurísticas de construção, melhorias, composição de rotas, meta-heurísticas, [NEG96], ou algoritmos aproximativos.

O PCV possui variações quando tratamos do grafo base. Dentre elas temos por exemplo quando o grafo a ser trabalhado for assimétrico, ou seja, um grafo direcionado cujos custos dos arcos (***i,j***) e (***j,i***) são distintos para algum par **i** e **j**, sendo chamado de Problema do Caixeiro Viajante Assimétrico. Pode-se trabalhar também com um grafo que não é totalmente completo, isto é, existem pares de nós que não estão diretamente conectados por arestas. Um grafo completo sempre terá solução, mas num grafo não completo nem sempre haverá solução hamiltoniana. Outro problema é encontrar se um dado grafo com n vértices possui ou não um ciclo Hamiltoniano de tamanho ***n***. Na versão simétrica, também é possível que não haja a

formação de ciclos hamiltonianos, porém na versão euclidiana sempre teremos um ciclo hamiltoniano de custo mínimo.

Variações do PCV existem, por exemplo, se o caixeiro não desejar voltar a origem. Nesse caso, teremos outro problema, que é encontrar um caminho de tamanho (***n***-1) e não um circuito de tamanho ***n***, denominado Problema do Caminho Hamiltoniano, [SDK83].

O PCV tem aplicação em várias áreas, tais como: roteamento de veículos, conexão de computadores, corte de papel, vidro e placas de metal, seqüenciamento de trabalhos, desenho de placas. Vários trabalhos de aplicações que utilizam o PCV foram publicados, como: perfuração de placas de circuitos impressos, análise da estrutura de cristais, manejo de materiais em armazéns, agrupamento de dados, seqüência de trabalhos numa máquina simples, [BS87], [RR81], [LLR85], [GG64].

1.6.1 – O Método Branch & Bound

Algoritmos *branch-and-bound* são baseados numa árvore de busca onde em cada passo todas as possíveis soluções de uma instância de problema são particionadas em dois ou mais subgrupos, cada uma representada pela fixação de um nó de uma árvore de decisão.

O particionamento é executado de acordo com alguma heurística, a qual reduz a quantidade de buscas a serem conduzidas. Após o particionamento, limites são computados para cada um dos subgrupos. O próximo espaço de solução a ser investigado é escolhido com o menor custo dos subgrupos pesquisados.

Por exemplo, para o PCV, cada solução será particionada em dois subgrupos: aqueles que contêm a aresta (*i*,*j*) e aqueles que não a contem ($\mathbf{x_{ij}} = 1$ e $\mathbf{x_{ij}} = 0$). O processo continua até que um ciclo Hamiltoniano seja obtido.

A implementação do PCV com *branch-and-bound* tem como entrada a quantidade de nós da rede, e uma matriz de custos de tamanho $n \times n$. Como saída teremos uma rota ótima a partir do primeiro nó.

O *branch-and-bound* é um método exato de busca. No pior caso, ele pode terminar explorando todas as possíveis soluções. Para um grafo assimétrico com ***n*** vértices, existem O(***n***-1)! ciclos hamiltonianos distintos entre todas as soluções possíveis de serem exploradas.

A complexidade de tempo do pior caso deste algoritmo pode ser ruim como O(***n*!**). Porém, estes métodos exploratórios são de ordem bem menor na maioria dos casos, ou seja, são $O(2^n)$. Em geral, é muito custoso obter uma solução exata para o PCV para redes com quantidade de nós maiores que 40, considerando os algoritmos B&B atuais, de ordem $O(2^{kn})$, $1<k<2$.

Um algoritmo genérico B&B pode ser descrito como na figura 1.18:

```
[01] procedure B&B
[02] z ← ∞; // Limite Inferior
[03] custo ← 0
[04] S1 ← a1;// Primeiro elemento da Solução (Origem)
[05] k ← 2;
[06] while k > 0 do
[07] begin
[08]    while (Sk ≠ ∅)  e (custo < z) do
[09]          begin
[10]             ak ← um elemento em Sk; // Branching
[11]             Sk ← Sk -{ ak }; // Uma tomada de decisão possível a partir de ak
[12]             custo ← custo (a1, a2, ..., ak);
[13]             // O limite inferior foi atingido - Bounding
[14]             if (a1, a2, ..., ak) não é uma solução e custo (a1, a2, ..., ak)>z Then Break;
[15]             if (a1, a2, ..., ak) é uma solução e (custo < z) then
[16]               begin
[17]                  salve(a1, a2, ..., ak) como a menor solução;
[18]                  z ← custom; // limite inferior
[19]               end-if;
[20]             k ← k + 1;
[21]             compute Sk;
[22]         end-while;
[23]    k ← k – 1; // Backtrack
[24]    custo ← custo(a1, a2, ..., ak);
[25]  end-while;
[26] end. { B&B }
```

Figura 1.18: Algoritmo genérico para o *branch-and-bound*.

A seguir mostramos passo a passo um exemplo de procedimento *branch-and-bound* para se encontrar um caminho entre dois pontos para o grafo da figura 1.19. Temos como objetivo encontrar um percurso mínimo do vértice origem **A** ao vértice destino **F**. Colocamos na raiz da árvore o nó origem, e expandimos a raiz obtendo como resultado novos nós que

correspondem aos nós de ligação ao nó origem. Calculamos o custo total da origem a cada nó e ordenamos do menor para o maior por custo. A cada iteração repetiremos este processo para todos os nós até encontrar o nó destino **F**. Então no final poderemos computar qual foi o menor caminho que parte de **A** até **F**, dentre os caminhos obtidos. Mostraremos algumas iterações que chegam a um caminho, sendo que os caminhos estão acompanhados do custo do nó origem até o percurso associado. Na figura 1.20 mostraremos uma árvore exibindo o caminho percorrido para se encontrar a primeira solução.

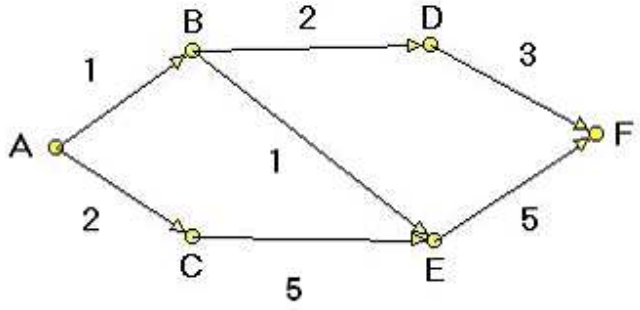


Figura 1.19: Grafo exemplo para o *branch-and-bound*.

K=1;

$z=\infty$; S_1= {A}; (a_1)=(A); custo=0;

K=2;

$z=\infty$; S_2= {B,C}; (a_1, a_2)=(A, B); custo=1; (Não é Solução)

K=3;

$z=\infty$; S_3= {D,E}; (a_1, a_2, a_3)=(A, B, D); custo=3; (Não é Solução)

K=4;

$z=\infty$; S_4= {F}; (a_1, a_2, a_3, a_4)=(A, B, D, F); custo=6; (É Solução)

custo<*z então z=custo*; S*={A,B,D,F};

K=5;

z=6; S_5= {}; Backtrack;

K=4;

$z=6$; S_4= {}; Backtrack;

K=3;

z=6; S_3= { D,E }; (a_1, a_2, a_3, a_4)=(A, B, E); custo=2; (não é Solução)

K=4;

z=6; S_4= { F }; (a_1, a_2, a_3, a_4)=(A, B, E, F); custo=7; (é Solução)

K=5;

z=6; S_5= { }; Backtrack;

K=4;

z=6; S_4= { }; Backtrack;

K=3;

z=6; S_3= {}; Backtrack;

K=2;

z=6; S_2= {B,C}; (a_1, a_2)=(A, C); custo=2; (Não é Solução)

K=3;

z=6; S_3= {E}; (a_1, a_2, a_3)=(A, C, E); custo=7; (Não é Solução)

custo>z então Backtrack;

K=2;

z=6; S_2= {}; Backtrack;

K=1;

z=6; S_1= {}; Backtrack (Fim)

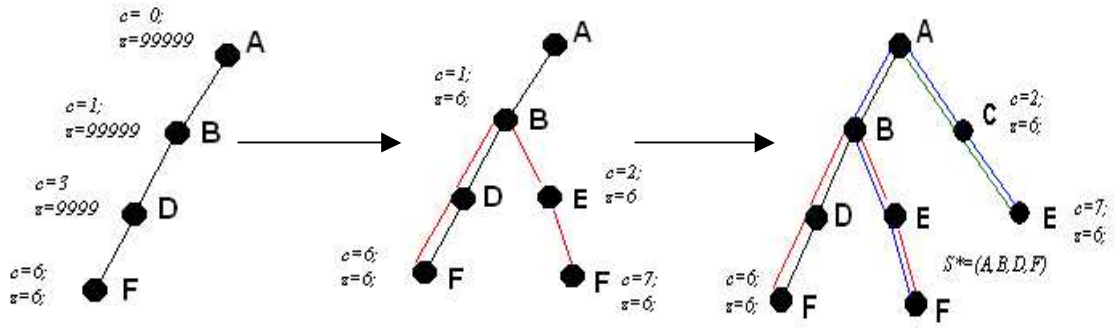


Figura 1.20: Árvore gerada a partir das iterações do *branch-and-bound*.

1.7 – O Problema do Caixeiro Viajante Generalizado - GTSP

O Problema do Caixeiro Viajante Generalizado (GTSP, *Generalized Traveling Salesman Problem*) é uma das variações do PCV. Neste problema, o caixeiro deve passar por um número pré-definido de grupos de clientes, visitando pelo menos um cliente de cada grupo, enquanto minimiza a soma dos custos da viagem. Em outras palavras, o GTSP consiste em determinar o Ciclo Hamiltoniano de menor custo, passando exatamente uma vez por cada grupo de vértices. Para determinar a ordem em que os grupos devem ser visitados, o caixeiro pode escolher o cliente ou os clientes a serem visitados em cada grupo.

O GTSP pode ser definido num grafo direcionado, simétrico, ou misto. Há versões do GTSP onde mais de um elemento por grupo pode ser visitado, porém a maioria das versões em estudo sobre este problema consideram que o caixeiro visita somente um cliente de cada grupo, [SDK83].

O GTSP foi introduzido simultaneamente por Srivastava et al (1969), que trabalharam com a versão simétrica do problema e propuseram uma aproximação através de programação dinâmica, e por Henry-Labordere (1969), que trabalharam com a versão assimétrica do problema, [SKGP69], [HEN69]. Saksena (1970) trabalhou com ambas as versões e apresentou uma aplicação na área de escalonamento, [SAK70]. Anos depois, Laporte e Nobert (1983) também trabalharam com as duas versões do problema, utilizando programação inteira e *branch-and-bound* para a sua solução, [LANO83].

Mais recentemente, o problema foi assunto de duas teses de doutorado, de Noon (1988) e Sepehri (1991), que apresentaram uma solução ótima para o GTSP assimétrico, onde os grupos de clientes são mutuamente exclusivos, [NOON88], [SEP91]. Em 1993, Noon and Bean apresentaram uma transformação do GTSP assimétrico num problema de caixeiro clusterizado equivalente, e também mostraram como construir para qualquer GTSP um PCV assimétrico equivalente, cuja solução ótima para o GTSP pode ser obtida a partir da solução ótima do PCV, [NB93].

Finalmente, Fischetti (1994) propôs um algoritmo de *branch-and-bound* para obter a solução ótima do GTSP simétrico, e mostrou o GTSP como um problema NP-Hard, [FIS94]. Jacques Renaud e Fayez Boctor (1996) propuseram heurísticas para a resolução do GTSP, [RB96]. Laporte trabalhou com o GTSP para encontrar uma solução aproximada para o PCRM, onde ele calculava o GTSP e o transformava em um PCV, [LAP97].

O GTSP possui várias aplicações potenciais. Laporte modelou uma variedade de problemas de otimização através do GTSP, são elas: problemas de localização, roteamento, sistemas de fluxo, *design*, e vários outros, [LAS95].

Dado um grafo **G** = (**V**, **A**), um conjunto $\boldsymbol{K_n}$ de grupos de vértices e $\boldsymbol{C_n}$ o conjunto de vértices selecionados. Um algoritmo genérico do tipo Monte Carlo para o GTSP pode ser colocado como na figura 1.21.

```
[01] procedure GTSP
[02] // Inicialização
[03]  Identificar em G os elementos de K_n
[04] //  Execução do algoritmo
[05]  for cada K_n do
[06]    Selecionar aleatoriamente um v ∈ V e v ∈ K_n
[07]    Armazene em C_n
[08]    Calcular o PCV entre cada v ∈ C_n
[09]  end-for
[10] end { GTSP }
```

Figura 1.21: Algoritmo genérico Monte Carlo para o GTSP.

Uma versão melhorada deste algoritmo pode ser colocada com a aplicação de uma heurística de tentativa de melhoria. Inicialmente, um vértice de cada grupo de arcos requeridos é selecionado, e então é gerado o percurso. Nesta heurística de melhoria, um melhor vértice para a conexão entre os grafos é procurado nos vértices de cada grupo. Ela inicialmente seleciona um vértice de cada grupo e calcula o GTSP. Então ela armazena a solução e troca por todos os vértices do mesmo grupo, um de cada vez, calculando a solução a cada troca. Assim, obtém-se a melhor solução para um vértice do grupo. O procedimento se repete para cada grupo existente, tendo no final uma boa solução para o GTSP.

A figura 1.22 mostra um pseudo-algoritmo para a melhoria do GTSP.

```
[01] procedure Heurística_GTSP;
[02] begin
[03] // (Passo – 1: Inicialização)
[04]    Selecione v_p para cada g_n ∈ G(x) e armazene em V(x).
[05]    Calcule o GTSP entre V(x) e armazene o custo de S.
[06] // (Passo – 2: Procedimento de melhoria)
[07]    For cada grupo g_n ∈ G(x) , n = 1, 2, ..., no. de grupos, do:
[08]        For cada vértice v_m ∈ G(x) , m = 1, 2, ..., no. de vértices, do:
[09]              Selecione um v_m para g_n
[10]            Insira v_m na posição n de V(x).
[11]            Calcule o PCV com V(x) e armazene o custo de S'
[12]            If custo(S' ) < custo(S ) Then S ← S'
[13] end. { Heurística_GTSP }
```

Figura 1.22: Algoritmo de melhoria para o GTSP.

Nós podemos verificar graficamente como o GTSP funciona, de um modo genérico, nas figuras abaixo. A figura 1.23 mostra um grafo inicial, onde iremos identificar nele grupos de arestas de interesse.

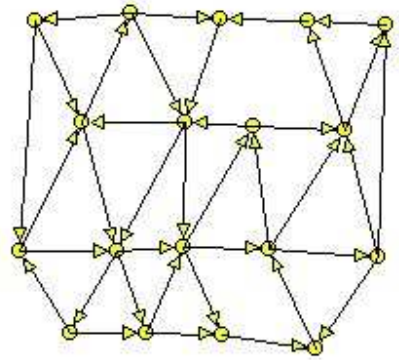


Figura 1.23: Grafo original.

Na figura 1.24 podemos visualizar os grupos identificados e os seus vértices. Para a solução do GTSP, um vértice de cada grupo será selecionado, e gera-se um PCV entre estes vértices, fechando um circuito que passa em cada grupo, com se pode verificar na figura 1.25. Assim, pelo menos um vértice de cada grupo será visitado.

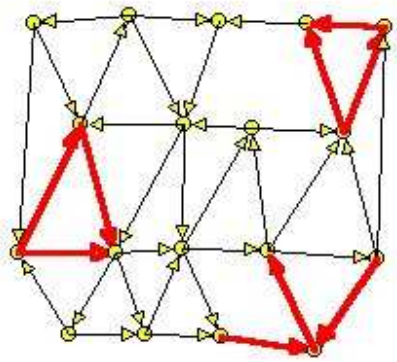


Figura 1.24: Grafo com grupos de arestas identificados.

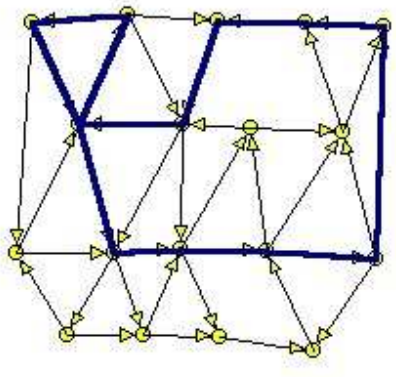


Figura 1.25: Percurso gerado entre os grupos.

1.8 – Relações de Equivalência

Uma relação de equivalência ***R*** é definida sobre um grupo ***S*** se para cada par de elementos (*a*,*b*), *a* e *b* ∈ ***S***, *a* ***R*** *b* é falso ou verdadeiro. Se ***R*** é verdadeiro, podemos dizer que *a* está relacionado a *b*.

Uma relação de equivalência é uma relação ***R*** que satisfaz as três seguintes propriedades:

Reflexiva : *a* ***R*** *a*, para todo *a* ∈ ***S;***

Simétrica : *a* ***R*** *b*, se e somente se *b* ***R*** *a* e

Transitiva : *a* ***R*** *b* e *b* ***R*** *c*, implica em *a* ***R*** *c*.

Entre *a* e *b* existem duas possíveis operações. A primeira é achar quem retorna o nome do grupo, ou seja, a classe de equivalência. A segunda operação adiciona relações. Se nós queremos adicionar a relação *a* ***R*** *b*, devemos verificar se *a* e *b* já estão relacionados. Isto é feito realizando buscas em *a* e *b*, e verificando quando eles estiverem na mesma classe de equivalência. Se eles não estiverem, então aplicaremos a operação de união. Esta operação junta as duas classes de equivalência, contendo *a* e *b* numa nova classe de equivalência.

Estas operações são importantes em muitos problemas de teoria dos grafos e de compiladores.

As relações de equivalência são utilizadas neste trabalho para se descobrir grupos de elementos requeridos. Cada aresta requerida foi considerada como um elo, e verificado então a

ligação de cada par de arestas do grupo de requeridos. Se elas tinham ligação direta, então pertenciam a mesma classe de equivalência. Se possuírem algum tipo de ligação indireta, também pertenciam a mesma classe. Cada classe de equivalência encontrada ao final do processo (após todas as arestas requeridas terem sido visitadas) corresponde a um vértice de um grupo de arestas interligadas.

Vários algoritmos para relações de equivalência se encontram em Weiss (1992). Dois deles são apresentados nas figura 1.26 e 1.27, [WM92].

```
[1] Procedure Find(x, S)
[2] If S[x] ≤ 0 then find = x
[3] else
[4]    begin
[5]       find := find(S[x], S);
[6]       find := S[x];
[7]    end-ifelse
[8] end { FIND }
```

Figura 1.26: Algoritmo genérico para o encontrar equivalências.

```
[1] Procedure Union (S, raiz1, raiz2) {união de classes pela altura}
[2] If S[raiz2] <  S[raiz1] then S[raiz1] = raiz2
[3] else
[4]    begin
[5]       If S[raiz2] = S[raiz1] then {Alturas iguais, atualizar }
[6]          S[raiz1] := S[raiz1]-1;
[7]       S[raiz2] : =  raiz1;  { fazer a raiz 1 ser a nova raiz }
[8]    end-ifelse
[9] end { UNION }
```

Figura 1.27: Algoritmo genérico para o unir equivalências numa mesma classe.

O vetor ***S*** é a solução do problema, onde inicialmente está limpo, e conterá no final quem é o pai do elemento correspondente ao índice do vetor.

O algoritmo **union** procura descobrir quem é o pai de um elemento durante uma iteração da busca, ou seja, ele descobre quem está diretamente ligado ao elemento. Ele atualiza a estrutura de dados utilizada com a informação de conexão entre os elementos.

Um exemplo gráfico do comportamento algoritmo **union** pode ser visualizado na figura 1.28. Nele temos um conjunto formado pelos números de 0 a 9, e **L** as ligações entre estes elementos, **L** = {(3,4), (4,9), (8,0), (2,3), (5,6), (5,9), (7,3)}, e à medida que estas ligações são identificadas, grupos de elementos são formados a cada iteração, obtendo no final os grupos 1, 9 e 0.

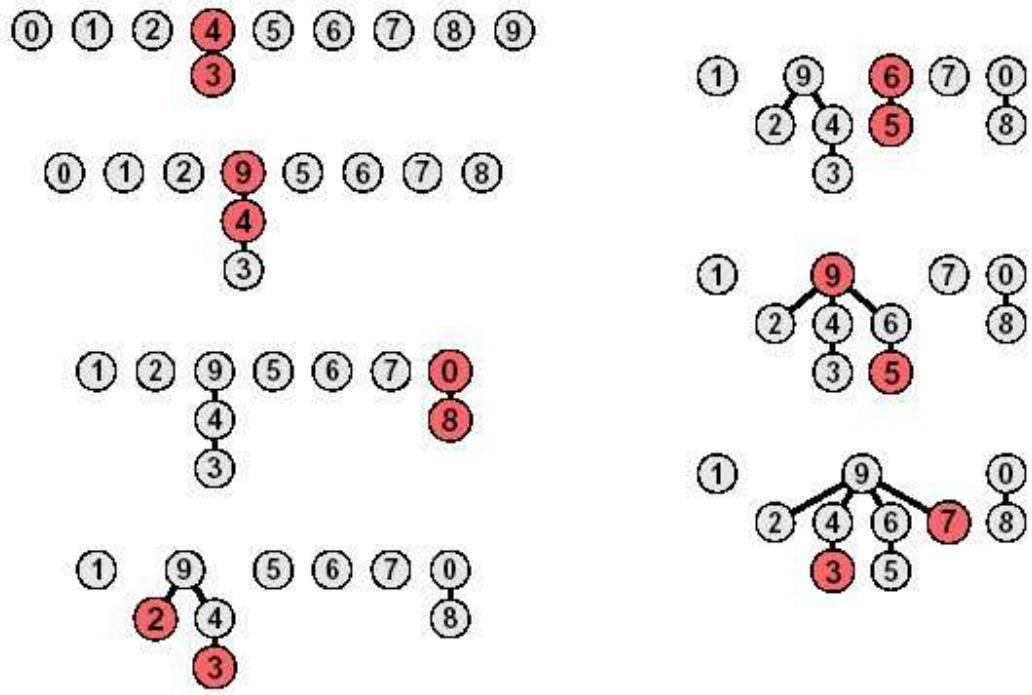


Figura 1.28: Aplicação do algoritmo **union**.

A função **find** procura a classe de equivalência do elemento ***x***, e caso ele ainda não a tenha, preenche o vetor ***S***. Caso contrário, é feita uma chamada recursiva de **find** no elemento que se encontra na solução ***S***. Este algoritmo retorna quem é o pai de um grupo de elementos, ou seja, retorna quem é a raiz do grupo. Ele encontra o pai final do elemento, ou seja, o nó raiz, e obtendo no final uma estrutura onde as raízes serão os grupos obtidos.

Como exemplo, supondo o vetor **V** = (1, 2, 3, 4, 5, 6, 7, 8), desejamos encontrar as classes de equivalências a partir de um conjunto **C**. O vetor será preenchido com as respostas das funções **find** e **union**, como abaixo descreveremos.

Dado o conjunto **C** = { (5,6) , (7,8) , (5,7) }, que é o conjunto de ligações entre i , $j \in$ **V**. Inicialmente aplicaremos **union** ao primeiro elemento do conjunto C, que é (5,6). Obteremmos que 5 é o pai de 6, exibido na figura 1.29:

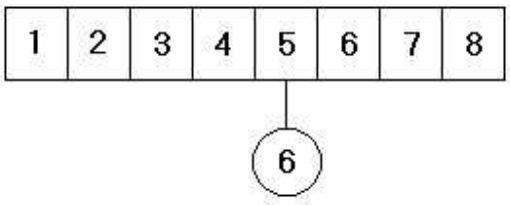


Figura 1.29: Primeira iteração – descoberta do pai de 6.

Na próxima iteração, aplicaremos **union** para a (7,8), e verificaremos que 7 é o pai de 8. Com a aplicação ao último elemento de **C**, (5,7), obteremos que o 5 é o pai de 7.

A partir deste ponto já temos alguns relacionamentos entre os componentes, mas eles não estão completamente relacionados, pois o 5 é pai do 6 e do 7, e o 7 é pai do 8, e conseqüentemente, o 5 seria o nó raiz e pai do 6, 7 e 8.

O algoritmo **find** seria aplicado então para a obtenção dos pais de cada elemento, e preenchimento do vetor ***S***. Aplicaríamos inicialmente em 1, e como ele não tem um pai, colocaríamos em ***S*** o valor –1 na sua posição correspondente, só para indicar que ele não tem pai. O mesmo ocorre se aplicarmos o **find** para 2, 3 e 4. Ao aplicarmos em 5, obteremos que ele mesmo é seu pai. Ao aplicarmos em 6, obteremos que o 5 é seu pai também. Ao aplicarmos em 7, também obteremos o 5 como pai. Ao aplicarmos em 8, teríamos uma chamada recursiva da função, pois inicialmente teríamos 7 como pai, mas como ***S***[7] já está preenchido, aplicaríamos **find**(***S***[7],***S***), e obteríamos 5 como pai. Ao final teremos ***S*** = (0, 0, 0, 0, 5, 5, 5, 5), como pode ser visto na figura 1.30.

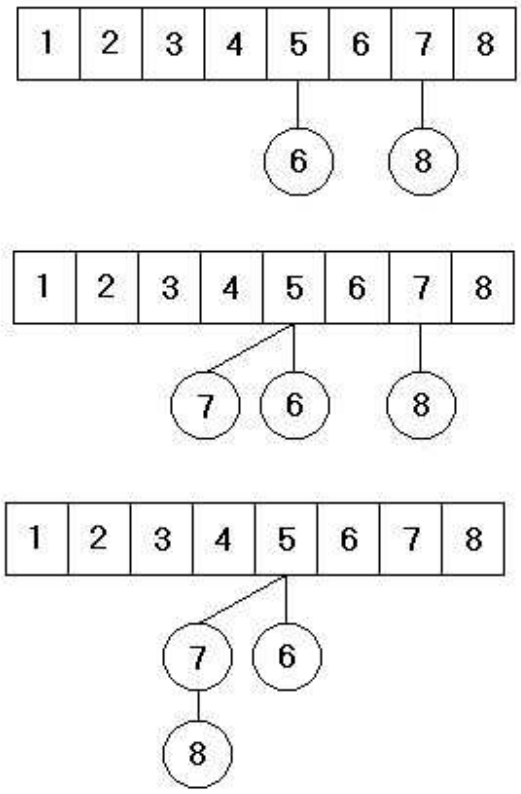


Figura 1.30: Aplicação do algoritmo **union** e **find**.

1.9 – Problemas de Fluxo em Redes

Os problemas de fluxo abordam o processo de otimização da distribuição de produtos originados em pontos de oferta e consumidos em pontos de demanda, dentro de uma rede de interligações possíveis, [GP00]. Estes problemas normalmente ocorrem dentro de plantas industriais, sistemas de comunicação e de transporte, distribuição de água, etc., contudo servem de modelo para inúmeras outras situações absolutamente diversas que lhes são assemelhadas por abstração. Normalmente, a oferta de cada produto, bem como a sua demanda, possui um valor conhecido. O processo de distribuição dos produtos não é realizado obrigatoriamente de um ponto de produção a um ponto de demanda, permitindo-se que pontos

intermediários tais como depósitos ou centros de concentração e distribuição sejam utilizados. As interligações podem possuir restrições de capacidade de tráfego e custos variados.

1.9.1 – Algoritmo Dual-Simplex (*out-of-kilter*)

Este algoritmo é similar ao algoritmo primal-dual e inicia com uma viabilidade dual, mas não necessariamente primal-viável e interage entre os problemas primal e dual até a otimalidade ser atingida. Entretanto, ele difere do algoritmo primal-dual quando o algoritmo *out-of-kilter* nem sempre mantém as folgas complementares. De fato, a principal verdade é se ater a completude das folgas. Uma versão do algoritmo pode ser projetada de modo a manter a viabilidade primal-dual (não necessariamente soluções básicas) e procurar atingir as folgas complementares.

Computacionalmente falando, o estado da arte dos códigos primal simplex rodam duas a três vezes mais rápido que os códigos de *out-of-kilter* pelos métodos tradicionais. Entretanto, avanços recentes nos métodos primal-dual que usam os ingredientes básicos do algoritmo *out-of-kilter*, apesar da forma diferente deste algoritmo, têm resultado em algoritmos que rodam duas a três vezes mais rápido do que os melhores códigos do primal-simplex com um fator de incremento de velocidade na resolução de problemas de larga escala (com milhares de vértices e arcos). Nesta visão, os conceitos sobre os vários passos desta classe de algoritmos são importantes de serem apresentados.

