UMA HEURÍSTICA BASEADA EM COLÔNIA DE FORMIGAS PARA O PROBLEMA DO CARTEIRO CHINÊS MISTO

JAIRO. VIANA JR.

UMA HEURÍSTICA BASEADA EM COLÔNIA DE FORMIGAS PARA O PROBLEMA DO CARTEIRO CHINÊS MISTO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais. Departamento de Ciência da Computação. como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

ORIENTADOR: PROF. SEBASTIÁN ALBERTO URRUTIA

Belo Horizonte

Julho de 2010

Viana Jr., Jairo.
V614h Uma heurística baseada em colônia de formigas para o problema do Carteiro Chinês Misto / Jairo. Viana Jr.. — Belo Horizonte, 2010
xviii, 46 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais. Departamento de Ciência da Computação.
Orientador: Prof. Sebastián Alberto Urrutia

1. Teoria dos Grafos - Tese. 2. Otimização - Tese. 3. Colônia de Formigas - Tese. 4. Problema do carteiro chinês misto - Tese. I. Orientador II. Título.

CDU 519.6*62 (043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Uma heurística baseada em colônia de formigas para o problema do carteiro chinês misto

JAIRO VIANA JÚNIOR

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. SEBASTIÁN ALBERTO URRUTIA - Orientador
Departamento de Ciência da Computação - UFMG

PROF. GERALDO ROBSON MATEUS
Departamento de Ciência da Computação - UFMG

PROF. THIAGO FERREIRA DE NORONHA
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 09 de julho de 2010.

Agradecimentos

Meus agradecimentos vão para todas as pessoas que contribuíram para conclusão deste trabalho. Agradeço especialmente a minha mãe, minhas irmãs e minha noiva, Moema Gomes Linhares, por todo o amor, paciência e incentivo nos momentos em que mais precisei. Agradeço ao meu orientador, Professor Sebastián Alberto Urrutia, pela paciência e compreensão, assim como as verdadeiras lições sobre computação científica e pesquisa em computação. Além disso, um agradecimento a meus companheiros de trabalho na Líder Aviação, pelo incentivo e companheirismo, e principalmente, a João Carlos Pereira Campos e Edgard de Araújo Mitre. Agradeço também áquelas pessoas que, mesmo durante as piores ventanias, não me deixaram desistir ou fraquejar e sempre tiveram uma palavra de incentivo. Nesse ínterim, além dos já citados, agradeço a Patrícia Corrêa, meus sobrinhos e cunhados e a Renato e Cândida Linhares. Que possamos conquistar cada vez mais.

"Pois a vontade, quando não quer, não se deixa vencer, como sucede á chama que mil vezes curvada pelo sopro, outras mil se ergue."

(Dante Alighieri)

Resumo

O Problema do Carteiro Chinês (Chinese Postman Problem, em inglês) é um dos desafios clássicos em Ciência da Computação, especificamente na área de grafos. É um problema de otimização combinatória bem conhecido com uma infinidade de aplicações práticas. Grande parte dos problemas do mundo real onde seja preciso percorrer uma cidade, um trecho de mapa ou uma rede de recursos, tais como coleta de lixo urbano, inspeção em fios, retirada de gelo ou entrega de cartas, podem ser modelados através do Chinese Postman problem (CPP) ou de suas variações. Basicamente, procura-se um caminho de custo mínimo em um grafo, de forma que o caminho contemple todas as arestas e arcos do grafo pelo menos uma vez. Este problema possui solução em tempo polinomial tanto para a versão orientada quanto para a versão não-orientada. O objeto de estudo deste trabalho é a variação do CPP conhecida como *Problema do carteiro chinês misto*, onde co-existem em um grafo arestas (não-direcionadas) e arcos. Esta variação do CPP é um problema NP-Completo [Papadimitriou, 1976]. É proposta uma heurística baseada em colônias de formigas para o MCPP (Problema do Carteiro Chinês Misto), assim como um procedimento de melhora contínua da solução.