Por conveniência de apresentação, a forma do problema de fluxo em redes de custo mínimo (PFRCM) que trabalharemos será:

$$[PFRCM] \quad Min \sum_{i=1}^{m} \sum_{j=1}^{m} c_{ij} x_{ij}$$

sujeito a

$$[1] \quad \sum_{j=1}^{m} x_{ij} - \sum_{k=1}^{m} x_{jk} = 0 \quad (i = 1, ..., m)$$

$$[2] \quad x_{ij} \geq L_{ij} \qquad (i, j = 1, ..., m)$$

$$[3] \quad x_{ij} \leq U_{ij} \qquad (i, j = 1, ..., m)$$

onde se entende que as somas e as desigualdades são tomadas apenas sobre os arcos existentes. Chamamos conservação de fluxo (escolha dos $\mathbf{x_{ij}}$), todas as restrições que satisfazem a primeira restrição do modelo.

A conservação de fluxo que satisfaz as restrições remanescentes $\mathbf{l_{ij}} \leq \mathbf{x_{ij}} \leq \mathbf{u_{ij}}$ é um fluxo viável (solução). Assumimos que $\mathbf{c_{ij}}$, $\mathbf{l_{ij}}$ e $\mathbf{u_{ij}}$ são inteiros e que $0 \leq \mathbf{l_{ij}} \leq \mathbf{u_{ij}}$, onde $\mathbf{l_{ij}}$, $\mathbf{u_{ij}} \in \mathbf{Z}^{+}$.

Uma vez que nos valores do lado direito das equações de conservação de fluxo são zero, concluímos que o fluxo na rede não tem um ponto inicial ou um ponto final, mas circula continuamente através da rede. Assim toda conservação de fluxo na rede será ao longo de circuitos (ciclos orientados). Por esta razão, a representação é conhecida como Problema de Circulação de Fluxo em Redes.

Problema Dual do Problema de Circulação do Fluxo em Redes

Se associarmos uma variável dual $\mathbf{w_i}$ a cada equação de conservação do nó, uma variável dual $\mathbf{h_{ij}}$ a uma restrição $\mathbf{x_{ij}} \leq \mathbf{u_{ij}}$ e uma variável dual $\mathbf{v_{ij}}$ com uma restrição $\mathbf{x_{ij}} \geq \mathbf{l_{ij}}$, a formulação dual para o problema de fluxo de custo mínimo é dada por:

$$[PDFCM] \quad Max \sum_{i=1}^{m}\sum_{j=1}^{m} l_{ij} v_{ij} - \sum_{i=1}^{m}\sum_{j=1}^{m} u_{ij} h_{ij}$$

s.a.

$$[1]\ w_i - w_j + v_{ij} - h_{ij} = c_{ij} \quad (i, j = 1,..., m)$$

$$[2]\ h_{ij}, v_{ij} \geq 0 \quad (i, j = 1,..., m)$$

$$[3]\ w_i \text{ irrestrito,} \quad (i = 1,..., m)$$

onde os somatórios e as restrições são tomadas sobre arcos existentes. O problema dual tem uma estrutura muito interessante. Suponha que selecionamos qualquer conjunto de $\mathbf{w_i}$'s (assumimos durante o desenvolvimento que todos de $\mathbf{w_i}$'s são inteiros). Então a solução dual para cada arco (**i**,**j**) se torna,

$$\mathbf{v_{ij}} - \mathbf{h_{ij}} = \mathbf{c_{ij}} - \mathbf{w_i} + \mathbf{w_j}\ ,\ \mathbf{h_{ij}} \geq 0,\ \mathbf{v_{ij}} \geq 0$$

e uma solução dual viável é obtida através de:

$\mathbf{v_{ij}} = \text{Max} \{ 0, \mathbf{c_{ij}} - \mathbf{w_i} + \mathbf{w_j} \}$

$\mathbf{h_{ij}} = \text{Max} \{ 0, - (\mathbf{c_{ij}} - \mathbf{w_i} + \mathbf{w_j}) \}$

Assim o problema dual sempre possui uma solução viável dado qualquer conjunto de $\mathbf{w_i}$'s. De fato, as escolhas de $\mathbf{v_{ij}}$ e $\mathbf{h_{ij}}$ permitem atingir valores de $\mathbf{v_{ij}}$ e $\mathbf{h_{ij}}$ para um conjunto fixo de $\mathbf{w_i}$'s.

As Condições de Folgas Complementares

As condições de folgas complementares para otimalidade da formulação do *out-of-kilter* são as seguintes:

$(\mathbf{x_{ij}} - \mathbf{l_{ij}}) \, . \, \mathbf{v_{ij}} = 0 \, , \, \mathbf{i} \, , \, \mathbf{j} = 1, \ldots, \mathbf{m}$

$(\mathbf{u_{ij}} - \mathbf{x_{ij}}) \, . \, \mathbf{h_{ij}} = 0 \, , \, \mathbf{i} \, , \, \mathbf{j} = 1, \ldots, \mathbf{m}$

Define $\mathbf{z_{ij}} - \mathbf{c_{ij}} = \mathbf{w_i} - \mathbf{w_j} - \mathbf{c_{ij}}$. Então pela definição de $\mathbf{v_{ij}}$ e $\mathbf{h_{ij}}$ temos:

$\mathbf{v_{ij}} = \text{Max} \{ 0, - (\mathbf{z_{ij}} - \mathbf{c_{ij}}) \}$

$\mathbf{h_{ij}} = \text{Max} \{ 0, \mathbf{z_{ij}} - \mathbf{c_{ij}} \}$

Note que $\mathbf{z_{ij}} - \mathbf{c_{ij}}$ seria o coeficiente conhecido de $\mathbf{x_{ij}}$ na linha da função objetivo da tabela do simplex, se tivesse uma solução básica do problema primal.

Dado um conjunto de $\mathbf{w_i}$'s podemos computar $\mathbf{z_{ij}} - \mathbf{c_{ij}} = \mathbf{w_i} - \mathbf{w_j} - \mathbf{c_{ij}}$. Notando as equações $(\mathbf{x_{ij}} - \mathbf{l_{ij}}) \, . \, \mathbf{v_{ij}}$ e $(\mathbf{u_{ij}} - \mathbf{x_{ij}}) . \, \mathbf{h_{ij}}$, as condições de complementaridade se verificam. Portanto, nós obtemos o seguinte resultado chave das condições de otimalidade do problema, assim como o principio de certeza do algoritmo *out-of-kilter*.

Teorema 01: Seja $\boldsymbol{x}$ qualquer conservação de fluxo e seja $\boldsymbol{w} = (\mathbf{w_i}, \ldots, \mathbf{w_m})$ qualquer vetor formado por um conjunto de valores inteiros. Então $\boldsymbol{x}$ e $\boldsymbol{w}$ são respectivamente soluções primal e dual para o primeiro modelo se e somente se para todo $(\boldsymbol{i} \, , \boldsymbol{j})$.

$$\begin{cases} z_{ij} - c_{ij} < 0, \text{implica } x_{ij} = l_{ij} \\ z_{ij} - c_{ij} > 0, \text{implica } x_{ij} = u_{ij} \\ z_{ij} - c_{ij} = 0, \text{implica } l_{ij} \le x_{ij} \le u_{ij} \end{cases}$$

O problema então é procurar valores de $\mathbf{w_i}$'s e conservação $\mathbf{x_{ij}}$'s até que as condições do teorema 01 acima sejam satisfeitas.

Partindo de $\mathbf{w_i}$'s = 0, e uma conservação de fluxo, digamos, cada $\mathbf{x_{ij}}$ = 0, podemos avaliar a otimalidade. Fazendo a fase primal do algoritmo *out-of-kilter*, devemos mudar os $\mathbf{x_{ij}}$'s e tentar trazer os arcos para dentro do estado *in-kilter*. Durante a fase dual mudamos os $\mathbf{w_i}$**'**s e tentamos atingir o estado *in-kilter* como será descrito.

Os Estados *kilter* e Números *kilter* para um Arco

Os estados *in-kilter* e *out-kilter* para cada arco na rede são dados na tabela 1.4. Note que um arco está *in-kilter* se $\mathbf{l_{ij}} \le \mathbf{x_{ij}} \le \mathbf{u_{ij}}$ e as condições do teorema 01 são satisfeitas. À medida que mudamos o fluxo no arco (**i** , **j**), o arco se move acima e abaixo de uma coluna específica da tabela, dependendo se $\mathbf{x_{ij}}$ aumenta ou diminui. À medida que mudamos os $\mathbf{w_i}$, os arcos movem-se para trás e adiante da linha (mudando de coluna). A figura 1.31 dá uma visão gráfica dos estados *kilter* de um arco. Cada célula na matriz da tabela 1.4 corresponde a uma sub-região particular na figura 1.31.

Tabela 1.4: Estados *Kilter*

	$z_{ij} - c_{ij} < 0$	$z_{ij} - c_{ij} = 0$	$z_{ij} - c_{ij} > 0$
$x_{ij} > u_{ij}$			
$x_{ij} = u_{ij}$			
$l_{ij} < x_{ij} < u_{ij}$			
$x_{ij} = l_{ij}$			
$x_{ij} < l_{ij}$			

In_Kilter *Out_kilter*

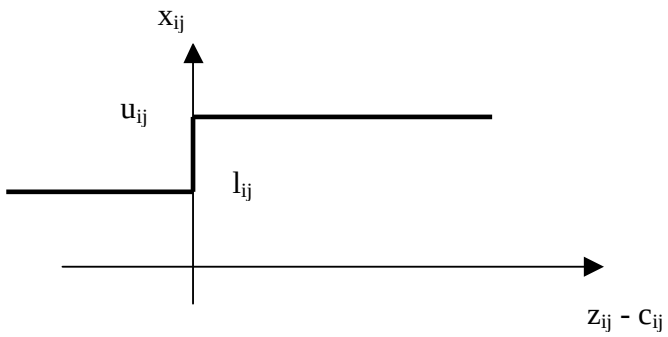


Figura 1.31: Função *in-kilter* (primal).

De modo a admitir que o algoritmo converge, necessitamos de alguma medida de distância da otimalidade. Podemos construir um algoritmo que periodicamente reduz a distância da otimalidade por um inteiro, então o algoritmo eventualmente convergirá.

Há muitas medidas de distância para o método *out-of-kilter*. Apresentamos na tabelas 13.5 e figura 1.32 uma medida de distância que chamamos de número *kilter* $\mathbf{K_{ij}}$ para o arco (**i**,**j**). O número *kilter* é definido como sendo a mudança mínima de fluxo no arco que é necessária para trazê-lo ao estado *in-kilter*.

O número *kilter* de um arco é ilustrado graficamente na figura 1.32. Note que desde que todos os termos envolvam valores absolutos, o número *kilter* para um arco é não negativo. Também note que se o arco está *in-kilter*, o número *kilter* associado é zero, e se o arco está *out-of-kilter*, o número associado é estritamente positivo. Note que se $\mathbf{z_{ij}} - \mathbf{c_{ij}} < 0$, então o arco (**i**,**j**) está *in-kilter* somente se o fluxo é igual a $\mathbf{l_{ij}}$ e, portanto, o número *kilter* $|\ \mathbf{x_{ij}} - \mathbf{l_{ij}}\ |$ indica quão longe o fluxo corrente $\mathbf{x_{ij}}$ está do caso ideal $\mathbf{l_{ij}}$. De modo similar, se $\mathbf{z_{ij}}$ - cij > 0, então o número *kilter* $|\ \mathbf{x_{ij}} - \mathbf{u_{ij}}\ |$ dá a distância do fluxo ideal de $\mathbf{u_{ij}}$.

Finalmente, se $\mathbf{z_{ij}} - \mathbf{c_{ij}} = 0$, então o arco está *in-kilter* se $\mathbf{l_{ij}} \leq \mathbf{x_{ij}} \leq \mathbf{u_{ij}}$.

Em particular, se $\mathbf{x_{ij}} > \mathbf{u_{ij}}$, então o arco é trazido ao estado *in-kilter* se o fluxo decresce de $|\mathbf{x_{ij}} - \mathbf{u_{ij}}|$, e se $\mathbf{x_{ij}} < \mathbf{l_{ij}}$, então o arco é trazido ao estado *in-kilter* se o fluxo cresce de $|\mathbf{x_{ij}} - \mathbf{l_{ij}}\ |$, e, portanto, os elementos na tabela 1.4 sobre a coluna $\mathbf{z_{ij}} - \mathbf{c_{ij}} = 0$.

Uma forma para atingir uma convergência finita do algoritmo *out-of-kilter* é garantir o seguinte:

- O número *kilter* de qualquer arco nunca cresce;

- Em intervalos finitos, o número *kilter* de algum arco é reduzido.

É exatamente o que o algoritmo *out-of-kilter* tem que ser capaz de atingir.

Tabela 1.5: Os números *kilter* K_{ij}.

	$z_{ij} - c_{ij} < 0$	$z_{ij} - c_{ij} = 0$	$z_{ij} - c_{ij} > 0$
$x_{ij} > u_{ij}$	$\lvert x_{ij} - l_{ij} \rvert$	$\lvert x_{ij} - u_{ij} \rvert$	$\lvert x_{ij} - u_{ij} \rvert$
$x_{ij} = u_{ij}$	$\lvert x_{ij} - l_{ij} \rvert$	0	0
$l_{ij} < x_{ij} < u_{ij}$	$\lvert x_{ij} - l_{ij} \rvert$	0	$\lvert x_{ij} - u_{ij} \rvert$
$x_{ij} = l_{ij}$	0	0	$\lvert x_{ij} - u_{ij} \rvert$
$x_{ij} < l_{ij}$	$\lvert x_{ij} - l_{ij} \rvert$	$\lvert x_{ij} - l_{ij} \rvert$	$\lvert x_{ij} - u_{ij} \rvert$

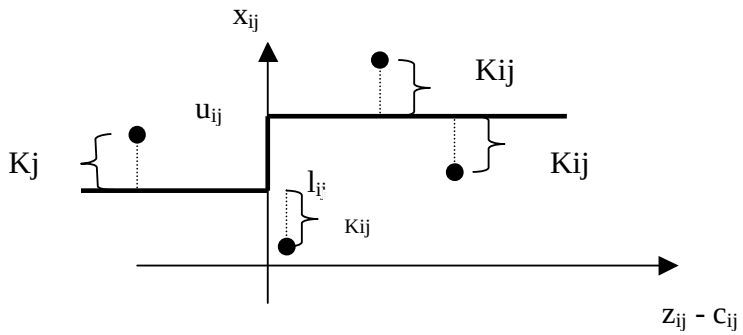


Figura 1.32: Os números *kilter* K_{ij}.

Estratégias do Algoritmo *out-of-kilter*

Como indicado antes, o algoritmo *out-of-kilter* deve ser geralmente visto como um algoritmo do tipo primal-dual.

Neste sentido, os passos genéricos do algoritmo são:

Passo 1: Inicie com uma conservação de fluxo, tal que cada $\mathbf{x_{ij}} = 0$, e uma solução viável para o dual com $\boldsymbol{w}$ arbitrariamente escolhido, tal que cada $\mathbf{w_i} = 0$, e com $\mathbf{h_{ij}}$, $\mathbf{v_{ij}}$ definidos pelas equações:

$\mathbf{v_{ij}}$ = Max { 0, - ($\mathbf{z_{ij}} - \mathbf{c_{ij}}$) }

$\mathbf{h_{ij}}$ = Max { 0, $\mathbf{z_{ij}} - \mathbf{c_{ij}}$ }

Identificar os estados *kilter* e os números *kilter* dos arcos.

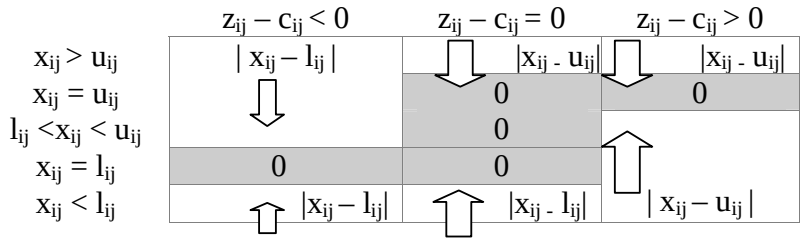
Passo 2: Se a rede tem um arco *out-of-kilter*, conduza a fase primal do algoritmo. Durante esta fase um arco *out-of-kilter* é selecionado e uma tentativa é feita de modo a construir uma nova conservação de fluxo de tal modo que o número *kilter* de nenhum arco seja piorado e que o arco selecionado seja melhorado.

Passo 3: Quando tal melhoria de fluxo não pode ser construída durante a fase primal, o algoritmo constrói uma nova solução dual de tal modo que nenhum número *kilter* seja piorado e em seguida repete-se o passo 2.

Passo 4: Interagindo entre os passos 2 e 3, o algoritmo constrói eventualmente uma solução ótima ou determina que nenhuma solução ótima existe.

Esta fase pode ser vista na tabela 1.6 e figura 1.33.

Tabela 1.6: Mudanças de direção de fluxo.



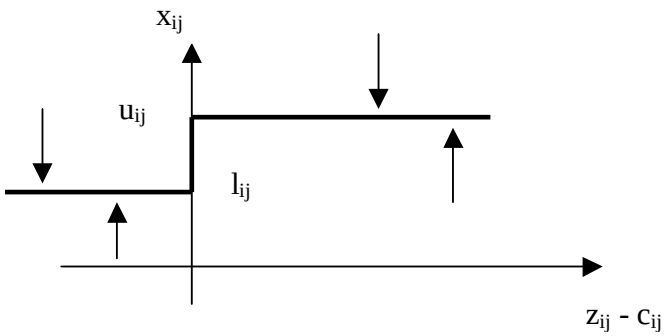


Figura 1.33: Mudanças de direção de fluxo.

Exemplo de funcionamento

Exemplificando o funcionamento do algoritmo de rotulação ao longo de todas as iterações, utilizaremos o grafo da figura 1.34 abaixo, e a descrição de seus custos na tabela 1.7. Nas tabelas 1.8, 1.9, 1.10, 1.11 temos iterações do *out-of-kilter* e na tabela 1.12 a solução encontrada.

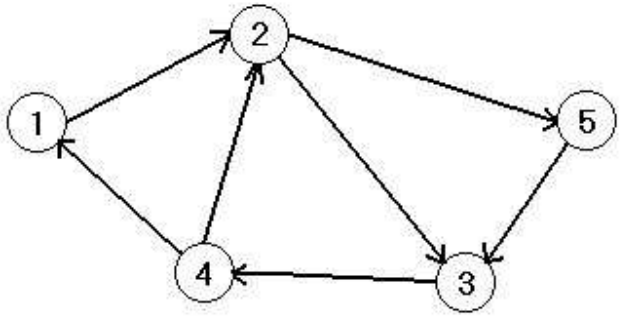


Figura 1.34: Grafo Original.

Tabela 1.7: Valores iniciais.

Arcos	$\mathbf{L_{ij}}$	$\mathbf{U_{ij}}$	$\mathbf{C_{ij}}$	
1 → 2	1	∞	3	
2 → 3	1	∞	2	Atributos dos Arcos
2 → 5	1	∞	3	L_{ij} = fluxo mínimo
3 → 4	1	∞	5	U_{ij} = fluxo máximo
4 → 1	1	∞	5	C_{ij} = Custo
4 → 2	1	∞	1	
5 → 3	1	∞	7	

Inicialização

Tabela 1.8: Tabela de inicialização para o *out-of-kilter*.

Arcos	w_i-w_j-c_{ij}	X_{ij}	N_o	W_i
1 → 2	-3	0	1	0
2 → 3	-2	0	2	0
2 → 5	-3	0	3	0
3 → 4	-5	0	4	0
4 → 1	-5	0	5	0
4 → 2	-1	0		
5 → 3	-7	0		

A seqüência de operações do algoritmo seria então:

1ª Iteração:

Passo 1: Procura a existência de um arco *out-of-kilter*.

Encontra-se o arco 1 → 2

Rotula-se o nó 2. R(2) = (+1)

Passo 2: Rotulação

R(3) = (+2)

R(5) = (+2)

R(4) = (+3)

R(1) = (+4)

Passo 3: O nó inicial foi rotulado (2). Delta = 1, modifica-se X_{ij} dos arcos: 1→2, 2→3, 3→4, 4→1

Tabela 1.9: Valores após a 1ª iteração.

Arcos	w_i-w_j-c_{ij}	X_{ij}	N_o	W_i
1 → 2	-3	1	1	0
2 → 3	-2	1	2	0
2 → 5	-3	0	3	0
3 → 4	-5	1	4	0
4 → 1	-5	1	5	0
4 → 2	-1	0		
5 → 3	-7	0		

2ª Iteração:

Passo 1: Procura a existência de um arco *out-of-kilter*.

Encontra-se o arco 2 → 5

Rotula-se o nó 5. R(5) = (+2)

Passo 2: Rotulação

R(3) = (+5)

Passo 4: Como o nó inicial (2), não foi rotulado, o algoritmo vem para o passo 4, alterando os w_i's.

Delta = 5

Altera-se os nós que não foram rotulados (1,2,4).

Tabela 1.10: Valores após 2ª iteração.

Arcos	w_i-w_j-c_{ij}	X_{ij}	N_o	W_i
1 → 2	-3	1	1	5
2 → 3	-2	1	2	5
2 → 5	-3	0	3	0
3 → 4	-5	1	4	5
4 → 1	-5	1	5	0
4 → 2	-1	0		
5 → 3	-7	0		

3ª Iteração:

Passo 2:

Como o arco 2 → 5 ainda não está em *In_Kilter*, continua a rotulação

R(5) = (+2)

R(3) = (+5)

R(4) = (+3)

R(2) = (+4)

Passo 3: O nó inicial foi rotulado(2). Delta = 1, modifica-se $\mathbf{X_{ij}}$ dos arcos:

$2 \rightarrow 5$, $3 \rightarrow 4$, $4 \rightarrow 2$, $5 \rightarrow 3$

Tabela 1.11: Valores após 3ª iteração.

Arcos	w_i-w_j-c_{ij}	X_{ij}	N_o	W_i
1 → 2	-3	1	1	5
2 → 3	-2	1	2	5
2 → 5	-3	1	3	0
3 → 4	-5	2	4	5
4 → 1	-5	1	5	0
4 → 2	-1	1		
5 → 3	-7	1		

4ª Iteração:

Passo 1: Fim da execução, todos os arcos In_Kilter.

Resultado Final

Tabela 1.12: Resultado final.

Arcos	Fluxo	
1 → 2	1	
2 → 3	1	
2 → 5	1	
3 → 4	2	
4 → 1	1	
4 → 2	1	
5 → 3	1	Custo = 31

1.10 – O Problema do Carteiro Chinês

Os problemas de Percurso em Arcos são dos mais antigos relacionados a grafos. A primeira referência que se conhece sobre eles vem do famoso problema das sete pontes de

Königsberg, figura 1.35. Buscava-se saber se havia um caminho fechado que atravessasse exatamente uma vez sete pontes sobre o rio Pregel em Königsberg, hoje Kaliningrad. O problema foi solucionado pelo matemático suíço Leonhard Euler (1736), que encontrou as condições para a existência de uma rota fechada (grafo euleriano), e mostrou que não havia solução que satisfizesse aquele caso particular, figura 1.36, [EU1736]. A preocupação de Euler foi exclusivamente sobre do caminho fechado, já a questão de determiná-lo foi resolvida 137 anos mais tarde por Heierholzer (1873) (percurso euleriano), [HE1873].

Figura 1.35: Visualização de Königsberg, e as sete pontes sobre o rio Pregel.

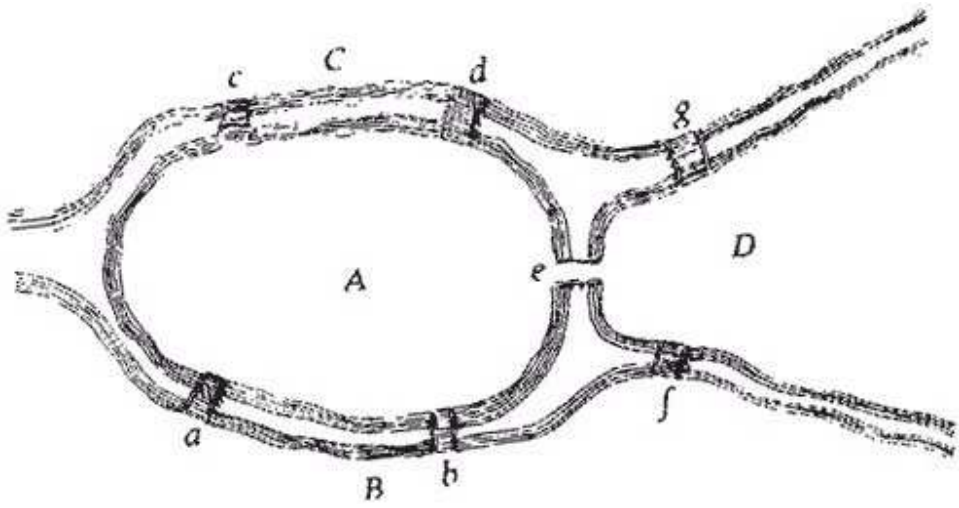


Figura 1.36: Grafo proposto por Euler em 1736, representando a situação do problema das sete pontes.

Anos mais tarde, em 1962, um matemático da Universidade Normal de Shangtun, Kwan Mei-Ko, quando de sua passagem como funcionário dos correios durante a revolução cultural chinesa, preocupou-se com uma situação semelhante à de Euler e Heierholzer, porém adequada ao percurso dos carteiros que atenderiam ruas de sua cidade. Neste caso, Kwan mostrou-se interessado em definir além da travessia, a forma mais fácil de fazê-la, percorrendo a menor distância possível. Kwan, definiu assim o problema: um carteiro tem que cobrir seu local de trabalho, antes de retornar ao posto. O problema é encontrar a menor distância de percurso para o carteiro, [KW62].

O termo percurso em arcos, foi estabelecido mais claramente anos mais tarde, em 1995 por Eilselt, Gendreau e Laporte, com dois trabalhos sobre divisões do Problema do Carteiro Chinês, e o estado da arte em algoritmos sobre o assunto, [EGL95]. A princípio, a notação considera um grafo **G** como um conjunto formado por vértices e ligações, **G** = (**V**,**L**). O conjunto das ligações pode ser ampliado para uma dupla de conjuntos, **L** = (**E**,**A**), onde **E** são denominadas as ligações não orientadas entre vértices de **G**, e **A** é o conjunto das ligações orientadas entre vértices de **G**.

A classificação variada dos diversos tipos de problemas de percursos em arcos abrevia-se no interesse deste trabalho, nos três principais tipos de abordagens do Problema do Carteiro Chinês. São elas:

1. **Problema do Carteiro Chinês – Caso Simétrico (PCCS)**, onde se deseja gerar um percurso de custo mínimo sobre um grafo **G** = (**V**,**E**), valorado e conexo, a partir de um vértice $\mathbf{v_0} \in \mathbf{V}$, origem.;
2. **Problema do Carteiro Chinês – Caso Dirigido (PCCD)**, onde se deseja gerar um percurso de custo mínimo sobre um grafo **G** = (**V**,**A**), valorado e fortemente conexo (*f*-conexo), a partir de um vértice $\mathbf{v_0} \in \mathbf{V}$, origem;
3. **Problema do Carteiro Chinês – Caso Misto (PCCM)**, onde se deseja gerar um percurso de custo mínimo sobre um grafo **G** = (**V**,**E**,**A**), valorado e fortemente conexo (*f*-conexo), a partir de um vértice $\mathbf{v_0} \in \mathbf{V}$, origem.

Segundo Edmonds & Johnson (1973), o PCCS e PCCD podem ser resolvidos em tempo polinomial, e segundo Papadimitriou (1976) o PCCM é NP-Hard, [EJ73], [PC76].

Apesar do problema tratar-se de uma aplicação aparentemente direcionada a carteiros, este problema também se aplica ao percurso de varredores de rua, leituristas de energia, teste topológico de sistemas computacionais em diferentes níveis, determinação de estados mínimos de energia em vidros spin, na minimização de vias em projetos de circuitos VLSI, na coleta de lixo, na remoção de neve, no corte de peças tais como o vidro e roupas, dentre vários outros, [NEG90], [EGL95] e [NEG96].

1.10.1 – Versão Orientada do PCC

No caso orientado/dirigido, o PCC pode ser resolvido a partir da determinação do multigrafo euleriano de **G** = (**V**,**A**), onde **G** é *f*-conexo (há pelo menos um caminho entre todo par de vértices da rede), através de um algoritmo de fluxo de custo mínimo ou circulação de custo mínimo em uma rede, ou através de uma transformação de **G** num grafo para o problema de transportes, [BB74], [EE73] e [YY88].

Seja **I** o conjunto dos vértices **v'**, onde $\mathbf{d^+(v_i)} > \mathbf{d^-(v_i)}$ e daí ligados a um novo vértice **f** denominado de fonte; e seja **J** o conjunto dos vértices **v** onde $\mathbf{d^-(v_j)} > \mathbf{d^+(v_j)}$, e daí ligados a um vértice **s**, o qual denominamos de sumidouro. Sendo $\mathbf{d^+(v_i)}$ definido como o número de arcos saindo e $\mathbf{d^-(v_i)}$ o número de arcos entrando em $\mathbf{v_i}$. A formulação do PCC-Orientado pode ser descrita como segue:

$$[PCCO] \quad Min \sum_{v_i \in I} \sum_{v_j \in J} c_{ij} x_{ij}$$

s.a.

$$[1] \quad \sum_{v_j \in J} x_{ij} = | d^+(v_i) - d^-(v_i) | \quad \forall v_i \in I$$

$$[2] \quad \sum_{v_i \in I} x_{ij} = | d^+(v_j) - d^-(v_j) | \quad \forall v_j \in J$$

$$[3] \quad x_{ij} \geq 0, \forall v_i \in I, \forall v_j \in J$$

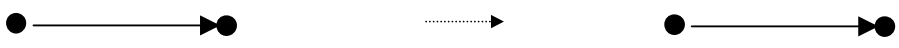
A formulação acima é definida como o Problema Clássico de Transportes adaptada ao caso de um grafo orientado para o problema do carteiro chinês orientado, [HI41], [KO49] e [AMO93].

Para resolver instâncias do PCC-Orientado, nossa preocupação agora é definir um procedimento genérico.

Procedure PCC-Orientado (**G**,**G'**)
// Algoritmo genérico para o PCC-Orientado

Passo 1: Tomar **G**, com as informações de distância, e testar a *f*-conectividade de **G**.

Passo 2: Se **G** é *f*-conexo, transformar as informações de distância para informações de fluxo, com os limites de fluxo máximo e mínimo permitidos por arco, e aplicar um algoritmo de fluxo em redes de custo mínimo, *out-of-kilter* (**G**, **G'**,Custo), [BHD90]. Senão Termine.



Passo 3: Após o passo e obtenção de **G'**, onde os resultados de fluxo indicam a quantidade de vezes que cada arco será repetido em **G'**, aplicar o resultado encontrado para formação da rota euleriana, usando o algoritmo c-euler.

Este algoritmo pode ser implementado em versões distintas para o Problema de Custo de Fluxo Mínimo, via algoritmos primal-dual (o*ut-of-kilter*), cuja complexidade pode chegar a $O(m^3)$, ou por outros métodos em $O(m^2 \log n)$, [PT90] , [OR84].

1.10.2 – Versão Mista do PCC

A versão mista do PCC está na classe de problemas NP-Árduos. Contém uma generalização dos grafos não-orientados, onde existem um conjunto de elos e arcos não vazios, [LR81].

Seja $\mathbf{G} = (\mathbf{V},\mathbf{L})$ um grafo fortemente conexo, onde $\mathbf{L} = (\mathbf{E},\mathbf{A})$, sendo $\mathbf{E} = \{(\mathbf{v_i},\mathbf{v_j}) : \mathbf{v_i},\mathbf{v_j} \in \mathbf{V}\} \neq \emptyset$ e $\mathbf{A} = \{<\mathbf{v_s},\mathbf{v_k}> : \mathbf{v_s}, \mathbf{v_k} \in \mathbf{V}\} \neq \emptyset$, e $\mathbf{G_m}$ o grafo aumentado de **G**, e $\mathbf{G_m}$ euleriano. **G** é par se o número de elos e arcos incidentes a todo vértice for par. **G** é simétrico se para cada vértice o número de arcos entrando é igual ao número de arcos saindo. Um grafo é balanceado, se as condições de balanceamento e unicursalidade são satisfeitas.

Propriedade da Unicursalidade

Seja **G** um grafo *f*-conexo. **G** é dito unicursal ou euleriano se existe um caminho fechado em **G** contendo cada aresta apenas uma vez e cada vértice pelo menos uma vez. As condições necessárias e suficientes para que um grafo *f*-conexo seja euleriano são dadas como segue:

1. Se **G** é não orientado (simétrico), todo vértice deve ter grau par, ou seja, um número par de elos incidentes – Teorema de Euler.

2. Se **G** é orientado, o número de arcos entrando e saindo de cada vértice é igual – Teorema de Ford e Fulkerson, [FF62];

3. Se **G** é misto todo vértice em **S** deve conter um número par de arcos orientados a ele ligados; além disto, para todo conjunto $\mathbf{S} \subseteq \mathbf{V}$, a diferença entre o número de

arcos de **S** para **V**-**S** e o número de arcos de **V**-**S** para **S** deve ser menor do que ou igual ao número de elos ligando **S** e **V**-**S** – Condições de balanceamento, [NP91].

Se **G** é par e simétrico, então ele é balanceado, porém a simetria não é uma condição suficiente a unicursalidade. Se já se sabe que **G** é euleriano, pode-se ter o problema de determinação de um circuito euleriano em **G**. Isto pode ser atingido em três fases:

1. Atribuir direções a alguns elos de forma que **G** seja simétrico;
2. Atribuir direções aos elos restantes;
3. Determinar a travessia atual de **G**.

Ford e Fulkerson propuseram o seguinte procedimento para transformar um grafo misto em grafo simétrico, [FF62].

Passo 1: Trocar cada elo de **G** com um par de arcos orientados opostos, assim obtendo um grafo orientado **G'** = (**V**, **A'**). Atribuir a cada arco $\mathbf{A} \cap \mathbf{A'}$ um limite inferior 1 e a cada arco $\mathbf{A'} \setminus \mathbf{A}$ um limite inferior 0. Atribuir também a cada arco de **A'** um limite superior de 1.

Passo 2: Usando um algoritmo de fluxo em redes, determinar uma circulação de fluxo viável em **G**. Fazer $\mathbf{x_{ij}}$ ser um fluxo no arco $<\mathbf{v_i}, \mathbf{v_j}>$.

Passo 3: Orientar alguns elos de **G** do seguinte modo : se $(\mathbf{v_i},\mathbf{v_j}) \in \mathbf{E}$, $\mathbf{x_{ij}} = 1$ e $\mathbf{x_{ij}} = 0$, orientar $(\mathbf{v_i},\mathbf{v_j})$ de $\mathbf{v_i}$ a $\mathbf{v_j}$.

Um procedimento para orientar completamente um grafo simétrico, pode ser colocado como segue.

Passo 1: Se todos os elos são orientados, parar.

Passo 2: Fazer **v** ser o vértice com pelo menos um elo/ligação incidente não orientado (**v**,**w**). Fazer $\mathbf{v_1} = \mathbf{v}$ e $\mathbf{v_2} = \mathbf{w}$;

Passo 3: Orientar $(\mathbf{v_1},\mathbf{v_2})$ de $\mathbf{v_1}$ a $\mathbf{v_2}$. Se $\mathbf{v_2} = \mathbf{v}$, ir ao passo 1.

Passo 4: Fazer $\mathbf{v_1} = \mathbf{v_2}$ e identificar um elo $(\mathbf{v_1},\mathbf{v_2})$ incidente a $\mathbf{v_1}$. Ir ao passo 3.

Nobert e Picard (1991) propuseram uma formulação matemática onde há apenas uma variável $\mathbf{y_{ij}}$, associada a cada elo de **E**. A solução de programação inteira, portanto, não especifica a dimensão dos elos. São impostas restrições para que o grafo aumentado satisfaça as condições necessárias e suficientes de unicursalidade, ou seja, o grafo deve ser par e balanceado, [NP91].

Seja um subconjunto próprio e qualquer ***S*** de ***V***, onde ***V*** é o conjunto de vértices, o conjunto:

$\mathbf{A^+(s)} = \{<\mathbf{v_i},\mathbf{v_j}> \in \mathbf{A : v_i} \in \mathbf{S}, \mathbf{v_j} \in \mathbf{V\backslash S}\}$

$\mathbf{A^-(s)} = \{<\mathbf{v_i},\mathbf{v_j}> \in \mathbf{A : v_i} \in \mathbf{V\backslash S}, \mathbf{v_j} \in \mathbf{S}\}$

$\mathbf{E(s)} = \{(\mathbf{v_i},\mathbf{v_j}) \in \mathbf{E : v_i} \in \mathbf{S}, \mathbf{v_j} \in \mathbf{V\backslash S} \vee \mathbf{v_i} \in \mathbf{V\backslash S}, \mathbf{v_j} \in \mathbf{S}\}$

e seja $\mathbf{u(S)} = |\mathbf{A^+(s)}| - |\mathbf{A^-(s)}| - |\mathbf{E(s)}|$. Portanto, se $\mathbf{S} = \{\mathbf{v_k}\}$, então $\mathbf{A^+(s)}=\mathbf{A^+(k)}$, $\mathbf{A^-(s)} =\mathbf{A^-(k)}$, e $\mathbf{E(s)} = \mathbf{E(k)}$.

As constantes $\mathbf{p_k}$ e as variáveis $\mathbf{z_k}$ são definidas como seguem, e apenas uma variável $\mathbf{y_{ij}}$ é definida para cada $(\mathbf{v_i},\mathbf{v_j})$ que devem ser adicionadas ao grafo para torná-lo euleriano. A formulação é dada a seguir.

(PCCM -1) Minimizar $\sum_{<v_i,v_j>\in A} c_{ij}\, x_{ij} + \sum_{<v_i,v_j>\in E} c_{ij}\, x_{ij}$

s.a.

[1] $\sum_{<v_i,v_j>\in A^+(s)} x_{ij} + \sum_{<v_i,v_j>\in A^-(s)} y_{ij} = 2z_k + p_k \ (v_k \in V)$

[2] $\sum_{<v_i,v_j>\in A^+(s)} x_{ij} - \sum_{<v_i,v_j>\in A^-(s)} x_{ij} + \sum_{<v_i,v_j>\in E} y_{ij} \geq u(S), \ \forall S \subset V, s \neq \phi$

[3] $z_k, x_{ij}, y_{ij} \in Z_+$

Nesta formulação, os dois primeiros conjuntos de restrições forçam com que todos os subconjuntos próprios não vazios **S** e **V** sejam balanceados, isto é feito pela imposição de que um número suficiente de arcos e elos seja introduzido para compensar a falta de balanceamento de **u**(**S**). A inclusão destas restrições foi feita como uma forma generalizada para o tratamento das desigualdades de *blossons*, como pode ser visto na quarta restrição.

$$\sum_{<v_i, v_J> \in A+(s)} x_{ij} - \sum_{<v_i, v_j> \in A-(s)} x_{ij} + \sum_{<v_i, v_j> \in E} y_{ij} \geq 1, \forall S \subset V, V \text{ é impar}$$

Usando essa formulação, Nobert e Picard (1991) resolveram instâncias de grafos com vértices variando entre $16 \leq |\mathbf{V}| \leq 225$ e o número de arcos variando em torno de $2 \leq |\mathbf{A}| \leq 5569$ e elos em torno de $15 \leq |\mathbf{E}| \leq 4455$ em tempo razoável, [NP91].

Algoritmo de Sherafat para o PCC Misto

Uma outra abordagem foi explorada por Sherafat (1988), a qual não considera inicialmente as condições de unicursalidade do grafo. A idéia central é buscar a unicursalidade ao longo do processo de encaminhamento da solução. Sherafat aplicou um modelo de fluxo para um grafo transformado em puramente orientado, onde os elos são convertidos em pseudo-ramos, figura 1.37. A direção do fluxo nos pseudo-ramos (arcos adicionados na transformação) é considerada na arborescência da solução do PCC-Misto, elemento que facilita o cálculo do percurso do carteiro, [SH88].

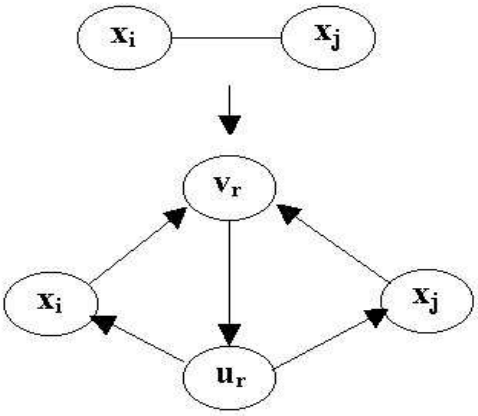


Figura 1.37: Conversão de um elo em pseudo-ramos não unicursais.

No problema de fluxo, a estrutura dos pseudo-ramos da figura 1.37 permite a passagem de fluxo de $\mathbf{x_i}$ a $\mathbf{x_j}$, via os arcos $<\mathbf{x_j}, \mathbf{v_r}>$, $<\mathbf{v_r}, \mathbf{u_r}>$ e $<\mathbf{u_r}, \mathbf{x_i}>$ ou na direção contrária via arcos $<\mathbf{x_i}, \mathbf{v_r}>$, $<\mathbf{v_r}, \mathbf{u_r}>$ e $<\mathbf{u_r}, \mathbf{x_j}>$. Em ambas as direções o fluxo existirá, obrigatoriamente, passando por $<\mathbf{u_r}, \mathbf{v_r}>$. A este arco associa-se o custo de $(\mathbf{x_i},\mathbf{x_j})$.

Chamando de $\mathbf{s} \in \mathbf{S}$ os arcos dos pseudo-ramos, o grafo transformado **G'** terá então novos vértices, $\mathbf{V'} = \mathbf{X} \cup \mathbf{V} \cup \mathbf{U}$, e **A'** como um novo conjunto de arcos.

O seguinte modelo se aplica ao problema transformado:

$$\text{(PCCM - 2) Minimizar} \sum_{<v_i, v_j> \in A} c_{ij}\, f_{ij} + \sum_{s \in S} c_{ij}\, f_{ij}$$

s.a.

[1] $\quad f_{ij} \geq 1, \quad \forall < x_i, x_j > \in (A \cup S)$

[2] $\quad \sum_{x_i \in V'} f_{ij} - \sum_{x_k \in V'} y_{ik} = 0, \quad (\forall\, x_j \in V')$

[3] $\quad f_{ij} \in Z_+, \forall < x_i, x_j > \in (A \cup S)$

Utilizando um algoritmo de fluxo de custo mínimo tipo primal-dual simplex para redes *out-of-kilter*, [BHD90], obtém-se uma tentativa no direcionamento dos ramos de modo a se achar as réplicas de arcos e elos visando um circuito completo de custo mínimo. Na solução ótima, haverá pelo menos uma unidade de fluxo circulando em cada arco e pseudo-ramo. Em cada pseudo-ramo $\mathbf{r_{ij}}$, o fluxo poderá passar de $\mathbf{x_i}$ a $\mathbf{x_j}$, ou de $\mathbf{x_j}$ a $\mathbf{x_i}$, ou eventualmente em ambos os sentidos. Entretanto, do modo como o modelo foi proposto, uma circulação de fluxo triangular não desejável em alguns pseudo-ramos poderá ocorrer, como nas figuras 1.38 e 1.39.

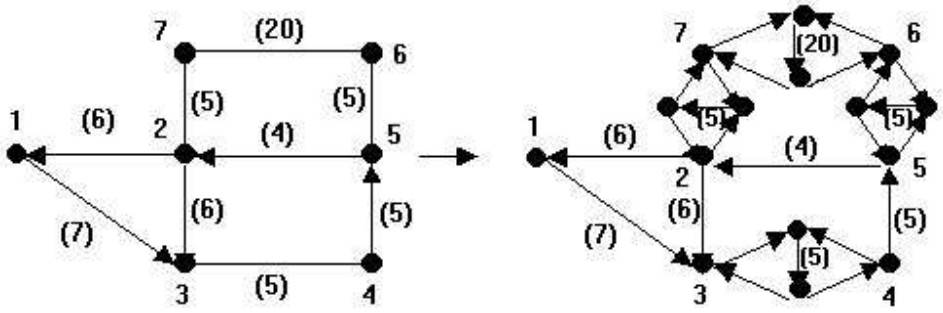


Figura 1.38: Grafo onde a aplicação direta do algoritmo de custo mínimo atinge a solução do problema sem a formação de circuitos triangulares, custo da rota ótima = 73.

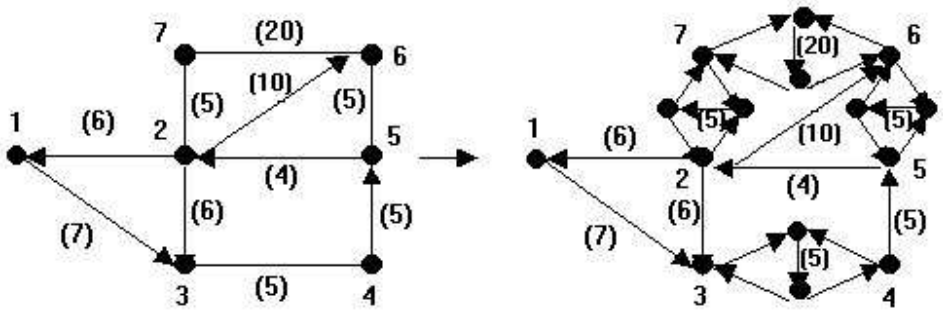


Figura 1.39: Grafo onde a aplicação direta do algoritmo de custo mínimo não atinge a solução do problema sem a formação de circuitos triangulares, o fluxo triangular acontece em (6,12,13,6), ao redirecioná-lo a rota ótima é atingida após 7 iterações com custo da rota = 92.

Duas observações são feitas quanto às propriedades de **G'**. A primeira é relativa ao direcionamento da passagem de fluxo nos pseudo-ramos. Se num arco $\langle \mathbf{x_i}, \mathbf{x_j} \rangle \in \mathbf{A}$, exige-se uma unidade de fluxo, ela deverá achar um caminho Γ que ligue o vértice $\mathbf{x_j}$ ao $\mathbf{x_i}$, para fechar o circuito. Γ poderá conter alguns pseudo-ramos, os quais serão forçosamente orientados conforme a direção do fluxo.

A segunda observação é quanto à passagem adicional de fluxo nos pseudo-ramos já orientados. Enquanto houver a possibilidade de direcionamento de outros ainda não orientados, uma vez que a simples orientação de tais ramos não acarreta em acréscimo no

custo da solução, passar por eles uma unidade de fluxo. Este procedimento promove o direcionamento de uma maior quantidade de fluxo possível nos pseudo-ramos.

Dado o PCC misto, se na sua solução todos os pseudo-ramos forem orientados, a solução ótima foi obtida, ou seja, as ligações do grafo **G** deverão ser multiplicadas / orientadas conforme o fluxo obtido para o grafo **G'**. Porém, se alguns pseudo-ramos permanecerem com fluxo triangular, recorre-se a uma rotina de *branch-and-bound*.

Considerando $\mathbf{r} \in \mathbf{R_0}$, onde $\mathbf{R_0}$ representa o conjunto de todos os pseudo-ramos com fluxo triangular. Como foi visto, **r** corresponde a um pseudo-ramo $\mathbf{r_{ij}} \in \mathbf{R}$ e é constituído por um conjunto de cinco arcos (dos pseudo-ramos). Fixando a direção de fluxo neste ramo, num dos sentidos $<\mathbf{x_i}, \mathbf{x_j}>$ ou $<\mathbf{x_j}, \mathbf{x_i}>$, temos:

$$\text{(i)} \begin{cases} f(x_i, v_j) \geq 1 \\ f(u_r, x_i) \geq 1 \end{cases} \qquad \text{(ii)} \begin{cases} f(x_j, v_r) \geq 1 \\ f(u_r, x_i) \geq 1 \end{cases}$$

Onde $f(\mathbf{x_i}, \mathbf{x_j})$ é o fluxo no ramo $<\mathbf{x_i}, \mathbf{x_j}>$.

Inserindo as restrições (i) e (ii) ao problema original, duas situações poderão ocorrer novamente. A primeira é quando a solução é viável para todos os pseudo-ramos, neste caso encontrou-se um limite superior do problema. A segunda deve-se a um novo $\mathbf{R_1}$ de pseudo-ramos com fluxo triangular que foi obtido, para este caso tem-se que repetir o procedimento, fixando a direção de fluxo para um novo $\mathbf{r} \in \mathbf{R}$. Tal procedimento deverá prosseguir numa árvore até que a solução seja ótima.

Um algoritmo exato para o PCC misto considerando as colocações acima pode ser descrito como segue:

Procedimento PCC-Misto (**G** , $\mathbf{G_m}$, Custo)
Dados de Entrada: Grafo (Vértices, Ligações e atributos)
Dados de Saída: Multigrafo com as réplicas das ligações e Custo Global – ou perímetro do multigrafo
Passo 0 : Fazer a transformação de **G** usando pseudo-ramos.
Passo 1 : Resolver o problema PCC Misto-2, **P**(0). Se o fluxo é viável em todos os seus pseudo-ramos a solução ótima foi encontrada. Ir ao passo 5. Caso contrário, fazer **LS** de custo igual a infinito. Fazer **i** = 0 e ir ao passo 2.

Passo 2 : *Branching* – escolher um pseudo-ramo $\mathbf{r_k}$ pertencer a $\mathbf{R_i}$ para direcionar. Fazer **i** = **i**+1. O novo problema a ser resolvido **P**(**i**) é constituído por **P**(**i**–1) acrescido de:

$f(\mathbf{x_i}, \mathbf{v_{rk}}) \geq 1$

$f(\mathbf{u_{rk}}, \mathbf{x_j}) \geq 1$

Ir ao passo 3.

Passo 3 : *Bounding* – Resolver o problema **P**(**i**). Calcular o limite inferior do custo de circulação, **LI**.

Se **LI** ≥ **LS** ir ao Passo 4.

Se **LI** < **LS** e a solução corrente é viável, fazer **LS** = **LI**, armazenar o vetor de fluxo e ir ao passo 4.

Se **LI** < **LS** mas a solução não é viável, ir ao passo 2.

Passo 4 : *Backtracking* - seja **t** pertencente a $\mathbf{R_j}$ o último pseudo-ramo orientado, mas ainda não re-orientado; **j** representa um estágio em que **t** foi orientado. Redirecionar **t** no sentido contrário. Fazer **i** = **i** + 1. O novo problema **P**(**i**) é **P**(**j**+1) acrescido por,

$f(\mathbf{x_i}, \mathbf{v_{rt}}) \geq 1$
$f(\mathbf{v_{rt}}, \mathbf{x_j}) \geq 1$

vá ao passo 3. Se não existir nenhum pseudo-ramo **t** não re-orientado, a solução ótima foi encontrada. Ir ao passo 5.

Passo 5 : Gerar o grafo aumentante de **G** a partir da solução armazenada no último vetor de fluxo registrado. Terminar.

No algoritmo de Sherafat (1988), os pseudo-ramos que resistem ao direcionamento não acontecem em número fixo. Na verdade, se limita ao número de problemas que serão criados no algoritmo *branch-and-bound*. Cada pseudo-ramo orientado nos passos 2 e 4 deverá fechar um novo ramo orientado em algum $\mathbf{r} \in \mathbf{R_i}$, e/ou duplicar outros que já têm fluxo. Pelas razões já expostas, os ramos em $\mathbf{R_i}$ por não aumentarem o custo teriam mais do que os outros, possibilidades de atraírem esse fluxo e se auto-direcionarem. Desta forma, os pseudo-ramos

sem fluxo viável se esgotam rapidamente. Além do mais, a solução do problema de fluxo **P(i)**, no passo 3 do algoritmo, geralmente requer um pouco de esforço computacional, uma vez que o problema **P(i**-1) é resolvido na etapa anterior. Na maioria das vezes, poucas atualizações nas variáveis primal e dual do algoritmo primal-dual simplex são suficientes, [SH88].

Nos resultados apresentados pelo autor, as soluções mostraram-se muito demoradas à medida que o número de elos aumenta proporcionalmente aos arcos. É de se esperar que esta técnica se torne atrativa quando se tem um grafo onde a dominância é maior no número de arcos, o que de fato observamos na execução do algoritmo após a sua implementação.

É importante reforçar aqui que, na forma como é resolvido o PCC-Misto, tem-se uma aproximação a um processo de puro direcionamento de um grafo. Isto significa que quanto maior a parte da rede com elos, pior será o desempenho deste método. Assim, para grafos puramente simétricos, recomenda-se que se use o método simétrico de Edmonds & Johnson e não este.

1.11 – O Problema do Carteiro Rural

O PCR (Problema do Carteiro Rural) é um problema de percurso em arcos, como o PCC.

Dado um grafo **G** = (**V**, **E**, **A**), e um outro conjunto **G'** = (**E'**, **A'**), onde **E'** é o conjunto de elos requeridos e **A'** é o conjunto de arcos requeridos. O termo requerido significa que sua passagem é obrigatória. O PCR consiste em, a partir de um vértice origem, a geração de um percurso que passe por todos os elementos e **E'** e **A'** e retorne ao ponto de origem. No PCR não é obrigatória a passagem por todas as arestas do grafo, somente as requeridas e as que forem necessárias para a formação do percurso.

Assim como o PCC, o PCR possui também três versões, que são:

1. **Problema do Carteiro Rural – Caso Simétrico**, onde se deseja gerar um percurso de custo mínimo sobre um grafo **G** = (**V**,**E**), valorado e conexo, a partir de um vértice $v_0 \in \mathbf{V}$, origem, passando por todos os elos requeridos;

2. **Problema do Carteiro Rural – Caso Dirigido**, onde se deseja gerar um percurso de custo mínimo sobre um grafo **G** = (**V**,**A**), valorado e fortemente conexo (*f*-conexo), a partir de um vértice $\mathbf{v_0} \in \mathbf{V}$, origem, passando por todos os arcos requeridos;
3. **Problema do Carteiro Rural – Caso Misto**, onde se deseja gerar um percurso de custo mínimo sobre um grafo **G** = (**V**,**E**,**A**), valorado e fortemente conexo (*f*-conexo), a partir de um vértice $\boldsymbol{v_0} \in \mathbf{V}$, origem, passando por todos os elos e arcos requeridos.

A figura 1.40 mostra um grafo com arcos requeridos, e a figura 1.41 mostra a solução do PCR, onde somente são percorridas as arestas requeridas e as necessárias para a geração do percurso, retornando à origem.

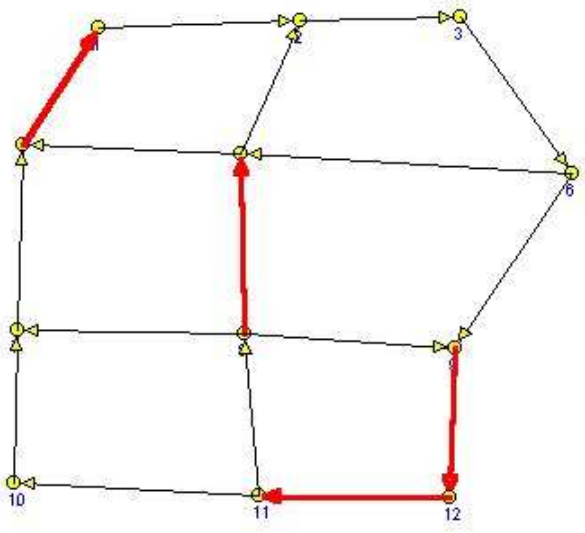


Figura 1.40: Grafo com arcos requeridos.

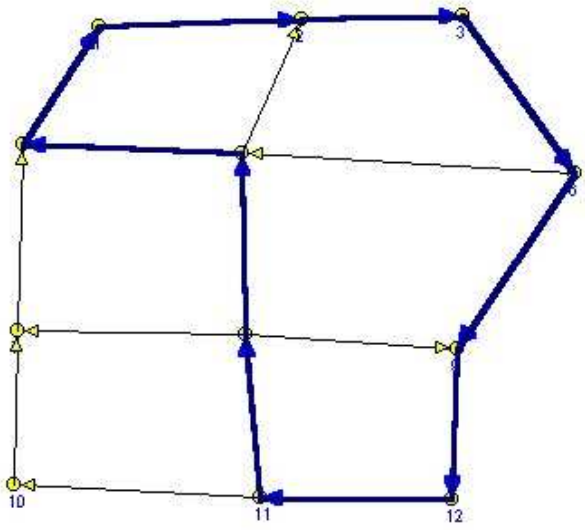


Figura 1.41: Solução para o PCR.

Este texto considerará a abordagem sobre o grafo misto, porém os algoritmos desenvolvidos podem ser perfeitamente estendidos para o caso simétrico sem perda de generalidade, considerando na fase do PCC a versão do algoritmo que o compete.

2. Problemas de Escalonamento de Tarefas

2.1 – Introdução

O problema de escalonamento periódico é um problema de otimização que em muitas vezes na prática não é aplicado de maneira ideal, ou seja, na resolução de instâncias deste problema não é utilizada nenhuma técnica mais elaborada, somente a experiência dos profissionais que cuidam do processo. Choques de horários, má distribuição da frota de veículos, ou mesmo a sub-utilização dos materiais são fatos que ocorrem no dia a dia das instituições que trabalham com escalonamento, seja de veículos, funcionários, atendimento a zonas, etc.

A implementação de um modelo matemático que atenda a todas as necessidades do escalonamento ajudaria a encontrar uma solução inicial ou próxima da solução ótima, que após a avaliação de um especialista seria posta em prática, reduzindo gastos desnecessários, e realizando de forma mais plena as tarefas necessárias.

Este capítulo visa a implementação de uma metodologia de solução para o problema de escalas periódicas em zonas de serviço sistemático, onde diversas variáveis são levadas em conta na busca de uma solução viável e factível de ser realizada na prática. Após esta solução inicial, procuramos demonstrar a utilização de meta-heurísticas para a melhoria das soluções.