Palavras-chave: Otimização, Colônia de Formigas, Carteiro Chinês Misto.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

O problema do Carteiro Chinês, ou Chinese Postman Problem (CPP) foi introduzido por Meigu Guan [1] [Meigu, 1962]. Uma das aplicações do CPP pode ser descrita como um carteiro que parte de um posto dos Correios, entrega cartas em um conjunto de ruas e volta ao posto dos correios original. O problema do Carteiro Chinês e suas variações formam uma classe dentro dos problemas de roteamento em arcos. O objetivo é encontrar um caminho que saia de um certo vértice-origem, percorra todos os arcos de um determinado grafo da forma mais econômica possível, retornando ao vértice-origem, criando um circuito que comece e termine em um mesmo vértice.

O problema do carteiro chinês é um problema de otimização combinatória clásssico com uma grante quantidade de aplicações práticas, tais como coleta de lixo urbano, inspeção em fios, retirada de gelo, entrega de cartas ou problemas semelhantes onde seja preciso percorrer uma cidade, um trecho de mapa ou uma rede de recursos. Tais aplicações se beneficiam das economias proporcionadas por um melhor uso do recurso em questão, seja ele uma pessoa, uma escavadeira ou um caminhão de lixo. Uma rota otimizada, na coleta de lixo urbano por exemplo, permite diminuir as janelas de coleta[2], economizando combustível, recursos humanos e melhorando a qualidade dos serviços.

Na versão clássica do CPP [Meigu, 1962], dado um Grafo G = (V,E), onde V é o conjunto de vértices e E é o conjunto de arestas, o objetivo é encontrar um circuito

[1] Este autor é referenciado em outros textos como Mei-Ku Kuan, Kwan Meiku, além de outras variações. Estas diferenças na grafia do nome do autor vem do fato de seu trabalho ter sido publicado apenas em chinês e que a transliteração de seu nome ao alfabeto latino aceite várias interpretações. A grafia *Meigu Guan* é citada em [Aminu & Eglese, 2006]

[2] *Janela de Coleta* é o tempo que um caminhão de lixo gasta percorrendo as ruas coletando lixo

de custo mínimo, que passe pelo menos uma vez por cada aresta de G. As arestas são percorridas, começando e terminando em um mesmo vértice. Existem versões onde as arestas são direcionadas ([Pearn et al., 1987], [Edmonds & Jonhson, 1973], [Malandraki & Daskin, 1993]), não-direcionadas ([Aminu & Eglese, 2006], [Pearn et al., 1987], [Wang & Wen, 2002], [Benavent et al., 1987]) ou mistas ([Benavent et al., 1987], [Sbihi & Eglese, 2007]).

Neste trabalho, será abordada a versão mista do CPP (MCPP), onde existem no grafo Arestas (que não são direcionadas) e Arcos (que fluem em apenas uma direção). No MCPP, temos um Grafo G = (V,E,A), onde V é novamente o conjunto de vértices, E são as arestas e A são os arcos, que são direcionados. O caso onde o grafo é completamente bidirecional (A = $\emptyset$), com apenas ruas de mão dupla por exemplo, foi estudado em detalhes e pode ser resolvido eficientemente [Edmonds, 1965]. O outro caso especial onde o grafo possui apenas arcos (E = $\emptyset$) também se provou ser de fácil resolução, em tempo polinomial [Edmonds & Jonhson, 1973]. O Problema do Carteiro Chinês misto foi provado como NP-Completo em [Papadimitriou, 1976].

1.1 Objetivo

O objetivo dessa dissertação é produzir um algoritmo heurístico de qualidade comparável aos algoritmos heurísticos de vanguarda para o problema, tanto em tempo de execução quanto ao custo da solução, para a versão mista do CPP. É proposta a construção de uma heurística baseada em colônia de formigas. Embora outras heurísticas tenham sido propostas, tais trabalhos focaram em necessidades específicas, como coleta de lixo em uma determinada cidade (Barcelona, em [Bautista et al., 2008]) ou outros tipos de heurísticas ([Corberán et al., 2002b], [Benavent et al., 1987], [Yaoyuenyong & Charnsethikul, 2002]). Serão usadas neste trabalho as instâncias utilizadas em [Corberán et al., 2007], onde os valores ótimos estão assinalados, permitindo uma análise sobre a qualidade da solução.