2.2 – Revisão Bibliográfica

O problema de escalas de serviços consiste em alocar determinados recursos, que podem ser trabalhadores humanos, veículos, máquinas, a tarefas que são atribuídas, formando uma escala de serviço, podendo ter horários pré-determinados [GUN99], [BEPSW96]. Este

problema é muito difundido nas mais diversas áreas, e há diversas técnicas de pesquisa operacional para a sua resolução, [CHE00], [GUN99], [BEPSW96].

O problema de escalonamento é uma importante área da otimização combinatória, e é definido como sendo o problema de alocar recursos escassos a atividades através do tempo, [BEPSW96].

Alguns exemplos de aplicações de escalas de serviços seriam: construção de escalas de trabalho para funcionários, como motoristas e cobradores de ônibus, escalas de trabalho para tripulação de trens e aviões, alocação de veículos a zonas de serviços, distribuição do gás e água, planejamento da produção, escalonamento computacional, dentre várias outras, [BEPSW96], [GUN99], [NEG90].

Alguns trabalhos mostram métodos heurísticos para resolver estes problemas, tais como relaxações lineares ou algoritmos heurísticos baseados em métodos exatos. Estas aplicações heurísticas são feitas em problemas que envolvem muitas restrições, pois os métodos exatos não demonstram eficiência computacional nestes casos. Existe uma ampla bibliografia sobre construção de jornada de trabalho para funcionários, com diversas técnicas distintas, [SIQ99], [SHA99].

Uma aplicação do problema de escalonamento é o problema de escalonamento periódico multidimensional onde se deve determinar para cada operação um período, um tempo inicial e uma unidade de processamento na qual será executada. Este problema é NP-Árduo, [VLAM00]. Dentre as suas aplicações reais deste problema, temos o planejamento do processamento de sinais digitais, [BEPSW96]. O problema que abordamos pode ser considerado como sendo uma instância do problema multidimensional, porém, também é NP-Árduo.

Outro problema é o escalonamento periódico dinâmico, onde se faz uma escala de serviços ao longo de um período, e durante alguma unidade de tempo do período, as escalas têm que mudar, replanejando a escala preliminar, ou seja, mudando dinamicamente a tomada de decisão ao longo do tempo, reaproveitando os recursos já utilizados. Uma versão deste problema pode ser encontrada em Cheton & Chetto (2000), onde se comenta sobre o escalonamento em tempo real e as tarefas que são demandadas ao longo do período de trabalho, [CHE00].

Um problema de escalonamento relacionado a transportes é o problema de escalonamento de locomotivas, que consiste em associar um grupo de locomotivas a cada trem de um pré-planejamento de trens, de forma a prover aos trens força suficiente para puxá-los das origens a seus destinos, [ALOSS02].

O problema do escalonamento da força de trabalho é um problema que consiste em determinar quantos trabalhadores devem ser associados a cada período de planejamento do tempo de trabalho, [GUN99]. Uma solução para o problema é resolver a escala de trabalho para cada empregado, especificando o tempo de trabalho e de descanso através do período determinado. Como aplicação prática, se enquadram neste problema escalas em hospitais, aeroportos, operadores telefônicos, patrulhas de departamentos de polícia, dentre outros.

Uma das formas de se resolver problemas de escalonamento desta natureza é através da programação em lógica e/ou da programação por restrições (*Constraint Programming*), particularmente aqueles que contenham muitas restrições complexas uma vez que podem ser modelados e expressos em resolvedores (por exemplo, CPLEX – SOLVER) que são especificamente elaborados para tratar com sucesso problemas de escalas, [HOK00].

Outros problemas clássicos de escalonamento de tarefas são o *open-shop*, o *flow-shop* e o *job-shop*. Basicamente eles consistem em alocar atividades que são executadas por processadores no tempo, com variações da forma de alocação, ou com a obrigação de passar por um determinado recurso e depois por outro, como numa linha de montagem.

2.3 – O Problema de Escalonamento Periódico

Em geral, problemas de escalonamento são caracterizados por três conjuntos: o conjunto **T** = $\{t_1, t_s, ..., t_n\}$ com *n* tarefas ou atividades, o conjunto **P** = $\{p_1, p_2, ..., p_m\}$ com *m* processadores ou executores, e o conjunto **R** = $\{r_1, r_2, ..., r_s\}$ com *s* tipos de recursos adicionais. Escalonamento significa associar processadores a partir de **P** e possivelmente recursos a partir de **R,** a tarefas a partir de **T**, com o objetivo de cumprir todas estas atividades de acordo com restrições impostas, [BEPSW96].

Existem duas restrições na teoria clássica do escalonamento: cada atividade pode ser executada por no máximo um processador num intervalo de tempo, e cada processador é capaz de processar no máximo uma tarefa por vez. Os processadores podem ser paralelos ou

dedicados. Em paralelo, eles executam a mesma função, e dedicados executam tarefas específicas.

Em geral, **T** é caracterizado pelos seguintes dados:

1. *Vetor de tempos de processamento*, onde este é o tempo necessário para que cada processador execute dada tarefa de **T**;
2. *Freqüência* ou *periodicidade*, que é o tempo em que cada tarefa está pronta para ser repetida;
3. *Tempo planejado de execução*, que é o tempo que cada tarefa tem para ser executada;
4. *Tempo real de execução*, que é o tempo real que cada tarefa tem para ser completada;
5. *Peso*, que é um fator que pode ser multiplicado às tarefas.

Definimos como planejamento da atividade o processo de decidir dinamicamente ou não a associação das tarefas e sua execução, [BBGT98]. A atividade de planejamento tem como estrutura um período de tempo, onde a unidade mínima é um dia. O período é o calendário de distribuição, onde as tarefas associadas a cada elemento de distribuição, que no caso específico do problema em estudo são os veículos de uma frota, se encarregará de fazer numa dada região ou zona.

Essas tarefas, além de estarem associadas a elementos de distribuição, também estão ligadas a zonas de distribuição, aumentando o nível de complexidade do planejamento. Este planejamento também depende de outras condições, previamente definidas. Estas condições são responsáveis por dar seqüência à distribuição, garantindo que todos os grupos sejam atendidos; são ditas capacitadas, pois cada veículo tem uma capacidade física para o armazenamento; e por fim, condições de periodicidade, pois se ocupam em garantir a freqüência em que cada grupo terá que ser visitado. O objetivo será definir a quantidade e a capacidade dos veículos, a quantidade de zonas de atendimento, a periodicidade mínima e máxima para atendimento.

Antes de começarmos propriamente, falaremos de um conceito comum para o escalonamento, que é o de preempção. Diz-se um modelo é preemptivo se ele pode interromper a sua execução. No nosso caso, o problema não considera preempção e, portanto,

os algoritmos não são preemptivos, pois leva-se em conta que eles não poderão ter sua execução interrompida até que cada unidade de execução tenha completado todos os seus ciclos periódicos.

2.4 – Um Modelo Matemático para o Problema de Escalonamento Periódico de Veículos (PEPV) em Zonas

Este problema é uma proposta preliminar de modelo de escalas. Para este problema, algumas condições foram impostas para facilitar o desenvolvimento do modelo e da implementação. Consideramos que apenas um veículo pode percorrer uma única área em um único dia, e que um veículo atende completamente uma área em um único dia.

Considerando as seguintes variáveis como:

Variáveis

y^k , número de veículos num determinado dia k

$x_v^{d,a}$ $\begin{cases} 1, \text{se o veículo v está alocado no dia d a zona a} \\ 0, \text{caso contrário} \end{cases}$

Parâmetros

l^a- Limite inferior da freqüência de atendimento na área *a*;
u^a- Limite superior da freqüência de atendimento na área *a*;
$f^a \in [l^a , u^a]$, intervalo da freqüência de visitação na área a;

ND – Número de dias do horizonte de planejamento;
NV – Número de veículos máximo da frota;
NA – Número de áreas disponíveis.

Modelo matemático

$$[\text{OB1}]\ \min \sum_{d=1}^{ND} y^d$$

$$[\text{OB2}]\ \min\left(\max y^d\right)$$

$$[\text{PEPV}]\ \min\ z$$

s.a.

$$(1)\quad y^d \le z,\ \forall d \le ND$$

$$(2)\quad x_v^{d,a} \ge \sum_{s=l^a}^{u^a} x_v^{d+s,a},\ \forall a \ge 1, \forall d + s \le ND, \forall v \ge 1$$

$$(3)\quad \sum_{a=1}^{NA} x_v^{k,a} \le 1 \quad ,\ \forall d \ge 1, \forall v \ge 1$$

$$(4)\quad \sum_{v=1}^{NV}\left(\sum_{a=1}^{NA} x_i^{d,a}\right) \le y^d,\ \forall d \ge 1$$

$$(5)\quad \sum_{v=1}^{NV}\sum_{d=1}^{ND} x_i^{k,a} \ge \left\lceil \frac{ND}{u^a} \right\rceil,\ \forall a \ge 1$$

$$(6)\quad \sum_{v=1}^{NV}\sum_{d=1}^{ND} x_v^{d,a} \le \left\lceil \frac{ND}{l^a} \right\rceil,\ \forall a \ge 1$$

$$y^k \in Z^+,\ \ \forall d \ge 1..ND$$

$$x_v^{d,a} \in \{0,1\},\ \ \forall v \ge 1..NV, \forall d \ge 1..ND, \forall a \ge 1..NA$$

[OB1] Minimizar o número de veículos utilizados no período

[OB2] Minimizar a frota

(1) Mudança de função objetivo do tipo min(Max) para min z;

(2) Atribuição de um veículo *v* a uma área *a* num determinado dia *d;*

(3) Para um dado dia *d*, um veículo *v* só vai para uma área *a;*

(4) Quantidade de veículos utilizados num determinado dia *d*;

(5) Garantia de que todas as áreas serão visitadas no mínimo a relação entre o número de dias do horizonte de planejamento dividido pela máxima freqüência da área;

(6) Garantia de que todas as áreas serão visitadas no máximo a relação entre o número de dias do horizonte de planejamento dividido pela mínima freqüência da área.

A figura 2.1 exibe uma visualização gráfica do modelo apresentado acima, mostrando o problema de atribuição de veículos a zonas de serviço periódico.

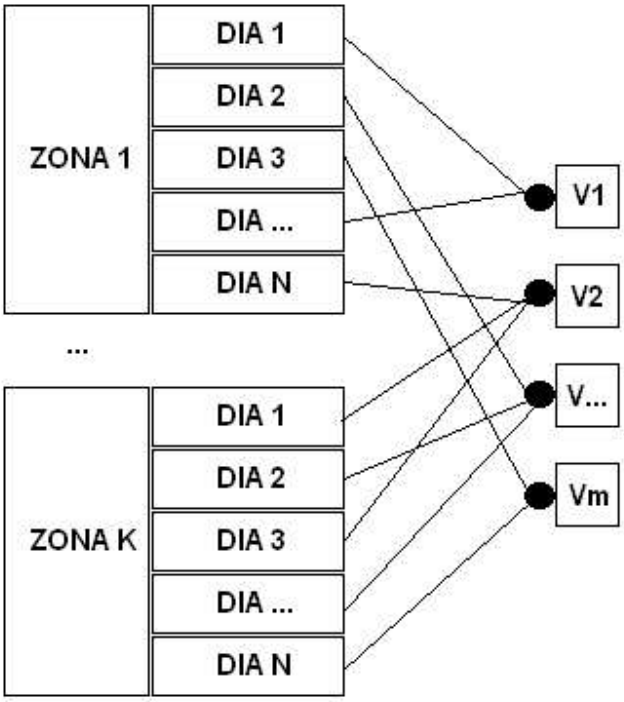


Figura 2.1: Representação do modelo (K zonas, N dias, M veículos).

2.5 – Custo Global de Atendimento - CGA

O problema acima não leva em conta os custos para o atendimento das zonas de serviço. Para que esta nova variável seja considerada, basta fazer uma alteração na função objetivo do problema, mudando a função objetivo, acrescentando-se os custos de atendimento, e passando a minimizar estes custos, conforme a função objetivo [OB3].

Custo Global de Atendimento

$$[\text{OB3}] \quad \min\left(\sum_{v=1}^{NV}\sum_{d=1}^{ND}\sum_{a=1}^{NA} C_v^{d,a} \,.\, x_v^{d,a}\right)$$

1. $C_v^{d,a}$ é o custo associado a cada área, veículo e dia para o cumprimento da atividade
2. A restrição 4 é desnecessária, porém aglutina a solução do problema, retornando as necessidades dos veículos
3. As demais restrições são necessárias

Um exemplo pode ser visualizado nas tabelas a seguir. A tabela 2.1 mostra os tempos de freqüência de atendimento para cada zona, com o campo $\mathbf{F_{min}}$ como sendo a freqüência mínima e $\mathbf{F_{max}}$ como a máxima. As tabelas 2.2, 2.3, 2.4, 2.5 e 2.6 mostram iterações entre áreas para a alocação dos veículos em determinados dias. Finalmente, na tabela 2.7 pode-se observar o resultado da escala.

Tabela 2.1: Tempos entre serviços consecutivos.

Área (zona)	**F_{min} (l^a)**	**F_{max} (u^a)**
A1	2	3
A2	2	5
A3	1	3
A4	1	5
A5	3	5

Tabela 2.2: Alocação de veículos para a área 1.

Área 1	**Dia**	**Veículo**
	1	V1
	2	
	3	
	4	V1
	5	
	6	
	7	V1
	8	

Tabela 2.3: Alocação de veículos para a área 2.

Área 2	**Dia**	**Veículo**
	1	
	2	V1
	3	
	4	
	5	V1
	6	
	7	
	8	V1

Tabela 2.4: Alocação de veículos para a área 3.

Área 3	Dia	Veículo
	1	V2
	2	
	3	V1
	4	
	5	
	6	V1
	7	
	8	V2

Tabela 2.5: Alocação de veículos para a área 4.

Área 4	Dia	Veículo
	1	
	2	V2
	3	
	4	V2
	5	
	6	V2
	7	
	8	

Tabela 2.6: Alocação de veículos para a área 5.

Área 5	Dia	Veículo
	1	
	2	
	3	V2
	4	
	5	
	6	
	7	V2
	8	

Tabela 2.7: Calendário de utilização dos veículos.

Veículo	Dia	Área
V1	1,4,7	1
	2,5,8	2
	3,6	3
V2	1,8	3
	2,4,6	4
	3,7	5

2.6 – Um Algoritmo para Formação de Escalas de Serviço

O trabalho se inicia com a definição preliminar de um algoritmo que faça uma versão simples de escala de serviços.

Por simplicidade, o algoritmo que implementamos realiza uma escala de serviço não valorada [OB1], ou seja, não leva em conta custos com a alocação, e sim a distribuição das tarefas entre zonas. Neste tipo de escalonamento não se considera qualquer tipo de informação histórica para a distribuição das atividades nas regiões, considera que se um veículo ao sair para sua região de atendimento nos dias selecionados, atende completamente a área naquele dia.

Alguns dados devem ser levados em conta para a formação da escala:

1. capacidade do caminhão;
2. quantidade de caminhões;
3. quantidade de funcionários;
4. demanda das regiões;
5. capacidade das regiões;
6. horário de atendimento;
7. dias de atendimento;
8. quantidade de viagens;
9. freqüência de visitas.

A partir destas informações, poderemos ter como saída (resultado) um calendário de atendimento onde periodicamente cada elemento de transporte visitará uma mesma região.

Nosso algoritmo preliminar considera as seguintes condições como iniciais: as escalas são seguidas (subseqüentes), ou seja, não se intercalam; está baseado em informações que influenciam na possibilidade ou impossibilidade da geração da escala de serviços, a possibilidade é a mesma para todas as áreas, não é considerado o custo das áreas, e a qualidade do resultado depende da formação inicial dos atendimentos. Uma visão deste algoritmo está na figura 2.2.

2.6.1 – Algoritmo para Escala Não Valorada - Com limitação de veículos na Frota - Algoritmo 1 [OB1]

```
[01] procedure EscalaNaoValorada; // Algoritmo 1
[02] begin
[03]   For a := 1 to Quantidade_de_zonas do
[04]   begin
[05]     while (dia < Dias_para_completar_as_viagens) do
[06]     begin
[07]         For  i := 1 to (Quantidade_de_veículos) do;
[08]          begin
[09]            Data := Data_inicial + desvio_nos_dias[a];
[10]            Ultimadata := Data;
[11]            Caminhao := caminhão_disponível;
[12]            If (caminhão < Quantidade_de_veículos) Then Incremente(caminhão_disponível)
[13]            Else caminhao := primeiro_caminhão_disponível;
[14]          end;  // for-i
[15]         dia:=dia +1;
[16]     end;  // while
[1]7     Data_inicial := Data_inicial + Ultimadata + 1;
[18]  end;  // for-a
[19] end;
```

Figura 2.2: Algoritmo guloso de escalonamento periódico em zonas para um número limitado de veículos, porém sem custo operacional por zona/dia.

Como dissemos, este algoritmo percorre as zonas seguidamente, com datas não alternadas. Se a freqüência for de 1 dia, todas as zonas serão percorridas em dias seguidos, o que poderia inviabilizar no tamanho da frota. Consideramos aqui a possibilidade de cada veículo fazer mais de uma viagem por dia a uma mesma zona. A variável **Data_inicial** é a data em que o período a ser escalonado inicia, ou seja, é o primeiro dia a ser analisado.

Toda a frota é utilizada na alocação, caso seja inicializado com os valores reais. Caso contrário, se for colocado um número de veículos menor, a alocação será somente para aquela quantidade de veículos, permitindo um remanejamento da frota.

Uma zona pode ter várias viagens para que seja totalmente atendida. Para que cada zona tenha um único veículo e uma única viagem, a capacidade dela deve ser no máximo igual à do veículo. Fazendo uma pequena alteração no algoritmo, poderemos fazê-lo realizar escalas alternadas, ou seja, em dias intercalados existirá viagens de um mesmo veículo em zonas distintas.

Na figura 2.3 temos um exemplo para o algoritmo de escalas 1.

Figura 2.3: Exemplo para o Algoritmo 1 onde 3 áreas são visitadas, cada uma após o término da anterior.

Poucas mudanças são necessárias para que o período entre áreas seja considerado de forma heterogênea, basicamente as mudanças são introduzidas na variável “desvio_nos_dias” de cada área a ser atendida.

2.6.2 – Algoritmo para Escala Não Valorada - Escalas Alternadas - Algoritmo 2 [OB1]

Se tomarmos o algoritmo 1 como referência, a variável "UltimaData" foi retirada. Ela garantia que sempre uma nova zona fosse percorrida numa data após o percorrimento total da zona anterior, no algoritmo 2 não temos isto. Nas escalas alternadas, é obrigatório que haja no mínimo um dia de folga entre as viagens de uma mesma região, ou seja, a freqüência de repetição do serviço tem que ser de pelo menos um dia. Se existirem mais de duas zonas, a freqüência de repetição do serviço de todas as zonas tem que ser pelo menos igual ao número de zonas. Na figura 2.4 pode-se observar um algoritmo para este problema.

```
[01] procedure EscalaNaoValorada; // escalas alternadas dentro em um período
[02] begin
[03]   For a := 1 to Quantidade_de_zonas do
[04]   begin
[05]     while (dia < Dias_para_completar_as_viagens) do
[06]     begin
[07]          For  i := 1 to (Quantidade_de_veículos * Máximo_de_viagens_do_caminhão_por_dia) do
[08]           begin
[09]             Data := Data_inicial + desvio_nos_dias;
[10]             Caminhao := caminhão_disponível;
[11]             If (caminhão < Quantidade_de_veículos) Then Incremente(caminhão_disponível)
[12]             Else caminhao := primeiro_caminhão_disponível;
[13]             Viagem := viagem_a_ser_realizada;
[14]             Viagem:=viagem + 1;
[15]          end;  // for-i
[16]          dia:=dia +1;
[17]          desvio_nos_dias := desvio_nos_dias + Freqüência;
[18]    end;  // while
[19]   Data_inicial := Data_inicial + 1;
[20] end;  // for-a
[21] end;
```

Figura 2.4: Algoritmo guloso para escala periódica em zonas, considerando múltiplas viagens em um mesmo dia.

Na figura 2.5 temos um exemplo para o algoritmo de escalas 2.

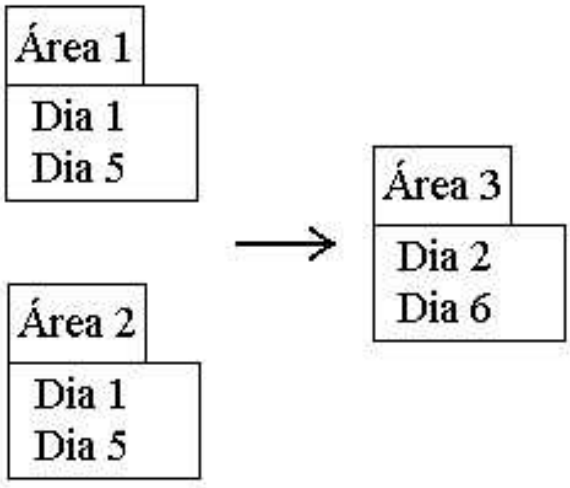


Figura 2.5: Exemplo para o Algoritmo 2 onde 3 áreas são visitadas em dias intercalados.

2.7 – Aplicações de Meta-Heurísticas

Com o objetivo de ampliar o espaço de soluções de nossos algoritmos gulosos para escala periódica de veículos, desenvolvemos versões que exploram estados de espaços de soluções bem mais significativos através das conhecidas GRASP, VNS e TABU-SEARCH, [FR95], [GL96] e [HM98]. Nosso objetivo aqui é o de procurar minimizar o custo (no. de veículos [OB2] ou custo global do serviço [OB3]) resultante da escala de serviços.

Nesse caso, o procedimento guloso anterior constrói uma escala viável a cada passada do algoritmo respectivo, porém agora levando em conta o custo operacional conforme o objetivo a ser atingido. Três algoritmos meta-heurísticos são abordados para o problema de escalonamento periódico, sendo um para cada meta-heurística.

2.7.1 – GRASP

Colocando genericamente o funcionamento do algoritmo, temos o seguinte procedimento que pode ser observado na figura 2.6:

```
[01] procedure EscalaGRASP;
[02]   // Entrada: Uma escala (v_i, p_i, z_i) viável, best := i;
[03]  begin
[04]   Criar uma lista de candidatos (LC) de possíveis escalas, LC = {( v_i, p_i, z_k), ∀ i ∈ P viável, ∀ k ∈ Z }
[05]    For (v_i, p_i, z_i) do
[06]        Trocar i por um elemento randomicamente selecionado j de LC, (v_j, p_j, z_j) ∈ LC
[07]        If P for viável Then
[08]            If C melhorar Then
[09]             begin
[10]               Manter troca - Nova situação será (v_i, p_i, z_j) e (v_j, p_j, z_i), best := j;
[11]               Esta será a nova solução e o novo custo (v_i, p_i, z_best), i := best;
[12]           end
[13]  Repeat até que o critério de parada seja atingido;
[14]  S = S ∪ {( v_i, p_i, z_best) }
[15] End; { EscalaGRASP }
```

Figura 2.6: Algoritmo GRASP para o Problema de Escalas Periódicas.

Este algoritmo implementa uma lista de candidatos (**LC**), que é construída a partir de possibilidades de escalas com a zona-período. Para cada elemento **i** da solução, será feita uma troca com um elemento **j** em seqüência da lista. Se esta troca respeitar as restrições de periodicidade (**P**) e o custo (**C**) melhorar, a troca é mantida, e repete-se o procedimento até que algum critério de parada seja atingido ou que a lista acabe. Neste caso específico, à medida que vai se construindo a solução GRASP, as escalas viáveis vão sendo geradas no conjunto final **S**. Optamos por construir desta forma, haja vista as possibilidades de seleção de soluções próximas com a mesma periodicidade desejada para um dado veículo. O procedimento pode ser amplamente modificado para que possamos encontrar soluções melhores no conjunto **S**.

2.7.2 – VNS

Colocando genericamente o funcionamento do nosso algoritmo, temos o procedimento da figura 2.7:

```
[01] procedure EscalaVNS;
[02] // Toma como solução inicial S – Conjunto das Escalas realizadas
[03] begin
[04]    Escolher randomicamente um par veículo, zona (v1, p1, z1)
[05]    Escolher randomicamente um novo par veículo, zona (v2, p2, z2)
[06]    If z1 ≠ z2 Then
[07]     Trocar z1 com z2 – Nova situação será (v1, p1, z2)  e  (v2, p2, z1)
[08]     Verificar P
[09]         If P for viável Then
[10]          Verificar C
[11]            If o custo melhorar Then
[12]               Manter troca
[13]               Esta será a nova solução e o novo custo
[14]  Repetir procedimento até que critério de parada seja atingido
[15] End; { Escala_VNS }
```

Figura 2.7: Algoritmo de VNS para busca local de soluções de escalas periódicas.

Este algoritmo implementa o VNS com uma troca apenas, onde são selecionados aleatoriamente dois veículos (v_1 e v_2), associados às suas respectivas zonas (z_1 e z_2). Caso as zonas sejam iguais, ignora-se a troca. Se as zonas forem diferentes, trocam-se os elementos e verifica-se a periodicidade (**P**). Caso **P** se mantiver, verifica-se se o custo (**C**) melhora. Se o custo melhorar, mantém-se a troca. Este processo se repete até que o critério de parada seja atingido (em geral um tempo de processamento ou o número de dias, como foi usado).

2.7.3 – Busca Tabu

A figura 2.8 exibe um pseudo-código para a meta-heurística Busca Tabu.

```
[01] procedure EscalaBuscaTabu;
[02] begin
[03]    k = 1;
[04]    Gerar solução inicial S0;
[05]    While (critério de parada não for atingido) do
[06]      Identifique zonas de uma vizinhança V;
[07]      Adicione zonas examinadas à lista tabu LT;
[08]      Trocar zonas da lista LT com zonas da iteração;
[09]      Aplique algum critério de aspiração;
[10]     Verificar P
[11]        If P for viável Then
[12]          Verificar C
[13]            If o custo melhorar Then
[14]              Manter troca
[15]     Incrementar k;
[16] end. { EscalaTabuSearch }
```

Figura 2.8: Algoritmo Busca Tabu para busca de soluções de escalas periódicas.

Este algoritmo implementa procedimento de busca tabu onde a partir de uma solução inicial $\mathbf{S_0}$ em que os elementos (veículos, periodicidade e zonas) são verificados, trocados e adicionados a uma lista tabu **LT**, irá conter elementos que não poderão mais ser trocados na iteração. É verificada a periodicidade (**P**). Caso **P** se mantiver, verifica-se o custo (**C**). Se **C** for melhor, então se mantém a troca. Na nossa implementação, foi aplicado como critério de aspiração a troca por zonas que já haviam sido trocadas anteriormente. Após um determinado número **k** de iterações, o processo pára.

2.8 –Descrição da função de custo do modelo

A função de custo empregada nas instâncias implementadas foi a seguinte: para cada dia da semana foi associado um valor, que começa em 1 e termina em 7, e inicia no domingo e termina no sábado. Se a escala levar mais de 7 dias, a função de custo para os dias da semana

repete-se até que a quantidade de dias da escala seja preenchida. Esta função pode ser vista na tabela 2.8.

Tabela 2.8: Função de custo para cada dia da semana

Dia	**Domingo**	**Segunda**	**Terça**	**Quarta**	**Quinta**	**Sexta**	**Sábado**
Custo	1	2	3	4	5	6	7

Os veículos também têm um custo associado, mas como a frota é homogênea, ou seja, todos os veículos são iguais, levou-se em conta que seus custos também serão iguais, e, portanto, poderão ser desprezados para os cálculos da escala, pois são constantes.

O custo das zonas foi elaborado de acordo com a seguinte função: para cada zona a ser visitada, seu custo será por ordem de visitação sempre começando de 1 e incrementado também de 1 para as demais zonas, até que termine o número de zonas. Quando uma zona for visitada de novo, seu custo será o mesmo da primeira visita. Esta função pode ser melhor vista na tabela 2.9.

Tabela 2.9: Função de custo para zonas

Dia	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**...**	**30**	**31**	**...**	**N**
Custo	1	2	3	4	5	6	7	8	9	...	30	31	...	n

Estes valores são padrões do sistema implementado, utilizados para que os custos sejam variados. Na implementação existe um módulo onde o usuário pode alterar qualquer valor de custo, tanto dos dias quanto para as zonas.

2.9 – Resultados

A solução inicial para cada instância encontra-se na coluna “Solução Básica”, e cada meta-heurística testada foi aplicada sobre esta solução inicial, com seus respectivos resultados nas suas respectivas colunas.

As unidades utilizadas para cada tabela foram: valores em unidades de valor, tempo em milisegundos (ms), e quantidade de operações sendo o número de operações realizadas durante o processo.

Foram utilizadas operações de trocas entre as escalas, como pode ser visto na figura 2.9. Estas trocas somente são realizadas dentro de uma mesma área.

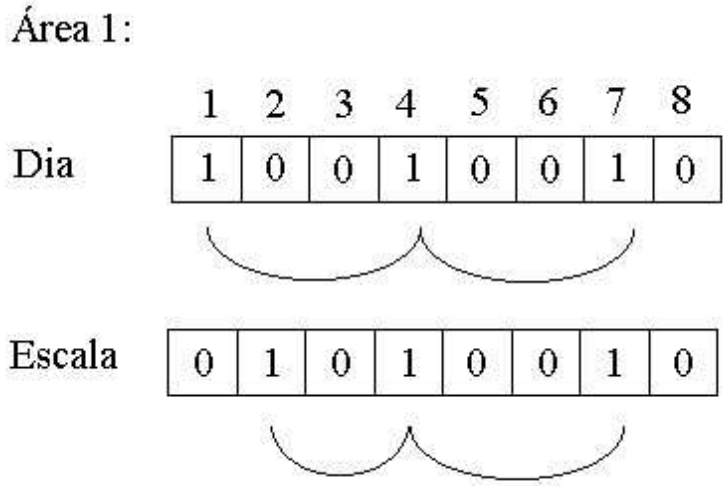


Figura 2.9: Processo de troca em uma escala.

Para o VNS, foi utilizada somente uma troca nas operações, e como critério de parada uma quantidade de iterações, que foi igual a quantidade de dias do período de trabalho a ser escalonado.

Para o GRASP, a quantidade de elementos da lista de candidatos foi igual a quantidade de zonas do problema. Como critério de parada foi utilizada uma quantidade de operações igual ao número de dias do problema. Também foi utilizada somente uma troca nas operações.

Para a BUSCA TABU, o critério de aspiração utilizado foi trocar elementos que já tinham sido trocados (zonas) anteriormente por elementos, e verificar se melhorava a solução. O critério de parada foi uma quantidade de operações igual ao número de dias do problema. Também foi utilizada somente uma troca nas operações.

Os testes foram compostos por instâncias que variavam de 1 mês (30 dias) até 1 ano (365 dias), mas a maioria foram instâncias de 3 meses, 6 meses e 1 ano. A quantidade de zonas, veículos e freqüência de repetição foram variadas. Seus resultados podem ser vistos nas tabelas 2.10, 2.11, e 2.12.

O testes foram feitos num computador PC, com processador AMD Athlon 1.2 Ghz, com 256 Mb de memória RAM, e com Windows 98 e na ferramenta Borland Delphi 5.0, [CANTU99].

Tabela 2.10: Resultados de valores da função objetivo (unidades de valor)

Tempo	**Zonas**	**Veículos**	**Freqüência**	**Sol. Básica**	**Tabu Search**	**GRASP**	**VNS**
1 mês	**10**	**5**	**2**	3285	3260	3260	3260
2 meses	**10**	**5**	**2**	6365	6340	6340	6340
3 meses	**20**	**8**	**5**	15174	14888	14886	14893
3 meses	**20**	**15**	**5**	15060	14760	14760	14760
3 meses	**100**	**30**	**5**	366000	358540	358500	358548
3 meses	**100**	**50**	**5**	368600	358768	358600	358704
3 meses	**80**	**40**	**6**	193580	192005	191980	191994
3 meses	**90**	**50**	**5**	297930	289930	289998	290010
4 meses	**20**	**10**	**5**	20205	19905	19905	19905
5 meses	**30**	**10**	**5**	55270	54470	54470	54470
6 meses	**30**	**15**	**5**	66840	66615	66615	66615
6 meses	**100**	**50**	**5**	725925	723435	723425	723433
6 meses	**80**	**50**	**4**	581820	581820	581820	581820
6 meses	**90**	**40**	**5**	589715	586920	586915	586920
6 meses	**50**	**40**	**4**	230230	230230	230230	230230
6 meses	**120**	**70**	**5**	1042955	1039460	1039455	1039458
1 ano	**20**	**10**	**5**	61365	61065	61065	61065
1 ano	**30**	**10**	**5**	135060	134760	134760	134760
1 ano	**50**	**20**	**5**	371125	369925	369925	369925
1 ano	**100**	**50**	**5**	1475825	1468328	1468325	1468328
1 ano	**110**	**60**	**7**	1174680	1018992	1018680	1018784

Tabela 2.11: Resultados de tempo (milissegundos)

Tempo	**Zonas**	**Veículos**	**Freqüência**	**Sol. Básica**	**Tabu Search**	**GRASP**	**VNS**
1 mês	**10**	**5**	**2**	14	27	94	17
2 meses	**10**	**5**	**2**	30	61	41	44
3 meses	**20**	**8**	**5**	50	89	747	61
3 meses	**20**	**15**	**5**	38	112	753	59
3 meses	**100**	**30**	**5**	254	3867	187887	2022
3 meses	**100**	**50**	**5**	180	4047	188182	1999
3 meses	**80**	**40**	**6**	169	1492	56240	827
3 meses	**90**	**50**	**5**	206	2957	124861	1524
4 meses	**20**	**10**	**5**	69	150	1221	91
5 meses	**30**	**10**	**5**	126	499	6451	277
6 meses	**30**	**15**	**5**	144	660	9069	377
6 meses	**100**	**50**	**5**	508	22295	1098915	11290
6 meses	**80**	**50**	**4**	388	19213	761576	10263
6 meses	**90**	**40**	**5**	210	12468	518746	5898
6 meses	**50**	**40**	**4**	280	4206	94496	2013
6 meses	**120**	**70**	**5**	391	26599	1535678	13009

1 ano	**20**	**10**	**5**	227	1012	9394	570
1 ano	**30**	**10**	**5**	289	2784	38314	1505
1 ano	**50**	**20**	**5**	433	13611	327989	7060
1 ano	**100**	**50**	**5**	1016	103496	4619155	48132
1 ano	**110**	**60**	**7**	371	47549	2895283	23584

Tabela 2.12: Quantidade de iterações (operações realizadas)

Tempo	**Zonas**	**Veículos**	**Freqüência**	**Sol. Básica**	**Tabu Search**	**GRASP**	**VNS**
1 mês	**10**	**5**	**2**	150	300	1500	150
2 meses	**10**	**5**	**2**	290	580	290	290
3 meses	**20**	**8**	**5**	360	720	7200	360
3 meses	**20**	**15**	**5**	360	720	7200	360
3 meses	**100**	**30**	**5**	1800	3600	180000	1800
3 meses	**100**	**50**	**5**	1800	3600	180000	1800
3 meses	**80**	**40**	**6**	1200	2400	96000	1200
3 meses	**90**	**50**	**5**	1620	3240	145800	1620
4 meses	**20**	**10**	**5**	480	960	9600	480
5 meses	**30**	**10**	**5**	900	1800	27000	900
6 meses	**30**	**15**	**5**	1080	2160	32400	1080
6 meses	**100**	**50**	**5**	3600	7200	360000	3600
6 meses	**80**	**50**	**4**	3600	7200	288000	3600
6 meses	**90**	**40**	**5**	3240	6480	291600	3240
6 meses	**50**	**40**	**4**	2250	4500	112500	2250
6 meses	**120**	**70**	**5**	4320	8640	518400	4320
1 ano	**20**	**10**	**5**	1460	2920	29200	1460
1 ano	**30**	**10**	**5**	2190	4380	65700	2190
1 ano	**50**	**20**	**5**	3650	7300	182500	3650
1 ano	**100**	**50**	**5**	7300	14600	730000	7300
1 ano	**110**	**60**	**7**	5720	11440	629200	5720

Os resultados foram obtidos rapidamente para a solução básica e VNS. Instâncias para a busca tabu demoraram um pouco mais, e para o GRASP elas levaram os maiores tempos. Porém, os melhores resultados foram obtidos através do GRASP.

Em geral, instâncias de pequeno porte (1 e 2 meses) quase não conseguiam melhorias com a aplicação das meta-heurísticas. À medida que o espaço de tempo aumentava, as melhorias tornavam-se mais aparentes, porém o tempo para conseguí-las era bem maior.

Um ponto que foi notado nos resultados foi a influência da freqüência de repetição. À medida que ela aumentava, a solução do problema tinha um custo bem menor. Para comprovar esta observação, foi feito uma bateria de testes com a seguinte descrição: manteve-se fixo os

valores do tempo, zonas e veículos, e variou-se a freqüência. Então se calculou a solução básica e as meta-heurísticas, como se pode observar nas tabelas 2.13 e 2.14. A partir dos resultados, construiu-se um gráfico para mostrar a influência da variação da freqüência nos custos, como se pode verificar na figura 2.10.

Tabela 2.13: Resultados de valores da função objetivo para uma instância fixa de 3 meses com variação da freqüência

Tempo	**Zonas**	**Veículos**	**Freqüência**	**Sol. Básica**	**Tabu Search**	**GRASP**	**VNS**
3 meses	**30**	**20**	**2**	83820	83820	83820	83820
3 meses	**30**	**20**	**4**	42180	42180	42180	42180
3 meses	**30**	**20**	**6**	27690	27491	27490	27490
3 meses	**30**	**20**	**8**	23880	23880	23880	23880
3 meses	**30**	**20**	**10**	17715	17323	17315	17321
3 meses	**30**	**20**	**12**	14670	14473	14470	14487

Tabela 2.14: Resultados de valores da função objetivo para uma instância fixa de 6 meses com variação da freqüência

Tempo	**Zonas**	**Veículos**	**Freqüência**	**Sol. Básica**	**Tabu Search**	**Grasp**	**VNS**
6 meses	**30**	**20**	**2**	168540	168540	168540	168540
6 meses	**30**	**20**	**4**	83610	83610	83610	83610
6 meses	**30**	**20**	**6**	54915	54515	54515	54515
6 meses	**30**	**20**	**8**	42825	42425	42425	42427
6 meses	**30**	**20**	**10**	34500	33700	33700	33700
6 meses	**30**	**20**	**12**	27690	27492	27490	27492

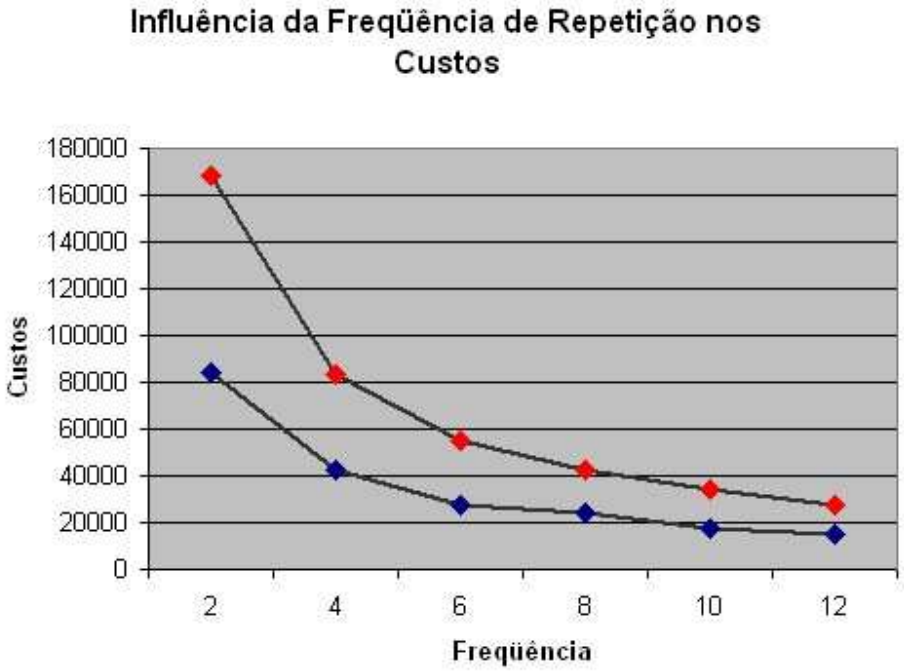


Figura 2.10: Influência da variação da freqüência nos custos.

2.10 – Comentários

Vários trabalhos podem ser desenvolvidos a partir do modelo aqui proposto e dos algoritmos aqui sugeridos. Pode-se desenvolver um modelo que suporte vários veículos percorrendo várias áreas num mesmo dia, e um veículo podendo não percorrer totalmente uma área num dia, e ter que completar sua tarefa em outro dia (pré-emptivo).

Um trabalho seria a utilização de outras meta-heurísticas para a verificação de melhorias na solução. Um estudo mais ampliado para avaliação dos limites inferiores seria outro ponto relevante, de modo a comparar eficiência dos métodos aqui propostos.

O desenvolvimento de um algoritmo que trabalhe com atividades preemptivas, ou seja, onde se possa interromper a sua execução e atender a outro serviço, ou dinamicamente combinar estas atividades, fazendo um escalonamento dinâmico, seria um trabalho mais complexo que poderia ser realizado, podendo render frutos importantes em áreas como no combate ao dengue.

Foi criado um modelo matemático para o Problema de Escalonamento Periódico de Veículos em Zonas (PEPVZ), bem como a utilização de meta-heurísticas para a melhoria dos resultados (VNS, GRASP, TABU-SEARCH).

Como são realizadas comumente, as escalas de serviço são uma dura tarefa para quem as realizam manualmente, e visivelmente os resultados não são bons, diferentemente de uma implementação computacional mais elaborada, onde a grande vantagem é a experimentação de variáveis, sendo a análise dos resultados bem mais precisa e rápida.

Preferimos não explorar um número de iterações maior nos procedimentos, pois verificamos que o espaço das trocas era muito limitado, porém seria um trabalho interessante avaliar o efeito do aumento do número de iterações e possivelmente trocas diferentes de 1, nas zonas. Outra situação interessante a ser explorada seria o efeito da heterogeneidade no processo de escala. Como nossos testes foram apenas para avaliar o desempenho das meta-heurísticas no contexto de instâncias testes, nós não nos preocupamos com maiores detalhamentos do processamento.

3. O Problema do Carteiro Rural

3.1 – Introdução

Os problemas de otimização combinatória são comumente problemas que requerem muito esforço computacional para a geração de soluções ótimas.

Roteamento de veículos é um problema que já foi trabalhado por vários pesquisadores, mas na maioria dos modelos, sua complexidade e esforço computacional são muito altos.

Neste capítulo procuraremos uma solução para o Problema do Carteiro Rural Misto (PCRM), e aplicaremos uma meta-heurística para a tentativa de melhoria da solução em uma das fases do mesmo.

3.2 – Estado da arte

Christofides (1981) propôs um algoritmo de *branch-and-bound* com relaxação lagrangeana para a resolução do PCR simétrico, e Corberan et al (1986) para o PCR assimétrico, [CCCM81], [CCCM86]. Corberán e Sanchis (1994) propuseram uma solução utilizando *branch-and-cut* em [CS94] e [SAN90], com uma performance melhor que a proposta de Christofides. Ghiani e Laporte também utilizaram *branch-and-cut* para resolver o PCR simétrico, [GL00]. Coberán, Marti e Romero (2000) desenvolveram um algoritmo baseado em busca tabu para a resolução do PCRM, [CMR00]. Romero (1997) e Corberán, Romero e Sanchis (2000), propuseram heurísticas para a resolução do PCRM, [ROM97], [CRS00].

Vários outros trabalhos sobre o PCRM utilizando heurísticas foram publicados. Corberán et al (1998), desenvolveram um procedimento que manipula o grafo e aplica fluxo de custo mínimo para resolver o PCRM, [CMR98]. Outras heurísticas podem ser encontradas em [HLN99] e [PW95].

Uma aplicação do PCR sobre corte de peças irregulares se encontra Soeiro et al (2002) e utiliza algoritmos meméticos para a solução, [FRM02].

Diversas aplicações do PCR existem na prática: coleta de lixo, percurso do fumacê, varredores de rua, entrega de correspondência, percurso de tapa valas, percurso de leituristas de energia e gás, percurso de ônibus escolares, remoção de neve, [EGL95].

3.3 – Proposta de um novo algoritmo para o PCRM

Como obrigar a passagem do veículo por um determinado arco, se não houver pelo menos uma unidade de fluxo? E como evitar que o veículo ande mais do que precisa para finalizar seu percurso?

Apesar de já existirem propostas de algoritmos meta-heurísticos (baseado em busca tabu) para o PCRM, testados e com bons limites inferiores, nossa investigação recai sobre um procedimento alternativo, e de complexidade mais baixa que permita encontrar resultados próximos ou melhores aos atingidos pelos melhores algoritmos desenvolvidos. Além disto, desejamos explorar uma combinação de métodos que permitem gerar soluções de boa qualidade para o PCRM, e que possa ser facilmente ampliado para outras instâncias de problemas do Carteiro Rural (Simétrico e Orientado, por exemplo), dando ao método mais flexibilidade, assim como a possibilidade de torná-lo um método possivelmente exato haja vista a formulação embutida no método.

Laporte (1999) desenvolveu uma conjectura de que o GTSP assimétrico poderia ser usado para resolver instâncias do PCR, para os casos simétrico e misto. Considerando algumas instâncias, ele usou o seu método B&B e mostrou que o GTSP sozinho poderia resolver várias instâncias do PCR simétrico e misto. Porém muitas não foram resolvidas na otimalidade, em razão das topologias e densidades de elos existentes nas redes testes. Baseado nesta dificuldade, formalizamos neste trabalho o nosso algoritmo para o PCRM, [LAP97].

Em nosso algoritmo propomos, inicialmente, desenvolver um processo de encontrar o número de grupos de ligações requeridas, componentes R-Conexas.

Temos que definir uma origem para a instancia do problema Tendo a origem definida, realizaremos algum procedimento que oriente o grafo, ou seja, que deixe o grafo totalmente orientado, sem elos.

Como os arcos requeridos geralmente são localizados numa região, fica claro e visível identificar grupos de arcos requeridos. Então, genericamente, uma forma de resolver o PCRM seria gerar um percurso do ponto de origem até estes grupos de arcos requeridos. A maior concentração de passagens que o veículo daria seria justamente na região dos grupos. Isto não impede de se ter um grafo formado totalmente por arcos requeridos, ou diversos grupos com poucos arcos espalhados por todo o grafo. Esta abordagem inicial é apenas para se obter um ponto de partida para a resolução do PCRM.

Para a resolução deste problema inicial, existe uma variação do PCV que calcula o menor percurso entre grupos de vértices e retorna ao ponto de origem, que é o GTSP como vimos no capítulo 2 deste trabalho. Aplicaremos então o GTSP ao nosso problema, e obteremos um percurso inicial que conecta todos os grupos de arcos requeridos.

Este percurso inicial já poderia ser uma solução para o PCRM se passasse por todos os arcos requeridos, mas o GTSP chega a um vértice de cada grupo, e possivelmente não passará por todos os arcos requeridos. Então aplicaremos um algoritmo de fluxo de custo mínimo ao grafo, para que haja uma circulação de fluxo em todos os arcos requeridos do grafo.

Mesmo aplicando o fluxo mínimo, não há garantia de que todos os arcos requeridos sejam percorridos. Então, logo após o cálculo do GTSP, associa-se a todos os arcos que fazem parte do percurso gerado pelo GTSP uma unidade de fluxo, para garantir que cada arco seja percorrido pelo menos uma vez. Seguindo este mesmo raciocínio, cada arco requerido também deve ter uma unidade de fluxo.

Na orientação do grafo, consideramos como requeridos os elos e arcos das R-Componentes Conexas, e a orientação assumida pela solução do GTSP, entre os grupos de componentes.

Agora, com os arcos possuindo unidades de fluxo, aplicamos o algoritmo de fluxo de custo mínimo, e garantiremos a passagem por todos os arcos requeridos. Obteremos, então, um grafo com tantas unidades de fluxo em cada arco que sejam necessárias para a geração de um percurso que saia do vértice de partida, passe por todos os arcos requeridos e retorne ao ponto de partida.

3.3.1 – Descrição do algoritmo

O grafo inicialmente tem que ser conexo. Escolhe-se uma origem e os arcos cuja passagem é obrigatória. O problema irá gerar um percurso que parte da origem, passa por todos os arcos requeridos e arcos de percurso que forem necessários para o caminho, e retorna à origem do problema.

Inicialmente utiliza-se algum procedimento que identifique relações de equivalência para encontrar vértices que tenham arcos requeridos em comum. Cada relação de equivalência será um grupo distinto de arcos requeridos. Caso a origem não esteja em nenhum grupo encontrado, cria-se um novo grupo contendo somente o vértice origem.

Em seguida é feita uma transformação no grafo original, para que todos os elos sejam orientados. Os elos serão orientados da seguinte maneira: se o elo não é requerido, troca-se o elo por dois arcos, um em cada sentido. Mas se o elo for requerido, aplicaremos a transformação de Sherafat, [SH88].

Para cada grupo de arcos requeridos encontrados, selecionamos um vértice que pertença a um dos arcos do grupo, para conexão entre os grupos. Será aplicado um GTSP entre estes grupos, de forma que os arcos requeridos irão se comunicar através de um percurso orientado. Neste ponto, existirá um percurso que ligará um ponto de cada grupo entre si, numa determinada seqüência, e um destes pontos será a origem do problema. Uma maneira de melhorar o algoritmo é aplicar alguma meta-heurística para melhoria do percurso gerado pelo GTSP, pois como o percurso gerado vai fazer parte da solução, quanto menor o percurso, uma probabilidade menor de arcos vai fazer parte da solução.

Após a conexão entre os grupos, coloca-se uma unidade de fluxo em cada arco pertencente ao percurso de conexão dos grupos, ou seja, aos arcos resultantes do GTSP. Neste ponto, todos os arcos requeridos e o caminho entre os grupos possuem fluxo mínimo igual a uma unidade.

Alguns arcos requeridos possivelmente não estarão na solução do GTSP. Então, coloca-se também em cada arco requerido uma unidade de fluxo, para garantir pelo menos uma passagem por este arco. Calcula-se o fluxo máximo de custo mínimo no grafo, com o objetivo de se encontrar um fluxo viável. Se a solução do fluxo for viável, teremos uma solução para o PCRM. O algoritmo de fluxo irá fazer com que haja uma circulação de

unidades de fluxo em determinados arcos do grafo, inclusive nos arcos requeridos e arcos de solução do GTSP.

Após a aplicação do fluxo mínimo, pode ser que sejam formados fluxos triangulares nos arcos que eram elos originalmente, ou seja, que em alguns arcos o fluxo foi e voltou pelo próprio arco sem antes passar por outro (não atravessou o arco). Então teremos que aplicar algum procedimento que verifique a formação de fluxos triangulares e resolva este problema.

Para resolver o problema dos fluxos triangulares, após a aplicação do algoritmo de fluxo de custo mínimo aplicaremos uma heurística que forçadamente irá direcionar um elo que tinha sido orientado segundo Sherafat, e que provocou um fluxo triangular. Após este direcionamento, calculamos novamente o fluxo mínimo e repetimos o processo até que não haja mais fluxos triangulares. Não obrigatoriamente iremos orientar todos os elos. Pode ser que somente um direcionamento já resolva o problema e não gere mais nenhum fluxo triangular. Porém, pode ser que todos os elos sejam orientados e o problema não tenha solução.

Obteremos assim um sub-grafo resultante do GTSP e do fluxo, que é conexo e por onde o percurso do PCRM irá passar. Neste ponto, poderemos gerar várias soluções a partir deste sub-grafo.

3.3.2 – Algoritmo

O algoritmo que desenvolvemos para o PCRM pode ser resumido da seguinte forma:

Algoritmo PCRM↔GTSP

Passo 1 : Identificação dos grupos
Identificar grupos de vértices das arestas requeridas, através de relações de equivalência.
Caso a origem do problema não esteja em nenhum grupo, considerar o vértice origem como um novo grupo, caso contrário, o vértice origem é o elemento de partida do grupo ao qual pertence.

Passo 2 : Cálculo do GTSP
Para cada grupo encontrado no Passo 1, selecionar aleatoriamente um vértice.
Calcular o GTSP entre eles, para a conexão dos grupos de arcos requeridos.

Passo 3 : Orientação do grafo
Orientar um grafo misto, de acordo com as seguintes regras:

1. Se o elo for requerido então
   Utilize a transformação de Sherafat, colocando como requerido o arco central
   Senão
      Se o elo for do percurso do GTSP Então Troque o elo existente por dois arcos, um em cada sentido, sendo incluído no arco no sentido do caminho do GTSP como requerido.
2. Incluir nos limites inferiores de fluxo dos arcos requeridos uma unidade de fluxo.

Passo 4 : Fluxo máximo de custo mínimo
Calcular o fluxo máximo de custo mínimo sobre o grafo orientado.
Se não houver geração de fluxos triangulares então
    Solução para o PCRM sob a forma de um multigrafo
Senão
    Repetir para cada elo requerido até que não exista mais fluxo triangular
        Selecionar um elo que foi orientado conforme Sherafat
        Orientar forçadamente este elo em qualquer sentido
        Calcular o fluxo de custo mínimo sobre o grafo
        Verificar a existência de fluxo triangular

Apesar de o algoritmo B&B para o PCV nortear a complexidade do processo acima proposto, se utilizarmos boas heurísticas e meta-heurísticas, o nosso procedimento poderá ter uma ordem não superior a $O(n^3)$.

Se desejarmos generalizar o nosso procedimento para o caso simétrico do PCR, podemos então rever o processo e reescrevê-lo como segue. A complexidade também não muda neste caso:

Algoritmo PCRS↔GTSP

Passo 1 : Identificação dos grupos
Identificar grupos de vértices das arestas requeridas, através de relações de equivalência.
Caso a origem do problema não esteja em nenhum grupo, considerar o vértice origem como um novo grupo, caso contrário, o vértice origem é o elemento de partida do grupo ao qual pertence.

Passo 2 : Cálculo do GTSP
Para cada grupo encontrado no Passo 1, selecionar aleatoriamente um vértice.
Calcular o GTSP entre eles, para a conexão dos grupos de elos requeridos.

Passo 3 : Conexão das R-Componentes do Grafo
Conectar as R-Componentes do Grafo, através do percurso gerado pela solução do GTSP, indicando como requerido todos os elos do caminho. Um sub-grafo com uma-componente (R=1) está gerado.

Passo 4 : Aplicar o Algoritmo do Carteiro Chinês

Sobre o Grafo do passo anterior, aplicar o algoritmo do Carteiro Chinês Simétrico de Edmonds & Johnson (1973).

3.3.3 – Exemplo de Solução

Inicialmente criamos um grafo no sistema XNês, utilizando as operações de edição de grafos, nele existentes, [MAEG99]. Colocamos também custos nas arestas. No módulo de seleção, escolhemos os elos e arcos que desejamos percorrer obrigatoriamente. O grafo preparado original pode ser visto na figura 3.1.

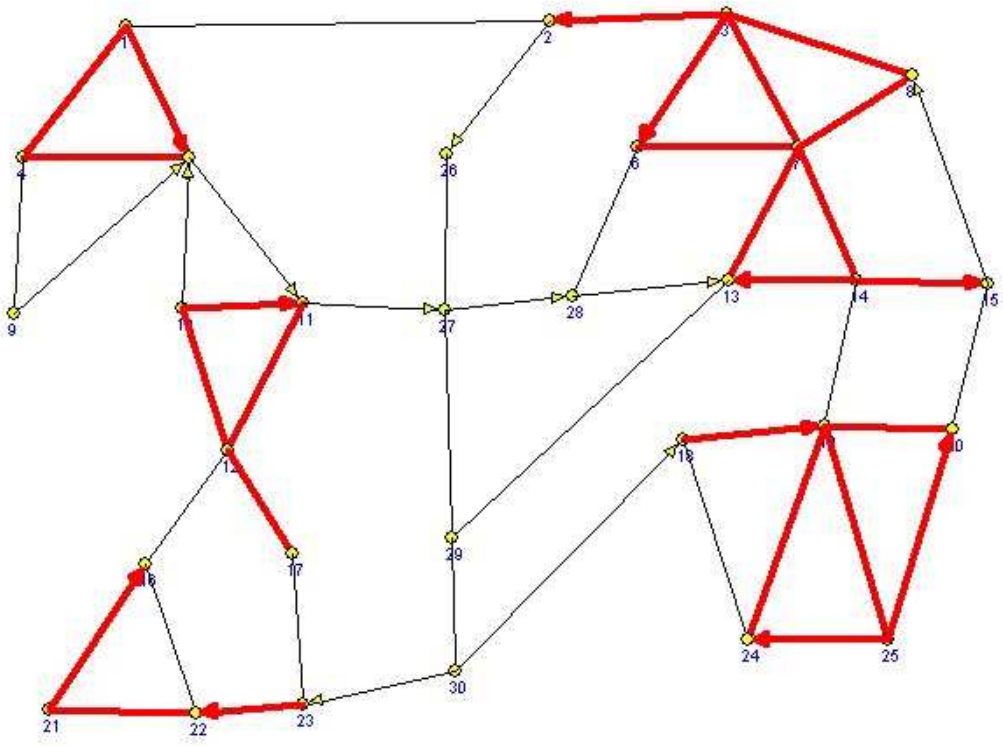


Figura 3.1: Grafo misto

Em seguida, identificamos grupos entre os arcos requeridos e selecionamos vértices de cada grupo para o calculo do GTSP. Caso a origem do problema não esteja em nenhum grupo, ela será considerada como um grupo. A figura 3.2 mostra o resultado da solução do GTSP, que obteve o custo de 438.