1.2 Alguns conceitos importantes

Um **grafo** misto G = (V,E,A) é formado por três conjuntos V, E e A. Os elementos de V são chamados de **vértices** ou **nós**. Os elementos de E são chamados de **arestas**. As duas pontas de uma aresta estão vinculadas a vértices e os conectam, na forma de um *par não-ordenado*. Se as duas pontas se conectam ao mesmo vértice, temos um *loop*.

Os elementos de A são chamados de **arcos**. Arcos são como arestas, com exceção de que a conexão entre seus dois vértices terminais é feita apenas em uma direção, através de um *par ordenado*, do vértice origem ao vértice destino.

Se o conjunto A é vazio, G é um grafo *não-direcionado*. Se o conjunto E é vazio, G é um grafo *direcionado*. Se os dois conjuntos forem não-vazios, temos um grafo *misto*. Um exemplo deste tipo de grafo está na figura 1.1. Nesse trabalho, será usado o termo *ligação* para arestas ou arcos, indiscriminadamente. Também é comum o uso do termo *enlace* na literatura, sendo ambos traduções do termo inglês *link*.

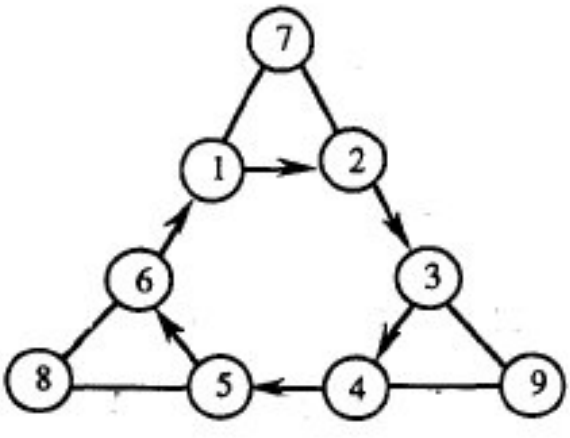

Figura 1.1. Exemplo de um grafo misto

Um *passeio* é uma sequencia de arestas e arcos que começa em um vértice qualquer, percorra uma das ligações desse vértice até o próximo vértice, e através da sucessão de percursos, retorna ao vértice original. Este passeio poderá ser chamado também de *ciclo* ou *circuito*. Os arcos são percorridos em obediência a suas orientações, ou seja, arcos são sempre percorridos do vértice origem ao vértice destino. Outros conceitos importantes em grafos podem ser encontrados em [Gross & Yellen, 2003].

1.3 Definição do problema

Seja um grafo misto G = (V,E,A), onde V é o conjunto de vértices, E é o conjunto das arestas e A, é o conjunto dos arcos . Cada aresta $e_{ij} \in E$ vai do vértice i até o vértice j e vice-versa. Cada $a_{ij} \in A$ vai apenas do vértice i para o vértice j. Todas as arestas e arcos possuem custos não-negativos. O Problema do Carteiro Chinês Misto (MCPP) consiste em achar um passeio de custo mínimo que percorra todas as arestas e arcos de G pelo menos uma vez. Em outras palavras, se o grafo G não é euleriano, o MCPP

é o problema de gerar este passeio passando pelo menos uma vez em cada aresta e arco de G de forma a minimizar o custo total de percurso do passeio que comece e termine no mesmo vértice.

Neste trabalho, serão usadas as seguintes notações:

- E_i : denotam o conjunto de arestas ligadas ao vértice i.
- A_i^+ : denotam o conjunto de arcos cujo vértice i seja o vértice de destino.
- A_i^- : denotam o conjunto de arcos cujo vértice i seja o vértice de origem.
- D_i^+ : é o grau de entrada do vértice i, que é a soma das arestas ligadas ao vértice mais os arcos que *chegam* a este vértice ($\left|E_i \bigcup A_i^+\right|$).
- D_i^- : é o grau de saída do vértice i, que é a soma das arestas ligadas ao vértice mais os arcos que *saem* deste vértice ($\left|E_i \bigcup A_i^-\right|$).
- D_i : é o grau do vértice i ($\left|E_i \bigcup A_i^+ \bigcup A_i^-\right|$).
- δ_{ij} : é o custo, ou distância, para percorrer a ligação entre os vértices i e j. Se esta ligação não existir, seu valor é ∞.