A partir do grafo misto, aplicamos um procedimento de orientação, seguindo a transformação de Sherafat, trocando os elos não requeridos por dois arcos, um em cada sentido, e os elos requeridos por uma pequena estrutura de arcos, e as ligações do percurso do GTSP não requeridas, as transformamos em requeridas e aplicamos uma unidade de fluxo na orientação definida pela solução do GTSP.

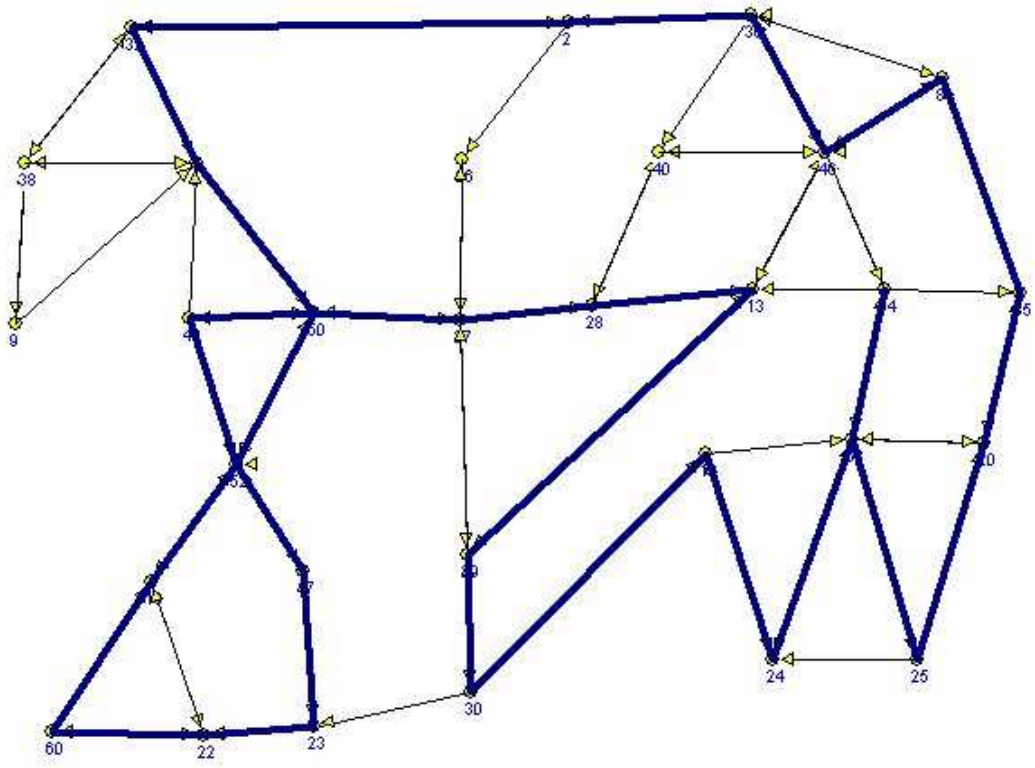


Figura 3.2: Solução do GTSP.

Neste momento, temos um circuito entre determinados vértices do grafo, que irá conectar os grupos de arcos requeridos identificados. Alguns arcos requeridos podem fazer parte do caminho gerado por este circuito, mas outros não.

Aplicaremos agora sobre o grafo um algoritmo de fluxo de custo mínimo, para garantir a passagem de fluxo nos arcos que não foram percorridos pelo GTSP. Os arcos que formaram o caminho do GTSP serão modificados para que pelo menos uma unidade de fluxo passe por eles, fazendo com que se tornem também arcos requeridos.

A figura 3.3 exibe a solução final do PCRM com custo final igual a 972.

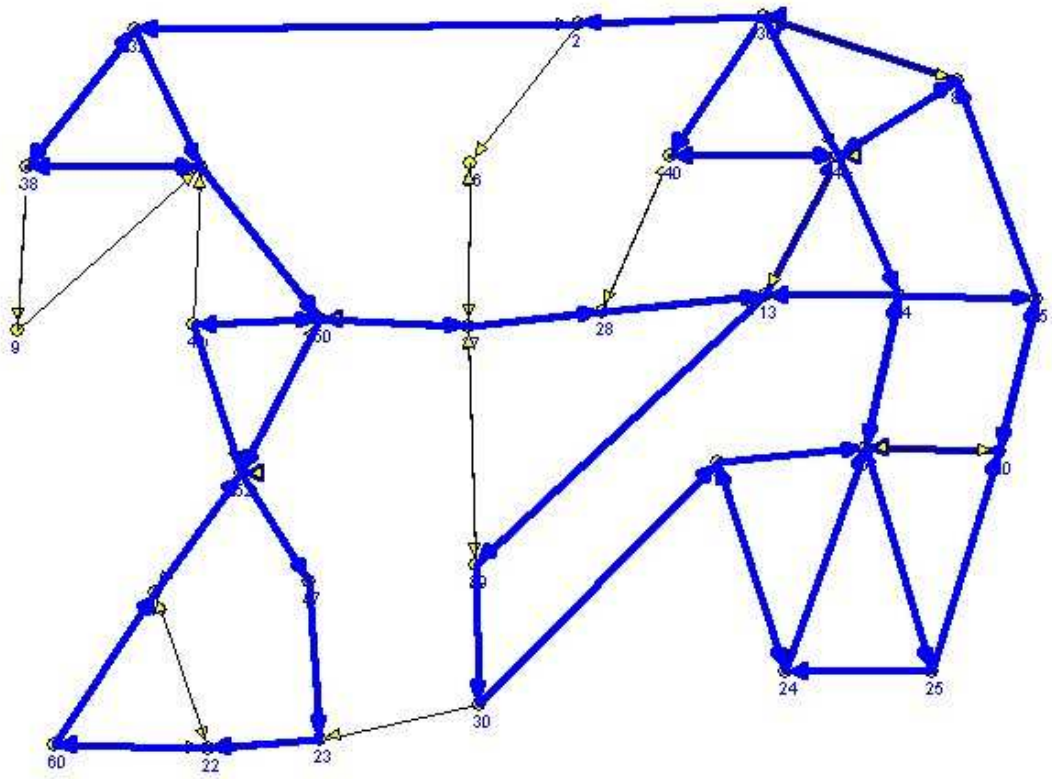


Figura 3.3: Solução do PCRM.

Finalmente, na figuras 3.4 temos o sub-grafo solução do PCR. Eles mostram apenas os arcos necessários para a geração de possíveis percursos do PCR.

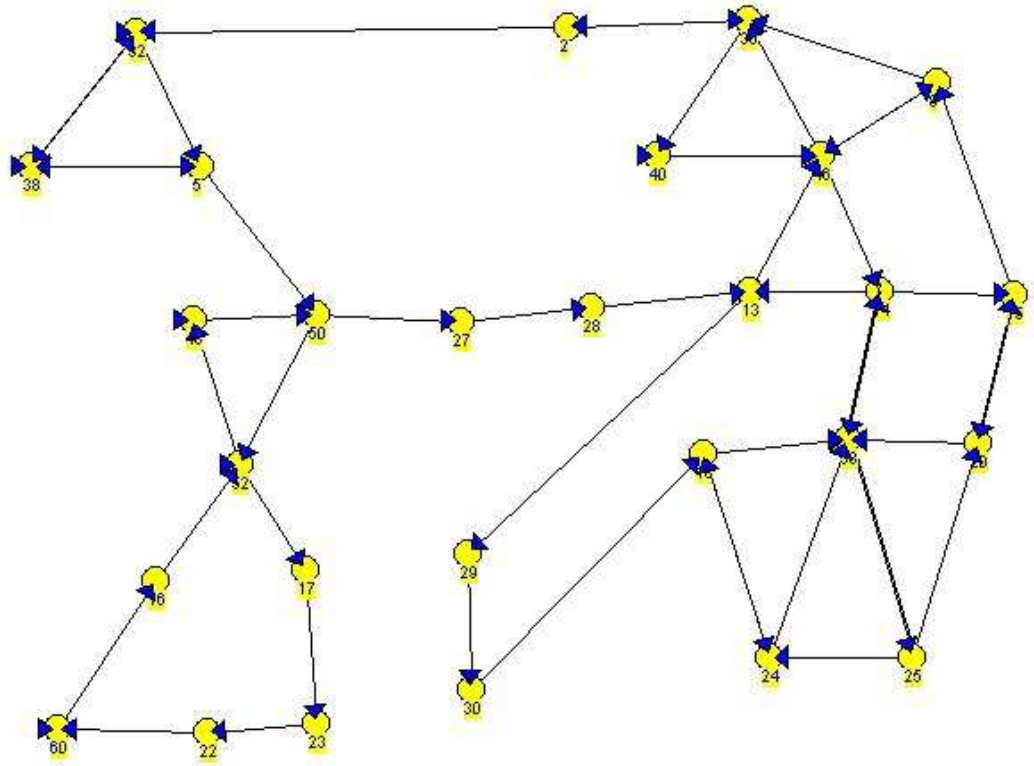


Figura 3.4: Sub-grafo solução do PCRM.

3.4 – Descrição dos Testes

Foram geradas várias instâncias de grafos, de diversos tamanhos para a bateria de testes, além de avaliadas instâncias da literatura gentilmente fornecidas pelo professor Angel Corberán. O testes foram feitos num computador PC, com processador AMD Athlon 1.2 Ghz, com 256 Mb de memória RAM, e com Windows 98 e na ferramenta Borland Delphi 5.0.

Foi implementado no sistema XNês para a resolução do PCRM um algoritmo de GTSP baseado em uma implementação dinâmica que pode ser encontrada em [RB96]. Para refinar a solução utilizamos um algoritmo de caixeiro viajante com *branch-and-bound*, e PDM, que podem ser ambos encontrados em [SDK83]. Foi aplicado também um algoritmo de fluxo de custo mínimo, o *out-of-kilter*, que pode ser encontrado em [AMO93].

Como solução do PCRM, obteve-se um subgrafo que continha somente os arcos necessários para a geração de um percurso para o PCRM.

Instâncias de grafos de tamanhos variados foram construídas, com quantidade de nós de 100 a 500 vértices. A quantidade de arcos e elos foi em torno de 100 a 500, e a de elos e arcos requeridos foi em torno de 10 a 500. Várias combinações foram utilizadas, de modo a observar o efeito do número de elos no algoritmo assim como o aumento do número de grupos.

A tabela 3.1 mostra a descrição das instâncias de grafos utilizadas na bateria de testes que criamos, indicando o nome, quantidade de vértices, elos e arcos, e o custo do sub-grafo solução quando aplicado a solução inicial e a heurística de tentativa de melhoria.

Foram testadas cerca de 100 instâncias fornecidas gentilmente pelo professor Angel Corberán, as quais podem ser encontradas em [CMR98] e [CRS01]. Estas instâncias foram divididas em dez grupos pelos autores, cada um com mais ou menos uma determinada quantidade de R-Componentes conexas. O nome da instância identifica a quantidade de R-componentes (quantidade de grupos), pois todas começavam com GXX, onde XX é o grupo o qual a instância pertence e a quantidade de componentes R-conexas do grafo. Para rodar estas instâncias no sistema XNÊS, foi necessário realizar uma transformação nos multigrafos fornecidos, de modo a manter a instância original (custos, ligações e no. de grupos) assim como gerar posições de localização, de modo a visualizar o grafo resultante. Neste caso, preferimos construir os grafos onde os vértices ocupam posições em um círculo, e as arestas correspondem aos elos e arcos do conjunto. Nas ligações repetidas, introduzimos um novo vértice e uma ligação de custo zero, de modo a manter o perímetro original e as ligações requeridas do conjunto.

Aplicamos então nosso algoritmo e obtivemos resultados de custo, tempo e quantidade de grupos formados, que podem ser vistas com maiores detalhes no anexo sob a forma de tabelas e gráficos.

Também calculamos a distância que nossa solução está do limite inferior apontado por Corberan & Sanchis (2000). Estes limites inferiores são resultados teóricos, que são aplicados sobre instâncias de um problema, ou seja, as instâncias são submetidas a um modelo teórico que retorna uma estimativa do valor do custo da melhor solução possível para a instância em análise, [CRS01].

As tabelas são compostas pelas colunas Instância, Custo PCR, Tempo PCR, Grupos PCR, Dist. PCR LI, LI, Custo C, Tempo C, Grupos C, Dist. C LI, que significam,

respectivamente, o nome do grafo utilizado, o custo, tempo, número de grupos encontrados e distância do limite inferior para nosso algoritmo, o valor do limite inferior, e o custo, tempo, número de grupos encontrados e distância do limite inferior para o algoritmo de Corberán.

Os gráficos do anexo comparam o custo e o tempo da solução do PCRM utilizando, o método de Corberán e nosso método. Eles possuem no eixo X as instâncias de cada grupo, e no eixo Y os custos e o tempo.

3.5 – Resultados

A tabela 3.1 mostra soluções de instâncias de grafos criados por nós. Todos os resultados tiveram como origem o vértice de número 1 e todas as arestas possuem custo igual a 1. Na tabela 3.2 pode-se observar as mesmas instâncias, mas com custos iguais à distância euclidiana entre os pontos. Os campos das tabelas são: Instância, com o nome do grafo, V, a quantidade de vértices, E, a quantidade de elos, A, a quantidade de arcos, E_R, a quantidade de elos requeridos, A_R, a quantidade de arcos requeridos, G_R, a quantidade real de grupos, G_C, a quantidade de grupos calculados, Custo, o custo da solução, e Tempo, o tempo da solução do problema.

Tabela 3.1: Instâncias de grafos com arestas de custos unitários.

Instância	**V**	**E**	**A**	**E_R**	**A_R**	**G_R**	**G_C**	**Custo**	**Tempo**
100_0	100	112	0	108	0	4	4	178	0,8
100_10	100	112	10	108	10	4	4	166	0,8
100_50	100	112	50	108	50	4	4	200	0,9
100_80	100	112	80	108	80	4	4	234	0,9
100_100	100	112	100	108	100	4	4	263	0,101
200_0	200	240	0	218	0	8	8	380	0,19
200_20	200	240	20	218	20	8	8	362	0,2
200_50	200	240	50	218	50	8	8	381	0,2
200_100	200	240	100	218	100	8	8	411	0,25
200_150	200	240	150	218	150	8	8	454	0,21
300_0	300	352	0	330	0	12	12	539	0,3
300_30	300	352	30	330	30	12	12	530	0,36
300_70	300	352	70	330	70	12	12	582	0,38
300_120	300	352	120	330	120	12	12	602	0,521
300_200	300	352	200	330	200	12	12	669	0,521
400_0	400	471	0	443	0	16	16	737	0,782
400_50	400	471	50	443	50	16	16	751	0,631

400_100	400	471	100	443	100	16	16	769	0,731
400_200	400	471	200	443	200	16	16	836	0,991
400_300	400	471	300	443	300	16	16	924	1,62
500_0	500	584	0	549	0	20	20	926	2,624
500_100	500	584	100	549	100	20	20	951	0,901
500_200	500	584	200	549	200	20	20	1026	0,841
500_300	500	584	300	549	300	20	20	1120	0,871
500_400	500	584	400	549	400	20	20	1184	0,881

Tabela 3.2: Instâncias de grafos com arestas de custos euclidianos.

Instância	**V**	**E**	**A**	$\mathbf{E_R}$	$\mathbf{A_R}$	$\mathbf{G_R}$	$\mathbf{G_C}$	**Custo**	**Tempo**
100_0	100	112	0	108	0	4	4	6677	0,8
100_10	100	112	10	108	10	4	4	6186	0,9
100_50	100	112	50	108	50	4	4	7651	0,9
100_80	100	112	80	108	80	4	4	9108	0,11
100_100	100	112	100	108	100	4	4	10179	0,12
200_0	200	240	0	218	0	8	8	12342	0,23
200_20	200	240	20	218	20	8	8	11504	0,221
200_50	200	240	50	218	50	8	8	12126	0,271
200_100	200	240	100	218	100	8	8	13964	0,291
200_150	200	240	150	218	150	8	8	15732	0,28
300_0	300	352	0	330	0	12	12	17137	0,441
300_30	300	352	30	330	30	12	12	16990	0,461
300_70	300	352	70	330	70	12	12	18673	0,561
300_120	300	352	120	330	120	12	12	19805	0,701
300_200	300	352	200	330	200	12	12	22345	0,711
400_0	400	471	0	443	0	16	16	22775	1,132
400_50	400	471	50	443	50	16	16	22932	1,72
400_100	400	471	100	443	100	16	16	24513	1,51
400_200	400	471	200	443	200	16	16	26987	1,152
400_300	400	471	300	443	300	16	16	30106	1,301
500_0	500	584	0	549	0	20	20	28275	2,204
500_100	500	584	100	549	100	20	20	29441	1,91
500_200	500	584	200	549	200	20	20	32184	1,142
500_300	500	584	300	549	300	20	20	35034	1,241
500_400	500	584	400	549	400	20	20	38348	1,242

Para os grafos de Corberán, colocamos a origem do problema como sendo um dos pontos da solução do GTSP. Todas as instâncias possuíam custos variados. O resultados das instâncias, tanto as nossas, quanto as de Corberán podem ser encontradas no anexo.

3.6 – Comentários dos resultados

A princípio devemos considerar relevante que o método proposto constrói uma única solução para o PCRM, com uma complexidade que pode ser polinomial, dependendo do algoritmo usado no GTSP. Isto significa que as soluções encontradas no GTSP não são garantidas serem ótimas, assim como a solução da última etapa, encontra uma única orientação viável para a única componente de arcos requeridos criada. Neste ponto repousa a fraqueza do método, pois ele não explora a diversificação possível que um método do tipo Monte-Carlo ou uma metaheurística como o GRASP, VNS ou Busca Tabu poderiam fazer. Na prática os métodos VNS e de Busca Tabu teriam que ser bem elaborados para permitir soluções interessantes para o PCRM utilizando o nosso algoritmo construtivo, porém uma versão GRASP é recomendada para avaliar melhor o desempenho comparativamente aos resultados de Corberan & Sanchis (2000).

Consideramos a princípio que os resultados obtidos foram satisfatórios, porém o fato de uma solução estar a 46,42% do ótimo nos salta aos olhos, assim como o fato de em um único exemplo (G12C2) a nossa solução tenha superado o melhor resultado de Corberán & Sanchis (2000). Na média, a distância de 7% da solução ótima é reveladora para prosseguirmos com um trabalho futuro no assunto, uma vez que o fato de aleatorizarmos orientações na fase de Sherafat e/ou de gerarmos soluções melhores para o GTSP baseadas em uma meta-heurística GRASP, poderia retornar reduções aos limites inferiores consideráveis.

Uma heurística que poderia melhorar a solução do problema atuaria durante o processo de construção do percurso do GTSP, como já dissemos. O método testaria o melhor vértice de cada grupo, mas não a melhor combinação de vértices que gere um percurso mínimo, podendo atingir até o percurso ótimo.

Nos gráficos das figuras 3.5, 3.6, 3.7, 3.8 e 3.9 podemos verificar o comportamento do tempo e do custo com aumento da quantidade de arestas requeridas, para instâncias de custo unitário nas arestas, referentes aos dados da tabela 3.1.

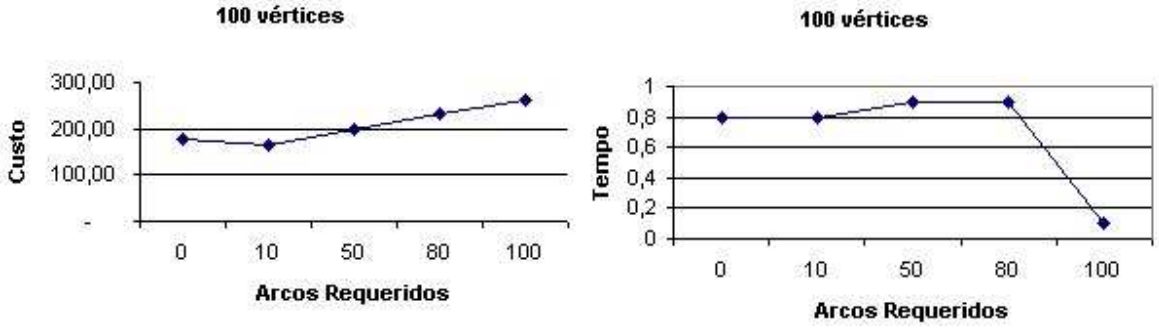


Figura 3.5: Gráficos de custo e tempo referentes à tabela 3.1 para grafos de 100 vértices.

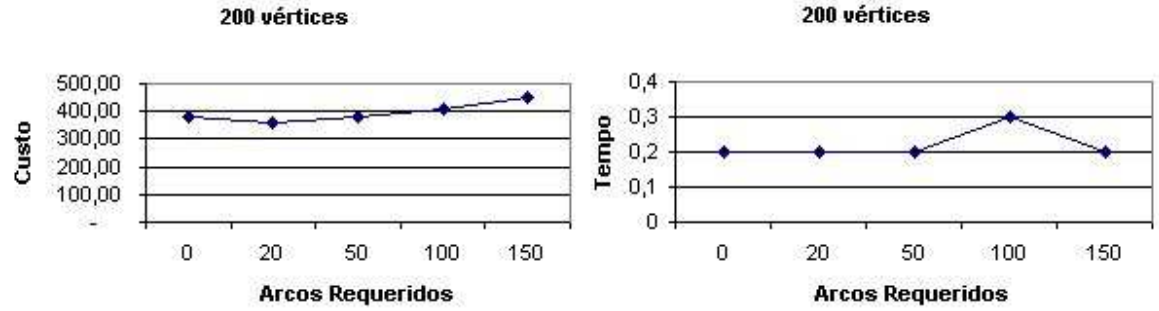


Figura 3.6: Gráficos de custo e tempo referentes à tabela 3.1 para grafos de 200 vértices.

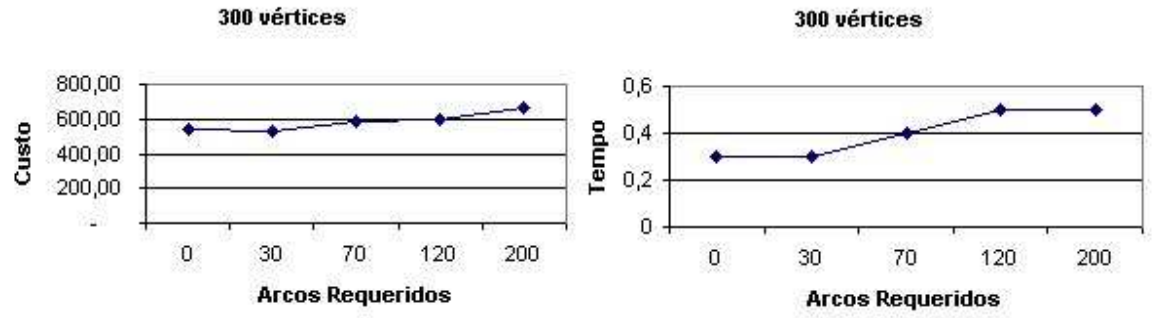


Figura 3.7: Gráficos de custo e tempo referentes à tabela 3.1 para grafos de 300 vértices.

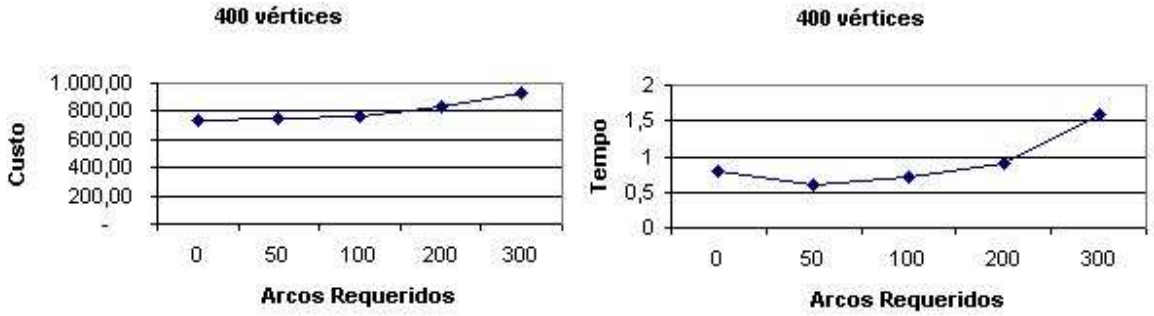


Figura 3.8: Gráficos de custo e tempo referentes à tabela 3.1 para grafos de 400 vértices.

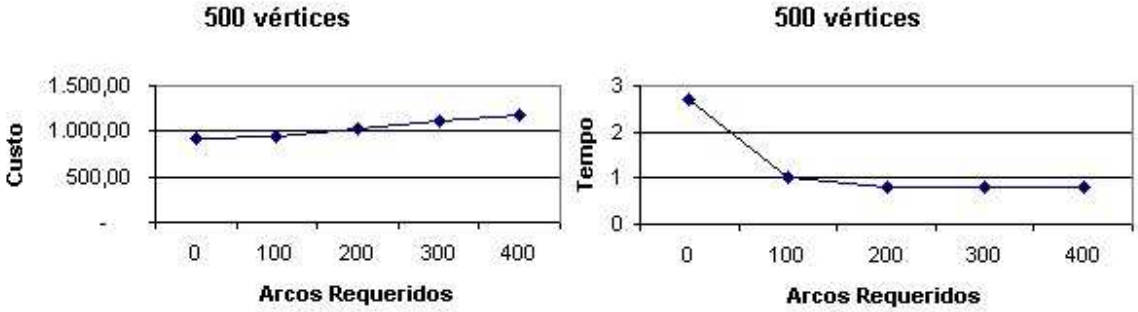


Figura 3.9: Gráficos de custo e tempo referentes à tabela 3.1 para grafos de 500 vértices.

Através de nossos testes, concluímos que nosso algoritmo não possui melhores resultados que os de Corberan como está em sua fase construtiva, porém estamos próximos deles. Apesar disto, o método é mais fácil de ser compreendido e mais fácil de ser implementado, desde que sejam conhecidos os métodos de fluxo em redes, caminho mínimo, GTSP e relações de equivalência aplicada ao contexto que ensaiamos.

Em várias instâncias conseguimos encontrar a mesma quantidade de componentes conexas que Corberán, porém em muitos exemplos o número de grupos foi maior, acarretando um aumento nos custos. Isto se deve à imprecisão do nosso algoritmo de obtenção de grupos, que em geral está a 1 grupo do número exato. Em alguns casos a falha é bem maior, revelando a necessidade de se trabalhar mais neste algoritmo para obtenção de resultados melhores.

Outro fato que a nosso ver provoca um aumento nos custos, é que nosso algoritmo de resolução do GTSP não é tão bom, e muitas vezes faz com que o caixeiro passe por mais de uma vez pelo mesmo grupo.

Um ponto fraco de nosso algoritmo é a utilização de um algoritmo de *branch-and-bound* para a resolução do GTSP, pois para determinadas instâncias, ele pode não conseguir exibir uma solução. Não foram vistos estes casos nas instâncias rodadas, mas sabemos que isto pode acontecer, conforme Syslo et al (1986).

Os gráficos comparativos encontrados no anexo entre os custos mostram que nossa solução acompanha de perto a solução de Corberán.

A tabela 3.3 mostra um resumo das informações encontradas no anexo, e informam a distância média do limite inferior de cada grupo de instâncias, e a média final para todas as instâncias, e estamos 7,12% na média pior que o LB, enquanto que Corberán está a 1,95 % pior que o LB na média.

Tabela 3.3: Média de distância do limite inferior para os testes.

Grupo de Instância	Média PCR (%)	Média Corberán (%)
G03	3,12	0,86
G04	5,76	1,19
G05	5,05	0,66
G06	3,86	0,98
G07	5,67	1,91
G08	6,03	1,03
G09	9,26	2,1
G10	8,32	2,98
G11	9,28	2,15
G12	14,91	5,7
Média	**7,12**	**1,95**

O tempo de resolução do PCRM utilizando nosso algoritmo foi bem menor, e bem mais constante, como podemos observar através dos gráficos no anexo. Isto acontece é claro porque o método proposto por Corberán & Sanchis é baseado em Busca Tabu, causando flutuações nos tempos de solução. Apesar disto, a curva do tempo (complexidade) das soluções de Corberán & Sanchis é diferente da obtida por nós, não sendo tão constante, demonstrando uma certa variação nos tempos de instância para instância.

Como melhoria, também poderia ser testada uma série de vértices de cada grupo, aplicando-se alguma meta-heurística, como o VNS, GRASP ou Tabu-Search, e descobrindo-se

o melhor vértice de entrada e o melhor vértice de saída de cada grupo, ou seja, haveria uma circulação dentro de cada grupo, um pequeno circuito que teria a origem e o destino descobertos pela heurística. Então, a partir destes pontos, seria calculado o GTSP e computando o melhor resultado, ou seja, o que gerasse o menor percurso entre os grupos.