1.3.1 Contexto teórico

1.3.1.1 Teorema de Euler

O teorema de Euler diz que um grafo conexo e não-direcionado G = (V,E) contém um circuito Euleriano se e somente se D_i é par para todos os vértices $i \in V$. Edmonds e Johnson [Edmonds & Jonhson, 1973] mostraram que é possível converter um grafo G em euleriano transformando todos os vértices de grau ímpar em vértices de grau par repetindo algumas arestas de E com custo mínimo em tempo polinomial. Com esse novo grafo sendo euleriano, é possível encontrar também em tempo polinomial o circuito euleriano relacionado a este grafo. Em função disso, o problema do carteiro chinês clássico é polinomial.

1.3.1.2 Teorema de Edmonds e Johnson [Edmonds & Jonhson, 1973]

Um grafo direcionado fortemente conexo G = (V,A) contém um circuito euleriano, se e somente se, para todos os vértices $i \in V$, $D_i^+ = D_i^-$. Ou seja, se o grau de entrada de todos vértices é igual a seu grau de saída, respectivamente.

Um grafo direcionado é fortemente conexo se existe um caminho direcionado do vértice i até qualquer outro vértice pertencente a V. Edmonds e Johnson [Edmonds & Jonhson, 1973] também apresentaram um algoritmo polinomial que equilibra todos os vértices adicionando alguns arcos a G com custo mínimo, gerando um novo grafo euleriano. Como encontrar o circuito euleriano deste grafo é um problema polinomial, o problema do carteiro chinês direcionado tem solução polinomial.

1.3.1.3 Teorema de Ford e Fulkerson [Ford & Fulkerson, 1962]

Ford e Fulkerson provaram que um grafo G = (V,E,A) misto e fortemente conexo contém um circuito euleriano se e somente se, satisfaz duas condições:

- Para todo vértice $i \in V$, D_i é par.
- Para todo subconjunto de vértices S $\subseteq$ V, a diferença entre o número de arcos originários em S e com destino em vértices de V/S e o número de arcos originários em V/S com destino em vértices de S é menor ou igual ao número de arestas unindo vértices em S e V/S

Um grafo misto que satisfaz estas duas condições possui um circuito eueleriano que é a solução ótima para o MCPP.

Capítulo 2

Revisão bibliográfica

Neste capítulo, será revista boa parte da literatura sobre o Problema do Carteiro Chinês, tanto na forma de livros quanto em artigos que exploram seus diversos aspectos e variações. Em particular, o problema do Carteiro Chinês Misto (MCPP), por sua complexidade, possui extensa bibliografia, que será revista em seus pontos mais importantes.

O artigo de Meigu Guan [Meigu, 1962], marca a introdução do Problema do Carteiro Chinês na literatura de Programação Matemática. Este artigo define o problema e propõe um primeiro modelo baseado em restrições para o CPP. As restrições propostas no modelo estavam relacionadas á proibição de uma determinada orientação em algumas arestas, assim como a obrigatoriedade de um determinada direção em outras, conforme um modelo de escolhas baseadas no grau de cada vértice. A contribuição seguinte veio em 1973 através do artigo de Edmonds e Johnson [Edmonds & Jonhson, 1973]. Aqui, os autores fazem as primeiras relações entre modelos utilizados na geração de passeios eulerianos e CPP, além da definição do poliedro de soluções viáveis para esta classe de problemas. Estabelecem também certas definições sobre grafos direcionados, não-direcionadas e mistos e como estes afetam a solução do problema. É sugerido o uso dos modelos e algoritmos de passeios eulerianos para resolução do CPP.

2.1 Variações do problema

Em 1987, Benavent et al. [Benavent et al., 1987] fazem o primeiro uso de heurísticas para resolução de CPP, na versão onde o agente possui certa capacidade de transporte

e deve retornar ao depósito em busca de novo material de distribuição.

A primeira menção ao CPP com ordenação das arestas é feita em [Desrochers et al., 1991]. Este artigo propõe um modelo onde certas arestas devem ser visitadas antes das outras, por uma relação de precedência. Podemos pensar em relações de precedência como um primeira tentativa de produção de janelas de tempo.

Um dos trabalhos mais importantes desta classe de problemas vem de Malandraki e Daskin [Malandraki & Daskin, 1993], em 1993, onde é proposto o *Maximum benefit Chinese Postman Problem*. No CPP clássico, o objetivo do problema era a minimização de custos para o passeio. Neste artigo, a restrição de que todas as arestas devem ser percorridas é relaxada e cada aresta possui um custo e um lucro ao atravessá-la e uma penalidade se a aresta não for atravessada no passeio final. Esse problema é formulado como um problema de Programação Linear Inteira.

2.2 Carteiro Chinês Misto

Em [Papadimitriou, 1976], demonstrou-se que o MCPP é NP-Completo. Alguns anos depois, Frederickson [Frederickson et al., 1976] sugeriu um algoritmo aproximativo para o MCPP, com o nome de **Misto**, em grafos genéricos que amplia as técnicas usadas para grafos pares (com todos os vértices de grau par). Frederickson também mostrou que os algoritmos usados até ali tem um pior caso de 2 vezes a solução ótima. Entretanto, aplicando as heurísticas e selecionando a melhor solução, o pior caso vai a 5/3.

Em [Christofides et al., 1983], foi desenvolvido um algoritmo exato para MCPP. O Algoritmo é baseado essencialmente em branch-and-bound usando relaxações lagrangeanas. Minieka [Minieka, 1979] apresentou uma transformação do MCPP em um problema de redes-com-ganhos. A transformação permite a resolução ótima do problema usando programação linear e técnicas de planos de corte. Nobert e Picard [Nobert & Picard, 1996] também desenvolveram um algoritmo de plano de corte baseado em Programação Linear. Infelizmente, estes métodos são computacionalmente ineficientes e apenas problemas de ordem pequena ou moderada podem ser resolvidos ao ótimo. Outro método exato está contido em [Sherafat, 1988] e utiliza uma formulação interessante: todas as arestas são transformadas em *pseudo-ramos* de uma arborescência, com capacidades e um problema de fluxo é resolvido. Uma explicação

bem detalhada desse procedimento pode ser encontrada em [Negreiros et al., 2009].

Em 1999, Raghavachari e Veerasamy [Raghavachari & Veerasamy, 1999] propuseram uma modificação do algoritmo de Frederickson [Frederickson et al., 1976] com um pior caso de 3/2. Mais ainda, além de seu bom desempenho teórico, o Algoritmo Misto Modificado proposto é também competitivo na prática, especialmente em grafos com média a alta porcentagem de arcos, como demonstrado em Corberán et al [Corberán et al., 2002b], no qual uma heurística GRASP para o MCPP também é proposta.

Pearn e Liu [Pearn & Liu, 1995] propuseram uma heurística baseada no procedimento proposto por Edmonds e Johnson, com excelentes resultados computacionais em relação ao tempo. Pearn e Chou [Pearn & Chou, 1999] evoluíram o artigo anterior, com alterações nas heurísticas propostas que melhoraram as implementações, ganhando desempenho. Em [Laporte, 1997], as transformações propostas por [Dror, 2000] são utilizadas, convertendo problemas de roteamento em arcos em problemas de roteamentos em vértices equivalentes, como o *Problema do Caixeiro Viajante*.

Angel Corberán e seus alunos fizeram muitas contribuições relevantes ao MCPP, comparando diferentes formulações matemáticas para o problema [Corberán et al., 2006] e estudando a versão rural do problema [Corberán et al., 2002a], entre outras contribuições importantes. Corberán também gerou um conjunto de instâncias de teste em [Corberán et al., 2007], utilizadas em vários artigos posteriores, assim como neste trabalho.