Outra melhoria poderia ser implementada durante o tratamento dos fluxos triangulares, pois cada elo foi orientado somente uma vez. Uma melhoria seria orientar em um sentido, e verificar se a orientação no outro sentido seria melhor ou não, e assim por diante.

3.7 – Aspectos do Sistema XNês

O Sistema Xnês foi concebido inicialmente com o intuito de servir como uma ferramenta de exploração do Problema do Carteiro Chinês (PCC). Ela foi desenvolvida em conjunto pela Universidade Estadual do Ceará, Universidade Federal do Espírito Santo, e a GRAPHVS [MAEG99].

O seu ambiente é gráfico, e possui como principal componente um gerador de instâncias de grafos, as quais podem ser descritas conforme as necessidades do usuário. O ambiente geral pode ser visto na figura 3.10, onde percebemos um conjunto de tarefas no menu, as quais são processados arquivos, que salvam as informações dos grafos e definem suas operações.

Instâncias com até 20 mil vértices e 60 mil arestas podem ser geradas pelo sistema, porém seu desempenho ainda não foi testado, haja vista se tratar apenas dos limites do sistema.

O Sistema Xnês foi desenvolvido dentro do paradigma da programação orientada a objetos, em ambiente DELPHI 6.0 (Borland International) [CANTU99], e em Borland C++ 4.0 (INPRISE INC). Está disponível no site http://www.uece.br/~negreiro numa versão demo (versão 1.0) para até 50 vértices, e um número limitado de arcos e elos a no máximo 1.000.

Por se tratar de um ambiente gráfico, os vértices são posicionados um a um através dos recursos do editor de grafos, e o grafo gerado é sempre de natureza simples, ou seja, o XNÊS não gera multi-grafos, a menos na solução de saída do carteiro, que pode ser exibida como um multi-grafo ou através de um percurso animado.

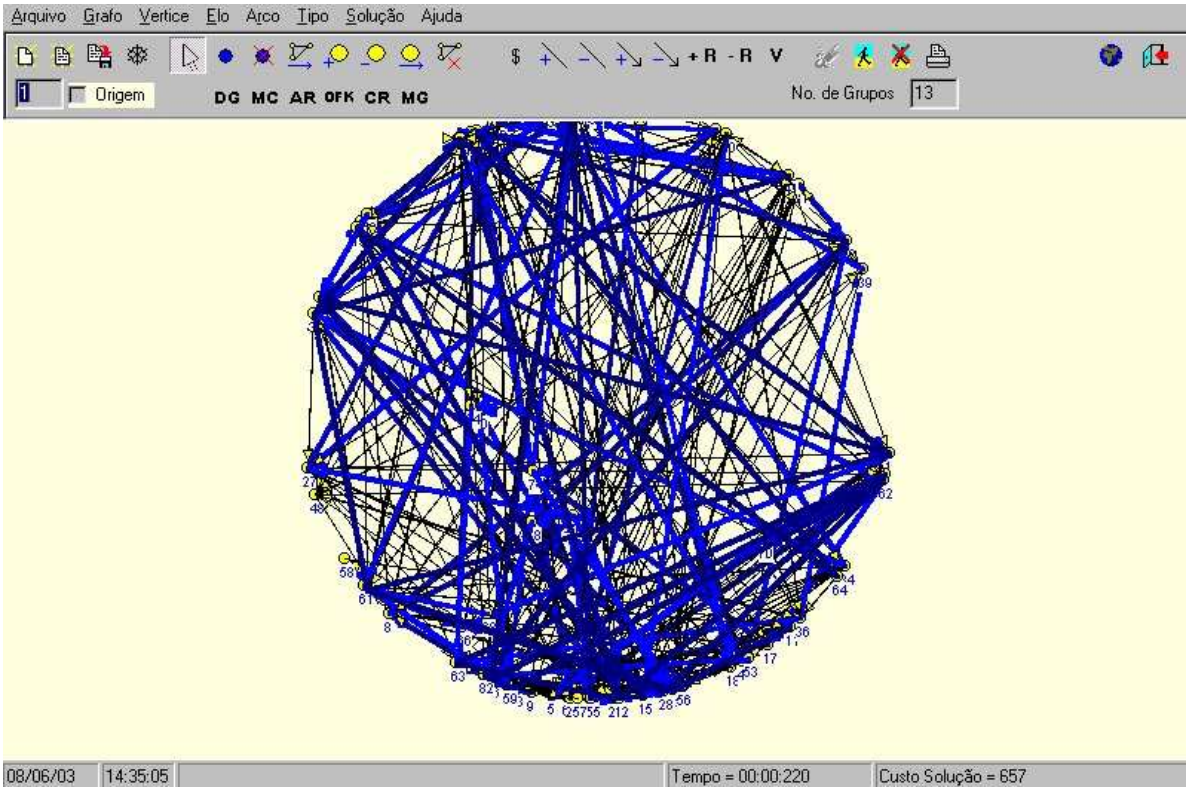


Figura 3.10: Ambiente do Sistema Xnês com grafo.

Entre as tarefas básicas sobre grafos temos: incluir / excluir vértices, arcos e elos, mover vértices, consultar e alterar informações dos elementos do grafo. Além disto, o sistema também possui uma opção de inserir custos nos arcos, de forma manual ou respeitando uma escala, de modo a preservar alguma relação de escala existente entre o mapa de fundo e o grafo gerado. Estas ferramentas permitem que os grafos sejam alterados dinamicamente e visualizados os efeitos das mudanças provocadas em função de variações que possam existir na instância do modelo proposto. Além dos custos, é possível informar as ligações requeridas e editá-las na grade de ligações. Após o processo de edição em grade, o estado do grafo é diretamente modificado no formulário gráfico de apresentação do grafo em uso.

Dentre as opções de geração de soluções, o Sistema Xnês checa automaticamente a conectividade do grafo em uso. O sistema também permite a execução de algoritmos, conforme o modelo em uso (simétrico, direcionado ou misto), processando os algoritmos do PCC e agora com o PCR.

Como saída dos algoritmos, é possível a visualização da solução do PCC sob a forma de animação no grafo, onde o caminho a ser percorrido é desenhado, ou sob a forma de um

multi-grafo. Na figura 3.11 observa-se o grafo original, e na figura 3.12 o seu multi-grafo correspondente. Este multi-grafo exibe a quantidade de vezes em que um arco é visitado. Para o PCR, é possível a visualização estática da solução no grafo, e também sob a forma de um sub-grafo, de onde serão geradas as possíveis soluções.

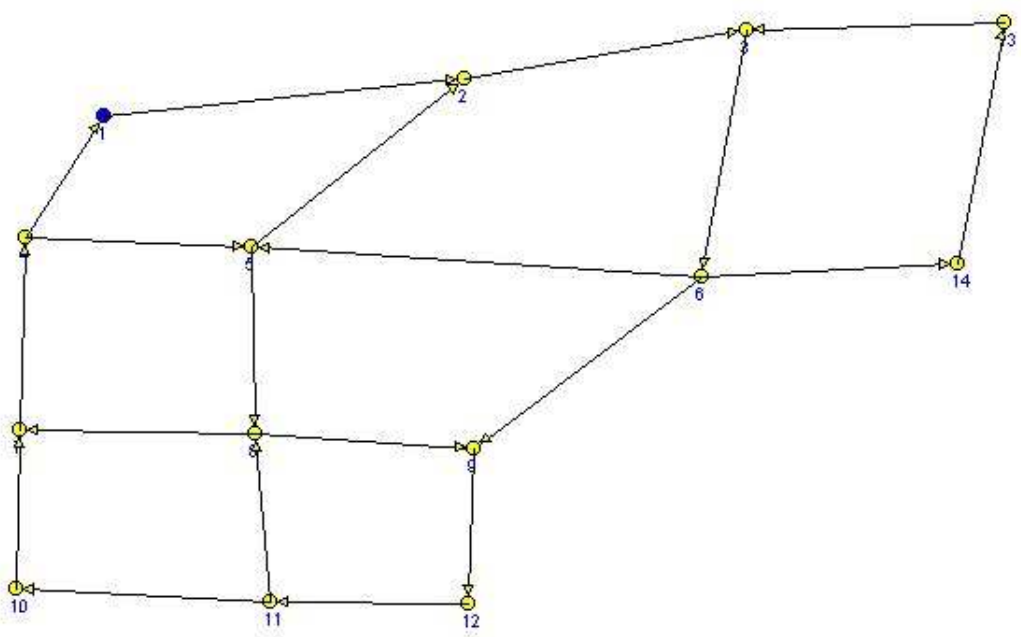


Figura 3.11: Grafo criado no Sistema Xnês.

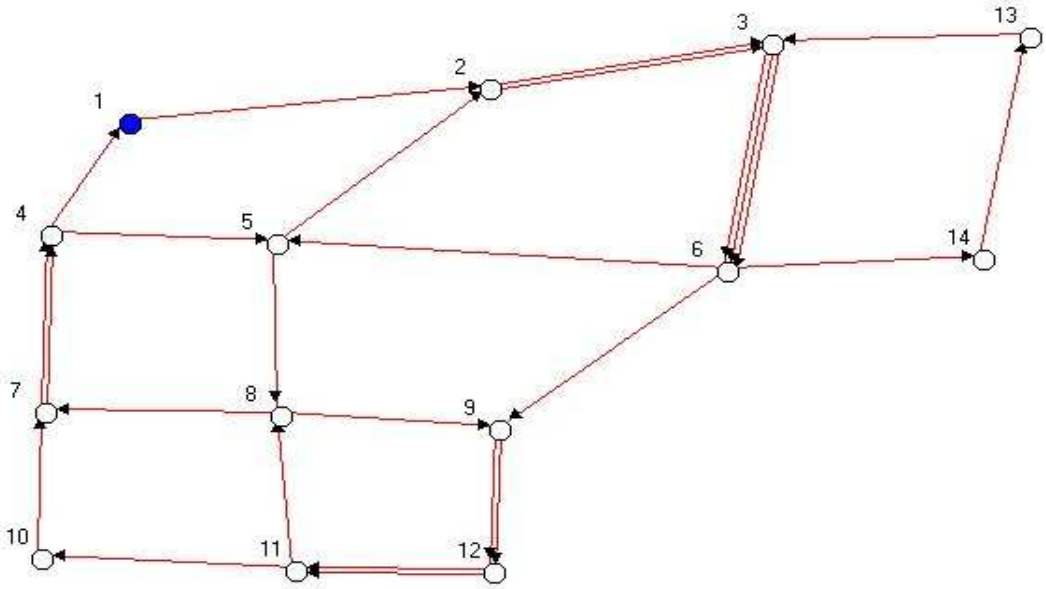


Figura 3.12: Visão do multi-grafo solução para o PCC da instância da figura 3.11.

3.8 – Funcionalidades de Resolução do PCRM do Sistema XNÊS

Foram implementadas algumas funcionalidades gráficas no sistema XNÊS para auxiliar na criação de instâncias de grafos e manipulação das informações necessárias ao PCRM.

3.8.1 – Módulo de Arcos Requeridos

Esta funcionalidade permite ao usuário selecionar os arcos que se deseja que o PCRM percorra, bem como definir seu fluxo mínimo e fluxo máximo.

Se um arco tem passagem obrigatória, seu fluxo mínimo será igual a 1. Esta associação permitirá ao algoritmo de fluxo mínimo a passagem obrigatória de pelo menos uma unidade de fluxo no arco, permitindo a circulação de fluxo no grafo.

3.8.2 – Sub-grafo Gerador de Soluções

Permite a visualização dos arcos que serão necessários para a geração de percursos soluções para o PCRM. Somente os arcos necessários serão visualizados. Várias soluções para o PCRM poderão ser geradas a partir deste sub-grafo.

É possível verificar a quantidade de vezes que é necessário passar por um determinado arco para se gerar um percurso solução para o PCRM.

3.8.3 – Módulo de Algoritmos do PCRM

Os algoritmos usados para o PCRM: Contagem de Grupos (Relações de Equivalência), Caminhos Mínimos, Caixeiro Viajante (B&B), GTSP e Fluxo Máximo de Custo Mínimo em Redes canalizadas, foram introduzidos como meio de visualização dos passos de solução do procedimento que construímos. Este modo serviu para dirigir o pensamento sobre a construção de nosso algoritmo para as instâncias desenvolvidas.

Este módulo será transferido para a versão do sistema XNÊS 2.0, que tem a capacidade de tratar instâncias muito maiores e um potencial de visualização muito mais elaborado que a primeira versão.

Conclusão

Os problemas de otimização combinatória são comumente problemas que requerem muito esforço computacional para a geração de soluções ótimas. Estas soluções ótimas podem ser bastante complicadas de se aplicarem na prática, onde são feitos muitas vezes “arredondamentos” na solução.

Foram abordados dois problemas de otimização combinatória: o problema de escalonamento e o problema de roteamento.

Quanto ao problema de escalonamento, foi criado um modelo matemático para o Problema de Escalonamento Periódico de Veículos (PEPV), bem como o desenvolvimento de um procedimento guloso combinado a meta-heurísticas para a melhoria dos resultados, utilizando-se do VNS, GRASP e BUSCA TABU.

Foi criado um software que gera soluções gulosas para uma escala de serviço baseada no número de veículos, zonas, intervalo de tempo, freqüência de repetição e custos.

Os resultados obtidos indicam que para instâncias de até “6 meses”, a solução é rapidamente obtida. Foi observado que à medida que se aumenta a freqüência de repetição de visitas em zonas, aumenta a probabilidade de se encontrar soluções melhores. Não observamos, no entanto o efeito do aumento do número de iterações de exploração do espaço de estados, pois não utilizamos meta-heurísticas ou mesmo algum método do tipo Monte Carlo para avaliar diversificação.

Foi desenvolvido um método para a solução do Problema do Carteiro Rural, baseado no GTSP e em fluxo de Custo Mínimo. Foi utilizado como ambiente de programação o sistema XNÊS, que possui toda a estrutura de geração de grafos.

O algoritmo proposto resolve o PCRM combinando algoritmos de GTSP e fluxo de custo mínimo, constituindo-se uma metodologia nova na área a qual obtém soluções heurísticas de boa qualidade para o problema. Porém, estas soluções podem ainda ser melhoradas, dependendo da heurística aplicada para a conexão inicial do grupos de arcos requeridos, ou da heurística aplicada para a geração do subgrafo. Mesmo com possíveis

melhorias da solução, os resultados foram satisfatórios, gerando boas soluções para o PCRM. Um fato verificado foi a influência do aumento de arestas no tempo da solução do problema, pois à medida que se aumentava a quantidade de arestas, a resolução era mais rápida.

Comparamos os resultados de nosso algoritmo com os resultados de Corberán & Sanchis (2000), e apesar de não termos conseguido melhorá-los, chegamos próximo das soluções por eles reportadas.

Em relação às escalas de serviço, vários trabalhos podem ser desenvolvidos a partir do modelo e dos algoritmos que elaboramos. Pode-se desenvolver um modelo que suporte vários veículos percorrendo várias áreas num mesmo dia, e um veículo podendo não percorrer totalmente uma área num dia.

Um trabalho seria a utilização de outras meta-heurísticas para a verificação de melhorias na solução, e um estudo melhor comparando as meta-heurísticas.

O desenvolvimento de um algoritmo que trabalha com atividades preemptivas, ou seja, que se possa interromper a sua execução e atender a outro serviço, ou dinamicamente combinar estas atividades, fazendo um escalonamento dinâmico, seria um trabalho mais complexo que poderia ser realizado.

Um estudo mais aprofundado de teoria dos grafos poderia gerar novos métodos que facilitariam a resolução do PCRM, além da utilização de meta-heurísticas para a busca de soluções melhores. Uma melhoria para o GTSP, melhorando a conexão inicial dos grupos, como a identificação do melhor vértice de entrada e do melhor vértice de saída seria um trabalho que reduziria o custo do problema. Um trabalho para escolha do melhor elo a ser orientado quando se encontra algum fluxo triangular seria importante para reduzir o número de passagens por um determinado arco.

A geração final de um percurso sobre o multi-grafo solução do PCRM seria um trabalho a ser realizado, pois seria a aplicação do Problema do Carteiro Chinês sobre o multi-grafo.

O ambiente de desenvolvimento XNÊS permitiria a implementação de diversas aplicações associadas ao roteamento. Uma utilização de meta-heurísticas e combinações delas seria um bom trabalho para a avaliação de desempenho dos métodos aplicados em determinadas situações de um problema.

Bibliografia

[AL97] Aarts, E., Lenstra, J. K. (1997) “Local Search in Combinatorial Optimization” – Wiley Interscience Series in Discrete Mathematics Optimization

[ALOSS02] Ahuja, R. K., Liu, J., Orlin, J. B. , Sharma, D., Shughart, L. A. (2002) – “Solving Real-Life Locomotive Scheduling Problems” – University of Florida, Massachusetts Institute of Technology

[AMO93] Ahuja, R. K., Magnati, T. L., Orlin, J. B. (1993) “Network Flows – Theory, Algorithms and Applications” – Prentice Hall, Upper Saddle River, New Jersey.

[BB74] Bodin, L. D., Beltrami, E. J. (1974) “Networks and Vehicle Routing for Municipal Waste Collection” – Networks, vol 4, pgs. 65-94

[BBGT98] Bizzarri, P. , Bondavalli, A , Di Giandomenico, Tarini, F. (1998) “Planning the Execution of Task Groups” Technical Report, Scuola Sup. S. Anna - Italia

[BEPSW96] Blazewicz, J., Ecker, K. H., Pesch, E. , Schmidt, G. , Weglarz, J. (1996) “Scheduling Computer and Manufacturing Process– Springer”

[BHD90] Bazaraa, N.S., Harvus, J.J., Sherali, H.D. (1990) “Linear Programming and Network Flow” – New York, John Wiley & Sons, Singapore, 2nd.Ed.

[BMN99] Bergson, J., Maculan, N., Negreiros Gomes, M. (1999) “Uma Avaliação da Metaheurística GRASP Aplicada ao Problema de Localização Não Capacitado” - XXXI SBPO – Juiz de Fora

[BG83] Bodin, L., Golden, B., Assad, A., Ball, M. (1983) “ The State of the Art in the Routing and Scheduling of Vehicles and Crews “ – Computers and Operations Research 10 (2), 63-211.

[BS87] Bland, R. E., Shallcross, D. F. (1987) “Large Traveling Salesman Problem Arising from Experiments in X-ray Crystallography: a Preliminary Report on Computation” Technical Report No. 730, School of OR/IE, Cornell University, Ithaca, New York.

[CANTU99] Cantu, M. (1999) – “Dominando o Delphi 4 - A Bíblia” – Marco Cantù – Makron Books

[CCCM81] Christofides, N., Campos, V., Corberán, A., Mota, E. (1981) ”An Algorithm for the Rural Postman Problem“ – Technical Report I.C.O.R..81.5, Imperial College

[CCCM86] Christofides, N., Campos, V., Corberán, A., Mota, E. (1986) “An Algorithm for the Rural Postman Problem on a directed graph” – Mathematical Programming Study, 26:155-166

[CHE00] Cheto, H., Chetto, M. (2000) “Scheduling Periodic and Sporadic Tasks in a Real-Time System” — Technical Report, Laboratoire d’Automatique de Nantes

[CMR98] Corberán, A. , Martí, R. , Romero, A. , (1998) – “Heuristics for the Mixed Rural Postman Problem” – Computers & Operations Research, 27:183-203, 2000

[CMR00] Corberán, A., Marti, R., Romero, A. (2000) “A Tabu Search Algorithm for the Mixed Rural Postman Problem” – Departamento de Estadistica e Invetigación Operativa – Facultad de Matemáticas – Universidad de Valencia – Spain

[CRR93] Reeves, C. R. , (1993) - “Modern Heuristic Techniques for Combinatorial Problems” – Blackwell Scientific Publications

[CRS00] Corberán, A., Romero, A., Sanchis, J. M. (2000) ”The General Routing Problem on a mixed graph” – Working Paper

[CRS01] Corberán, A., Romero, A., Sanchis, J. M. (2001) ”The Mixed General Routing Polyhedron” – Working Paper – Dept. d’Estadística i Investigación Operativa, Universitat de Valencia, Spain – Dept. de Matemática Aplicada, Universidad Politécnica de Valencia, Spain

[CS94] Corberán, A., Sanchis, J. M. (1994) “A polyhedral approach to the rural postman problem” – European Journal of Operational Research, 79:95-114, 1994.

[EE73] Edmonds, J., Johnson, E. (1973) – “Matching, euler tours and the chinese postman problem” – Mathematical Programming, vol 5 pgs. 88-124

[EGL95] Eiselt, H. A, Gendreau, M., Laporte. G. (1995) “Arc Routing Problems, Part I: The Chinese Postman Problem” – Operations Research, vol. 43(2) pgs. 231-242.

[EJ73] Edmonds, J., Johnson, E. L. – (1973) – “Matching, Euler tours and the Chinese postman problem. Mathematical Programming” – 5:88 – 124

[EU36] Euler, L. (1736) "Solutio problematics ad geometrian situs pertinentis" – Comentarii Academae Scientarum, Petropolinae 8, pgs. 128-140

[EVN99] Neto, E. V., (1999) “Uma Heurística GRASP para o Problema de Corte Unidimensional” – Dissertação de Mestrado – Universidade Estadual do Norte Fluminense – UENF – Campos dos Goytacazes – RJ

[FF62] Ford, L.R., Fulkerson, D.R. (1962) “Flows in Networks” – Princeton University Press, Princeton NJ (USA)

[FIS94] Fischetti, M., J. J. S. Gonzalez and P. Toth (1994) “A branch-and-cut algorithm for the Symmetric generalized traveling salesman problem” – Operations Research 45, 378–394.

[FR95] Feo, T., Resende, M. (1995) “Greedy randomized adaptative search procedures”, Journal of Global Optimization Algorithms, vol. 6, pgs 109-133 - (Kluwer Academic Publishers, Boston)

[FRM02] Ferreira, J. S., Rodrigues, A. M., Moreira, L. (2002) “Components Cutting Optimization by Memetic Algorithms” - XII CLAIO 2002, Concepcion – Chile – A24N13 in CD-ROM

[GG64] Gilmore, P. C., Gomory, R. E. (1964) “Sequencing a one state variable machine: a solvable case of the travelling salesman problem”, Oper. Research 12, 1964, 655--679.

[GG78] Gavish, B., Graves, S. (1978) “The Traveling Salesman Problem and Related Problems” – Working Paper, Graduate School of Management, University of Rochester

[GL89] Glover, F (1989) “Tabu Search - Part I” – ORSA Journal on Computing 1(3): 190-206.

[GL96] Glover, F., Laguna, M. (1996) “Tabu Search”, Kluwer Academic Publishers.

[GL00] Ghiani, G., Laporte, G. (2000) “A branch-and-cut algorithm for the Undirected Rural Postman Problem” – Mathematical Programming, 87(3):467-481

[GP00] Goldbarg, M. C., Luna, H. P. L. (2000) “Otimizacao Combinatoria e Programacao Linear – Modelos e Algoritmos” – Editora Campus – ISBN 85-352-0541-1

[GUN99] Günes, E. D. (1999) “Workforce Scheduling” — Technical Report, Department of Industrial Engineering – Bilkent University – Ankara

[HEN69] Henry-Labordere, A. L. (1969) “The record balancing problem: A dynamic programming solution of a generalized traveling salesman problem” – RAIRO B2, 43-49.

[HE73] Heierholzer, C. (1873) ”Uber die moglichkeit, einen linienzung ohne wiederholing und ohne unterbrechung zu umfahren” – Mathematische Annalen IV, pgs.. 30-32

[HLN99] Hertz, A., Laporte, G. , Nanchen, P. (1999) – “Improvement procedures for the undirected rural postman problem” – INFORMS – Journal on Computing – 11, 53-62.

[HAM98] Hansen, P., Mladenović, N., (1998) “Variable Neighborhood Search: Principles and Applications” – Les Cahiers du GERAD – Annals of Operation Research

[HI41] Hitchcock, F.L. (1941) “The distribution of a product from several sources to nomerous facilities” – d Journal of Mathematical Physics, vol. 20, pgs 224-230

[HM00] Hansen, P., Mladenović, N., (2000) “Variable Neighborhood Search: A Chapter of Handbook of Applied Optimization” – Les Cahiers du GERAD

[HM98] Hansen, P. , Mladenovic, N. (1998) “An Introduction to Variable Neighborhood Search” , Les Cahiers du GERAD, G-97-51

[HOK00] Hooker, J. (2000) “Logic-Based Methods for Optimization – Combining Optimization and Constraint Satisfaction” –Wiley Interscience in Discrete Mathematics and Optimization

[KO49] Koopmans, T.C. (1949) “Optimum utilization of the transportation system” – Econometrica, vol 17

[KW62] Kwan, Mei-Ko (1962) “Graphic programming using odd and even points” –Chinese Mathematics, vol 1 pgs 273-277

[LANO83] Laporte, G., Nobert, Y. (1983) “Generalized traveling salesman problem through n sets of nodes: An integer programming approach” – INFOR 21, 1, 61-75.

[LAP97] Laporte, G. (1997) “Modeling and Solving Several Classes of Sparse Arc Routing Problems as Traveling Salesman Problems” – Centre de Recherche sur les Transports – CRT – 95 – 81

[LAS95] Laporte, G., Asef-Vaziri, A., Sriskandarajah, C. (1995) “Some applications of the generalized traveling salesman problem », Working paper CRT-95-59, Centre de Recherche sur les Transports, Université de Montréal.

[LLR85] Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A. H. G. , Shmoys, D. B. (1985) “The Traveling Salesman Problem”, John Wiley, Chichester.

[LR81] Lenstra, J., Rinooy Kan, A. (1981) “Complexity of Vehicle Routing and Scheduling Problems” – Networks, Vol. 11 pgs. 221-227.

[MAEG99] Negreiros Gomes, M. J., Palhano, A. W. C., Coutinho, E.F., Castro, G. A. , Negreiros Gomes, F. J., Rezende, B. F., Barcellos, G. C., Pereira, L. W. L. (1999) “Xnês : Um Ambiente Visual para Geração de Soluções Ótimas de Instâncias do Problema do Carteiro Chinês”- IV SPOLM - Rio de Janeiro, Dezembro.

[NB93] Noon, Ch., Bean, J. C. (1993) “An efficient transformation of the generalized traveling salesman problem”, INFOR 31, 1,39-44.

[NDSA00] Noronha, T. F., DaSilva, M. M., Aloise, D. J. (2000) “Uma Abordagem sobre Estratégias Metaheurísticas”– Relatório Técnico, UFRN – UnP

[NEG90] Negreiros Gomes, M. J. (1990) “O problema de Planejamento e Percurso de Veículos para a Coleta de Lixo Urbano Domiciliar”, Dissertação de Mestrado, Eng. de Sistemas e Computação/COPPE/UFRJ.

[NEG96] Negreiros Gomes, M. J. (1996) “Contribuições para Otimização em Grafos e Problemas de Percurso de Veículos: Sistema SisGRAFO ” – Tese de Doutorado – UFRJ

[NOON88] Noon, Ch. E. (1988) “The generalized traveling salesman problem” – Ph. D. Dissertation, niversity of Michigan.

[NP91] Nobert, Y., Picard, J. C. (1991) “An optimal algorithm for the Mixed Chinese Postman Problem” – CRT#799, Montreal.

[OR84] Orlin, J.B. (1988) “A faster strongly polynomial minimum cost flow algorithm” – Preceedings of the 20th ACM Symposium on the Theory of Computing, pgs. 377-387

[PC76] Papadimitriou, C. H. (1976) “On the complexity of edge traversing” – Journal of ACM – 23:544 – 54

[PRARIB99] Prais, M., Ribeiro, C.C. (1999) "Parameter variation in GRASP implementations.(In Proc. of the Third Metaheuristics International Conference, páginas 07 e 08. Angra dos Reis (RJ) – Brasil

[PT90] Plotkin, S., Tardos, E. (1990) “Improved dual network simplex” – Preceedings of the Frist ACM-SIAM Symposium on Discrete Algorithms, pgs. 367-376

[PW95] Pearn, W. L., Wu, T. C. (1995) “Algorithms for the rural postman problem” - Computers&Operations Research – 22:819 – 28

[RB96] Renaud, J., Boctor, F. F. (1996) “An Efficient Composite Heuristic for the Symmetric Generalizes Traveling Salesman Problem” – Télé-Université, Université du Québec, Canada, and Université Laval, Canada.

[RIAR98] Ribeiro, C. C., Aragão M. P. (1998) “Metaheurísticas” - Escola Brasileira de Computação – 1998 - http://www.inf.puc-rio.br/~celso/~poggi

[ROM97] Romero, A. (1997) “On Mixed Rural Postman Problem” – PhD thesis, Department of Statistics and OR, University of Valencia

[RR81] Ratliff, H. D., Rosenthal, A. S. (1981) "Order-Picking in a Rectangular Warehouse: A Solvable Case for the Traveling Salesman Problem" PDRC Report Series No. 81-10. Georgia Institute of Technology, Atlanta, Georgia.