O algoritmo de Branch-and-Cut proposto em [Corberán et al., 2007] é capaz de resolver instâncias de MCPP. 17 de 24 instâncias com 3000 vértices e mais de 9000 ligações foram resolvidas até a otimalidade sem o uso de heurísticas que produzam *upper bounds* iniciais.

Em [Yaoyuenyong & Charnsethikul, 2002], é proposta uma heurística onde a solução inicial é construída com um algoritmo de fluxo de custo mínimo. Na fase de otimização, o algoritmo reorienta algumas arestas e deleta arcos redundantes. Neummann [Neumann, 2008] utiliza algoritmos evolucionários em busca de boas soluções, devido ao caráter NP-completo do problema. [Bautista et al., 2008] utiliza uma nova abordagem, algoritmos de colônias de formigas, para resolver o MCPP, adicionando ao problema restrições de capacidades e outras restrições ligadas ao problema prático de coleta de lixo em uma cidade na aréa metropolitana de Barcelona.

Os artigos de [Benavent et al., 1992] e [Bodin et al., 1983] ilustram a quantidade de conhecimento construída na aréa de roteamento em arcos nos últimos 27 anos. [Negreiros et al., 2009] é a compilação mais recente sobre o assunto, focando nas versões direcionada, não direcionada e mista do CPP, apresentando o software XNÊS, que inclui um gerador de grafos para as versões citadas, o algoritmo de [Sherafat, 1988], e um algoritmo GRASP chamado **Combinado&Híbrido**.

É interessante observar que Yan e Thompson em [Yan & Thompson, 1998] modificam a formulação proposta por Win [Win, 1987] em sua tese de doutorado, transformando o MCPP em um problema de cobertura de conjuntos, propondo um algoritmo exato e uma heurística para a solução do problema, assim como [Lee & Wakabayashi, 2002]. Ambos são baseados em algoritmos branch-and-bound desenvolvidos para a solução de problemas de cobertura de conjuntos, particionamento e empacotamento.

Capítulo 3

A heurística proposta

Neste capítulo, será proposta a heurística usada para a solução do problema do carteiro chinês e os fundamentos desta heurística. A heurística proposta se baseia na meta-heurística de colônias de formigas. Computacionalmente, a meta-heurística cria um conjunto de elementos, um *banco de circuitos*, que serão usados na construção de soluções semi-aleatórias, ou seja, que possuem um caráter aleatório guiado por um parâmetro de distribuição, em busca da melhoria da solução corrente para o problema. Essas soluções são analisadas e sua qualidade é levada em consideração na geração de novas soluções, através de parâmetros probabilísticos, gerados pela qualidade das soluções anteriores. Essas fases se repetem até que um critério de parada (de tempo ou qualidade da solução) seja atingido.

Para melhor entendimento, será feita uma breve introdução ás heurísticas baseadas em colônias de formigas e em seguida, a aplicação destas ao problema do carteiro chinês.

3.1 Colônias de Formigas

Otimização por colônia de formigas (OCF ou ACO, do inglês Ant Colony Optimization) é uma meta-heurística baseada em população que pode ser usada para encontrar soluções aproximadas para problemas difíceis de otimização.

Em OCF, um conjunto de agentes (formigas artificiais) procuram por soluções para um certo problema de otimização. As formigas artificiais constroem de forma incremental soluções através da agregação de elementos ou conjuntos de elementos do grafo. O processo de construção das soluções é estocástico e guiado pelo modelo de

feromônios, que é um conjunto de valores associados com as componentes do grafo, sejam vértices ou arestas, e que sofrem alteração durante o trabalho das formigas artificiais.

3.1.1 Um exemplo - O problema do caixeiro viajante (PCV)

A forma mais fácil de se entender como uma OCF funciona é através de um exemplo, utilizado em [Dorigo, 1992], a tese de doutorado de Marco Dorigo que propôs inicialmente esta meta-heurística. Considere o problema do Caixeiro Viajante. No PCV, um conjunto de cidades e distâncias entre estas são dados. O problema consiste em achar um circuito de custo mínimo que visite cada cidade apenas uma vez.