[SAK70] Saksena, J. P. (1970) “Mathematical model of scheduling clients through welfare agencies”, CORS Journal 8, 185-200.

[SAN90] Sanchis, J. M. (1990) “El Poliedro del Problema del Cartero Rural” – PhD thesis, Universidad de Valencia, Spain

[SC03] Schrijver, A. (2003) “On th History of Combinatorial Optimization (till 1960)”

[SDK83] Syslo, M. M., Deo, N., Kowalic, J. S., (1983) “Discrete Optimization Algoritms with Pascal Programs” - Prentice-Hall - Englewood Cliffs, New Jersey

[SEP91] Sepehri, M. M. (1991) “The symmetric generalized traveling salesman problem”, Ph. D. Dissertation, University of Tennessee, Knoxville.

[SH88] Sherafat, H. (1988) “Uma Solução para o Problema do Carteiro Chinês Misto”, Anais do IV CLAIO – XXI SBPO, pgs. 157-170, Rio de Janeiro.

[SHA99] Shaerf, A. (1999) “A survey of automated timetabling” – Artifitial Intelligence Review vol 13(2) pgs. 87-127

[SIQ99] Siqueira, P. H. (1999) “Aplicação do Algoritmo do Matching no Problema da Construção de Escalas de Motoristas e Cobradores de Ônibus” – Dissertação de Mestrado – UFPR – Curitiba

[SKGP69] Srivastava, S., Kumar, S. S., Garg, R. C., Sen, P. (1969) “Generalized traveling salesman problem through n sets of nodes”, CORS Journal 7, 97-101.

[VLAM00] Verhaegh, W. F. J. , Lippens, P. E. R. , Aarts, E. H. L., van Meerbergen , J. L. , Werf, A. (2000) “The Complexity od Multidimensional Periodic Scheduling” — Technical Report, Phillips Research Laboratories, The Netherlands Eindhoven University of Technology

[WM92] Weiss, M. A. (1992) “Data Structures and Algorithm Analysis” – The Benjamin / Cummings Publishing Company, Inc.

[WOL98] Wolsey,Laurnce A. (1998) “Integer Programming – Wiley-Interscience Series in Discrete Mathematics and Optimization”, John Wilen & Sons, INC

[YY88] Yaxiong, L., Yongchang, Z. (1988) – “A new algorithm for the direct chinese postman problem” – Computers & Operations Research, vol. 15 pgs 577-584

Anexo

Tabela A.1: Resultados para grafos do grupo G3.

Instância	Custo PCR	Tempo PCR	Grupos PCR	Dist. PCR LB	LB	Custo C	Tempo C	Grupos C	Dist. C LB
G3C1	497	0,5	3	7,112069	464	469	0,98	3	1,077586
G3C2	968	0,6	3	2,325581	946	950	2,58	3	0,422833
G3C3	2379	0,11	3	0,168421	2375	2380	20,43	3	0,210526
G3C4	1730	0,11	3	1,944608	1697	1704	4,73	3	0,412493
G3C5	1866	0,11	3	4,187605	1791	1811	5,71	3	1,116695
G3C6	1720	0,11	3	3,427541	1663	1678	4,23	3	0,901984
G3C7	2337	0,17	3	0,732759	2320	2353	22,97	3	1,422414
G3C8	1332	0,11	5	7,073955	1244	1262	6,48	5	1,446945
G3C9	4151	0,22	3	1,441838	4092	4109	33,23	3	0,415445
G3C10	3384	0,22	3	2,951019	3287	3327	49,57	3	1,216915

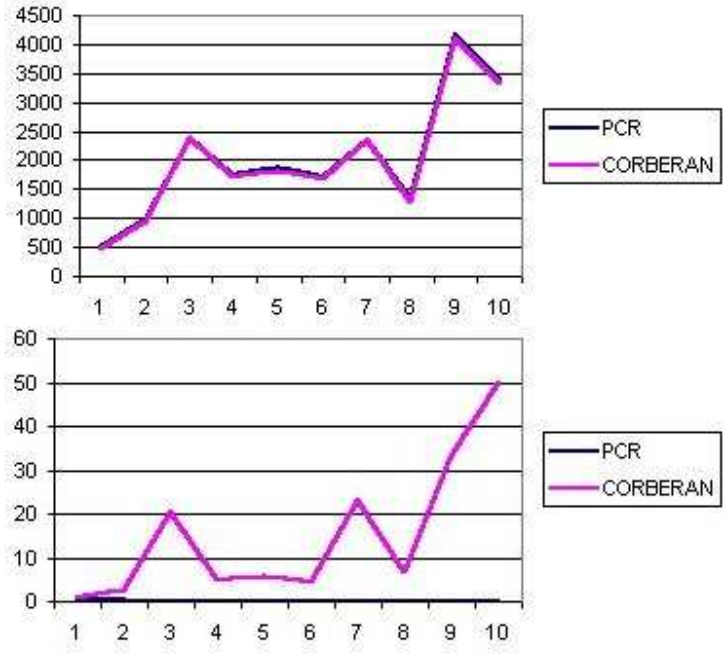


Figura A.1: Gráfico de custo e tempo para o grupo G3.

Tabela A.2: Resultados para grafos do grupo G4.

Instância	Custo PCR	Tempo PCR	Grupos PCR	Dist. PCR LB	LB	Custo C	Tempo C	Grupos C	Dist. C LB
G4C1	864	0,6	4	8,953342	793	793	4,12	4	0
G4C2	1537	0,5	4	3,851351	1480	1482	3,13	3	0,135135
G4C3	2378	0,5	4	2,943723	2310	2324	9,72	4	0,606061
G4C4	2112	0,11	4	1,587302	2079	2092	9,99	4	0,625301
G4C5	1924	0,6	4	3,887689	1852	1869	3,57	4	0,917927
G4C6	1071	0,5	7	6,039604	1010	1013	3,25	5	0,29703
G4C7	1683	0,11	4	6,451613	1581	1594	4,66	4	0,822264
G4C8	3461	0,22	4	1,944035	3395	3412	28,62	3	0,500736
G4C9	1400	0,11	15	18,04384	1186	1271	5,33	15	7,166948
G4C10	3034	0,16	4	3,939705	2919	2946	28,62	2	0,924974

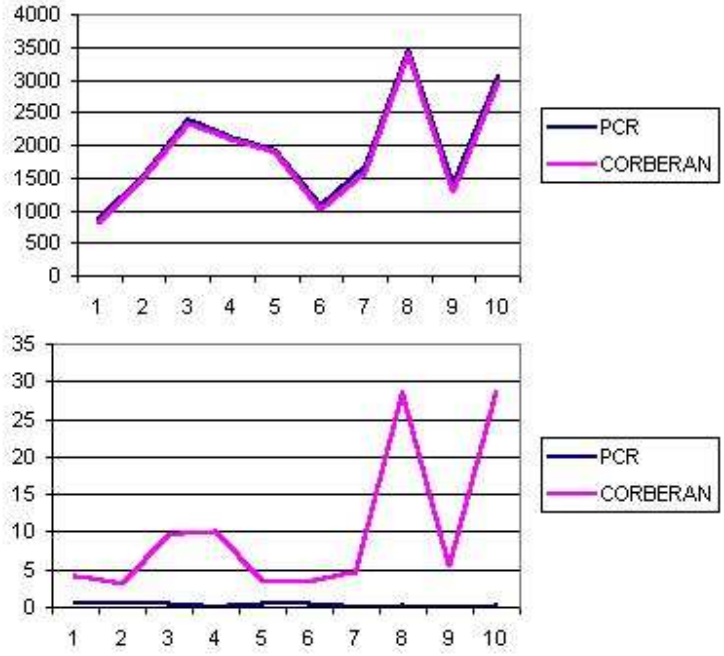


Figura A.2: Gráfico de custo e tempo para o grupo G4.

Tabela A.3: Resultados para grafos do grupo G5.

Instância	Custo PCR	Tempo PCR	Grupos PCR	Dist. PCR LB	LB	Custo C	Tempo C	Grupos C	Dist. C LB
G5C1	420	0,22	5	13,82114	369	370	2,25	5	0,271003
G5C2	1265	0,6	5	5,946399	1194	1203	3,95	5	0,753769
G5C3	1437	0,6	5	3,530259	1388	1405	3,79	5	1,224784
G5C4	1133	0,55	5	5,88785	1070	1073	5,28	4	0,280374
G5C5	1161	0,77	6	5,163043	1104	1124	3,89	6	1,811594
G5C6	3512	0,93	5	2,810304	3416	3430	17,53	5	0,409836
G5C7	2445	0,88	5	4,587745	2338	2355	11,92	5	0,737889
G5C8	2785	1,4	5	2,843427	2708	2711	20,21	5	0,110783
G5C9	3220	1,1	5	3,271328	3118	3140	10,27	4	0,705581
G5C10	2872	0,99	5	2,718169	2796	2805	8,79	4	0,321888

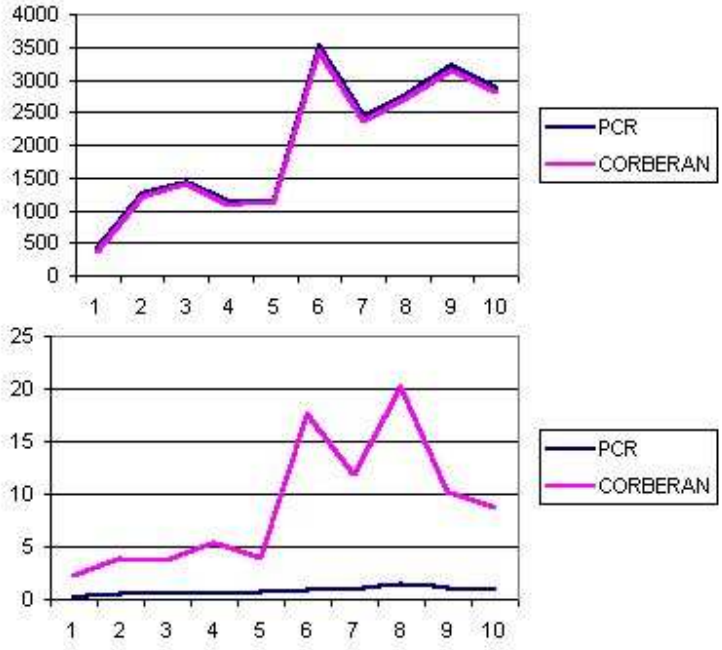


Figura A.3: Gráfico de custo e tempo para o grupo G5.

Tabela A.4: Resultados para grafos do grupo G6.

Instância	Custo PCR	Tempo PCR	Grupos PCR	Dist. PCR LB	LB	Custo C	Tempo C	Grupos C	Dist. C LB
G6C1	333	0,6	6	2,777778	324	327	2,96	6	0,925926
G6C2	1226	0,6	6	1,869547	1204	1216	4,44	6	1,038637
G6C3	1129	0,11	6	2,079566	1106	1118	3,96	5	1,084991
G6C4	1751	0,11	6	5,038992	1667	1684	4,12	6	1,019796
G6C5	2652	0,16	6	4,204322	2545	2558	5,33	6	0,510806
G6C6	1298	0,11	6	3,097697	1259	1263	2,85	6	0,317712
G6C7	1742	0,11	6	3,752233	1679	1718	5,49	6	2,322811
G6C8	2147	0,17	6	8,106747	1986	2026	13,02	5	2,014099
G6C9	2803	0,16	6	1,927273	2750	2750	10,71	2	0
G6C10	3123	0,22	6	5,792683	2952	2970	28,9	6	0,609756

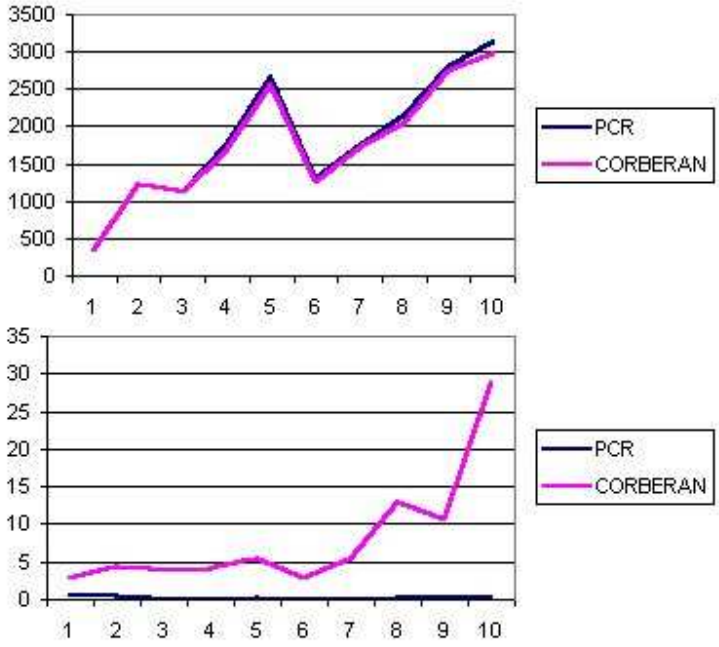


Figura A.4: Gráfico de custo e tempo para o grupo G6.

Tabela A.5: Resultados para grafos do grupo G7.

Instância	Custo PCR	Tempo PCR	Grupos PCR	Dist. PCR LB	LB	Custo C	Tempo C	Grupos C	Dist. C LB
G7C1	260	0,6	7	10,4034	235,5	255	1,87	6	8,280255
G7C2	863	0,11	7	5,759804	816	831	8,46	6	1,838235
G7C3	916	0,11	8	8,146399	847	862	5,93	8	1,770956
G7C4	2185	0,11	7	6,950563	2043	2072	7,69	7	1,419481
G7C5	2293	0,11	7	4,322111	2198	2216	6,37	7	0,818926
G7C6	2964	0,17	7	3,365301	2867,5	2879	12,04	7	0,401046
G7C7	2798	0,22	7	5,884579	2642,5	2683	9,39	7	1,53264
G7C8	2729	0,11	7	3,764259	2630	2646	8,91	6	0,608365
G7C9	2006	0,17	7	5,467928	1902	1939	19,44	8	1,945321
G7C10	3106	0,22	7	2,71164	3024	3040	16,48	6	0,529101

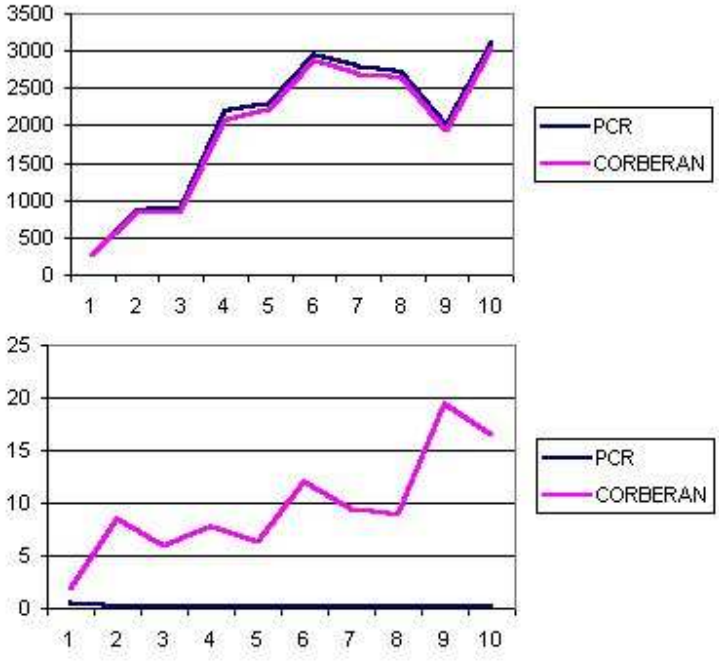


Figura A.5: Gráfico de custo e tempo para o grupo G7.

Tabela A.6: Resultados para grafos do grupo G8.

Instância	Custo PCR	Tempo PCR	Grupos PCR	Dist. PCR LB	LB	Custo C	Tempo C	Grupos C	Dist. C LB
G8C1	694	0,6	8	6,687164	650,5	654	4,18	8	0,538048
G8C2	1242	0,6	8	4,194631	1192	1203	4,45	8	0,922819
G8C3	1091	0,5	8	5,410628	1035	1042	5,44	7	0,676329
G8C4	2081	0,11	8	5,688167	1969	1991	3,74	8	1,117318
G8C5	1881	0,11	8	3,636364	1815	1850	7,58	8	1,928375
G8C6	1915	0,11	8	14,07841	1678,67	1704	9,84	8	1,508933
G8C7	2698	0,17	8	2,977099	2620	2628	23,07	6	0,305344
G8C8	2419	0,17	8	9,308631	2213	2242	14,67	8	1,310438
G8C9	2432	0,16	8	2,832981	2365	2382	23,67	7	0,718816
G8C10	3138	0,27	8	5,514459	2974	3013	14,11	7	1,311365

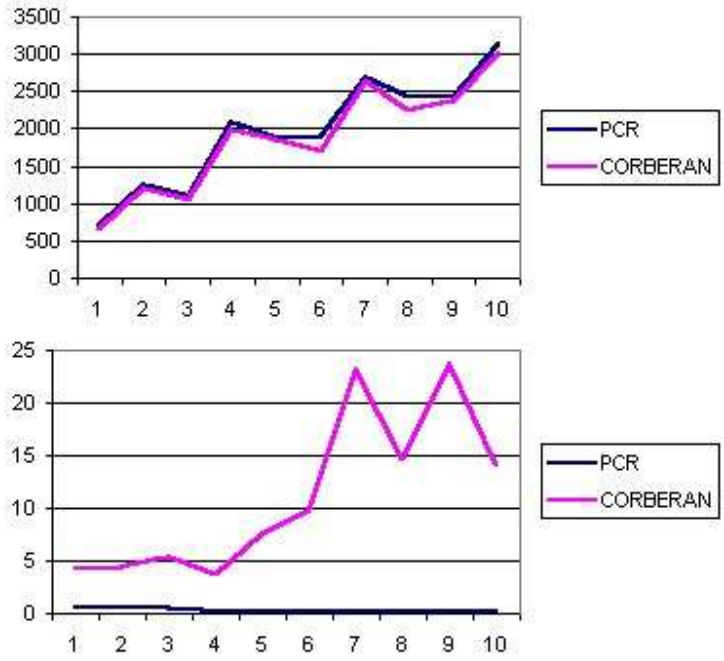


Figura A.6: Gráfico de custo e tempo para o grupo G8.

Tabela A.7: Resultados para grafos do grupo G9.

Instância	Custo PCR	Tempo PCR	Grupos PCR	Dist. PCR LB	LB	Custo C	Tempo C	Grupos C	Dist. C LB
G9C1	415	0,5	9	11,55914	372	409	3,63	8	9,946237
G9C2	1102	0,5	9	7,512195	1025	1030	4,67	9	0,487805
G9C3	1754	0,11	9	14,11841	1537	1556	6,98	9	1,236174
G9C4	1082	0,5	9	12,00828	966	977	4,56	9	1,138716
G9C5	2609	0,16	9	6,751227	2444	2464	6,04	9	0,818331
G9C6	3189	0,16	9	9,399657	2915	2974	5,39	8	2,024014
G9C7	2298	0,22	9	15,24574	1994	2026	7,69	8	1,604814
G9C8	2224	0,11	9	4,11985	2136	2151	11,09	7	0,702247
G9C9	3575	0,17	9	5,519481	3388	3409	15,44	7	0,619835
G9C10	2064	0,22	9	6,446622	1939	1986	16,59	7	2,42393

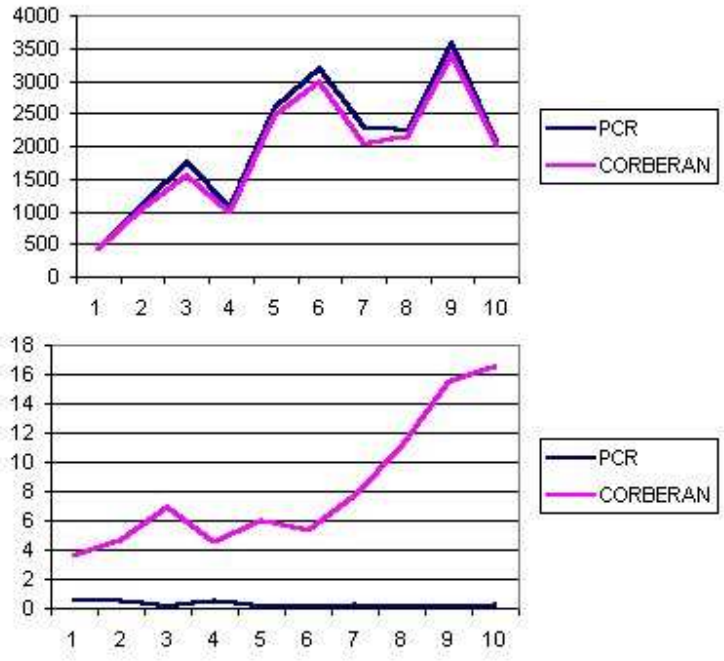


Figura A.7: Gráfico de custo e tempo para o grupo G9.

Tabela A.8: Resultados para grafos do grupo G10.

Instância	Custo PCR	Tempo PCR	Grupos PCR	Dist. PCR LB	LB	Custo C	Tempo C	Grupos C	Dist. C LB
G10C1	497	0,6	10	11,18568	447	485	5,11	10	8,501119
G10C2	1272	0,6	10	10,3209	1153	1186	3,08	10	2,862099
G10C3	1122	0,11	10	18,04314	950,5	975	3,69	10	2,577591
G10C4	1227	0,11	10	6,05013	1157	1197	5,16	10	3,457217
G10C5	2777	0,11	10	5,469047	2633	2673	7,3	10	1,51918
G10C6	2288	0,17	10	5,827937	2162	2219	6,28	10	2,636448
G10C7	3205	0,16	10	4,904146	3055,17	3117	13,19	10	2,023783
G10C8	2700	0,17	10	4,006163	2596	2612	6,81	9	0,616333
G10C9	2255	0,11	10	6,418122	2119	2164	6,04	8	2,123643
G10C10	2780	0,16	10	11,00773	2504,33	2592	8,19	10	3,500737

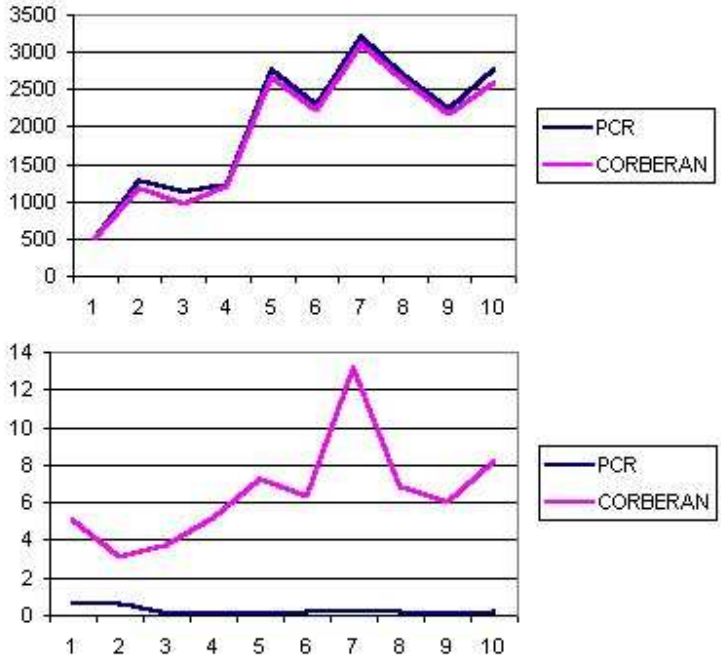


Figura A.8: Gráfico de custo e tempo para o grupo G10.

Tabela A.9: Resultados para grafos do grupo G11.

Instância	Custo PCR	Tempo PCR	Grupos PCR	Dist. PCR LB	LB	Custo C	Tempo C	Grupos C	Dist. C LB
G11C1	228	0,5	3	21,92513	187	187	1,38	3	0
G11C2	560	0,5	10	10,89109	505	516	4,06	9	2,178218
G11C3	585	0,5	13	16,76647	501	538	2,59	12	7,38523
G11C4	1205	0,6	11	5,148342	1146	1159	4,89	10	1,13438
G11C5	2137	0,11	11	5,322819	2029	2068	5,05	11	1,922129
G11C6	2789	0,17	11	6,247619	2625	2696	5,6	11	2,704762
G11C7	3095	0,16	11	5,775803	2926	2956	5,38	11	1,02529
G11C8	2521	0,11	11	5,26096	2395	2407	4,29	10	0,501044
G11C9	2379	0,17	11	9,833795	2166	2227	5,11	10	2,816251
G11C10	2249	0,17	11	5,68609	2128	2169	9,28	10	1,926692

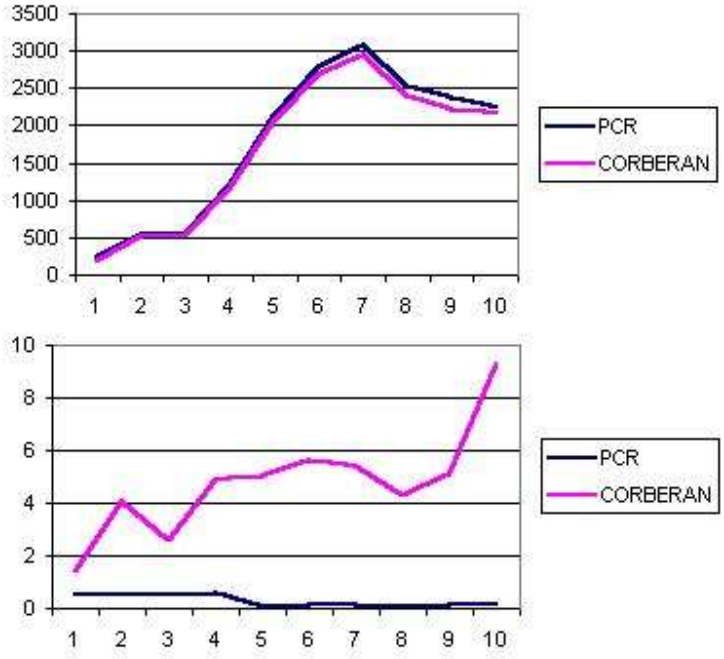


Figura A.9: Gráfico de custo e tempo para o grupo G11.

Tabela A.10: Resultados para grafos do grupo G12.

Instância	Custo PCR	Tempo PCR	Grupos PCR	Dist. PCR LB	LB	Custo C	Tempo C	Grupos C	Dist. C LB
G12C1	165	0,6	8	25,9542	131	136	1,92	7	3,816794
G12C2	451	0,5	11	46,42857	308	373	2,03	11	21,1039
G12C3	586	0,11	16	22,08333	480	556	2,69	15	15,83333
G12C4	2300	0,17	12	11,98372	2053,87	2102	7,59	12	2,343381
G12C5	1404	0,11	12	7,257448	1309	1324	3,24	12	1,145913
G12C6	2397	0,17	12	7,44061	2231	2272	7,1	12	1,837741
G12C7	2450	0,22	12	8,792185	2252	2329	6,81	12	3,419183
G12C8	2679	0,22	12	7,245797	2498	2556	11,04	11	2,321857
G12C9	2534	0,22	12	8,221226	2341,5	2389	6,98	11	2,028614
G12C10	2711	0,22	12	3,710788	2614	2648	11,81	10	1,300689

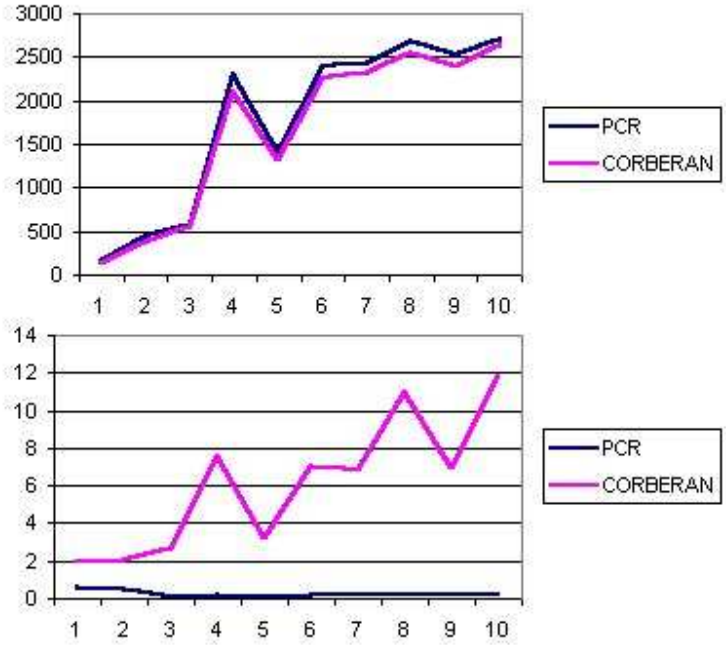


Figura A.10: Gráfico de custo e tempo para o grupo G12.