Para que se aplique OCF no PCV, associe os vértices de um grafo com o conjunto de cidades. Este grafo é chamado *Grafo de construção*. Considere também que as distâncias entre as cidades estão representadas por arestas e que estas arestas tem valores heurísticos e feromônicos associados. Os valores de ferômonios são atualizados em tempo de execução e representam a experiência acumulada da colônia de formigas, enquanto valores heurísticos são dependentes do problema e que no caso do PCV, são inicializados com o valor do inverso das distâncias ($\frac{1}{\delta_{ij}}$).

As formigas constroem as soluções da seguinte forma: Cada formiga começa em uma cidade aleatória (um vértice do grafo de construção) e a cada passo da construção, se move pelas arestas. Cada formiga mantém a memória de seu caminho e, nos próximos passos, evita cidades já visitadas. Uma formiga acabou a construção de uma solução uma vez que tenha visitado todos os vértices do grafo. A cada passo da construção, a formiga sorteia, dentro de uma distribuição de probabilidades, a aresta a ser seguida dentre aquelas que levam a vértices não-visitados. A distribuição probabilística é guiada pelos valores ferômonicos e pela informação heurística: Quanto mais ferômonio e maior o valor heurístico associado á aresta ($\frac{1}{D_{ij}}$), maior a probabilidade de que essa aresta seja escolhida. Assim que todas as formigas completam seus circuitos, os ferômonios são atualizados. Cada um dos valores ferômonicos é decrescido de uma porcentagem (Evaporação). Cada aresta então recebe uma quantidade adicional de feromônios proporcional á qualidade da solução á que pertence.Este procedimento é repetido até que um critério de parada seja satisfeito.

3.1.2 A inspiração na natureza

No experimento da ponte dupla, um formigueiro é conectada a uma fonte de alimento por duas pontes. As formigas podem atingir a comida e trazê-la ao ninho através das duas pontes. O objetivo do experimento é observar o comportamento da colônia. Observa-se que se as duas pontes tem o mesmo tamanho,, as formigas tendem a convergir ao uso de apenas uma. Se o experimento é repetido um certo número de vezes, nota-se que cada ponte é usada em 50% das vezes. Estes resultados explicam-se pelo fato de que as formigas, ao se movimentarem, depositam feromônios no solo, e independentemente do caminho seguido, sua escolha é guiada pelos ferômonios deixados pelas outras formigas:quanto mais feromônio em um caminho em particular, maior a probabilidade de que aquele caminho seja escolhido.

Considere o caso em que as duas pontes tem o mesmo tamanho. A figura 3.1 pode ser usada para ilustrar o experimento.

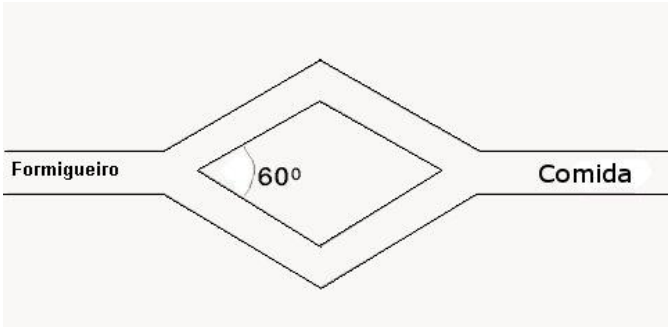

Figura 3.1. Caminhos de tamanhos iguais

No começo do experimento, as formigas exploram as vizinhanças do formigueiro. Quando chegam ao ponto de decidir qual das pontes seguir, elas escolhem probabilisticamente, com a distribuição da probabilidade guiada pelos ferômonios em cada uma das pontes. Inicialmente, cada formiga escolhe uma das pontes com 50% de probabilidade, já que não há feromônio em nenhuma delas. Entretanto, após algum tempo e em função das flutuações aleatórias, uma das pontes começa a apresentar uma maior concentração de feromônios e, portanto, atrai mais formigas, o que aumenta o nível de ferômonio. Este ciclo virtuoso faz com que a colônia use apenas uma das pontes.

Se uma das pontes é significantemente menor que a outra, um segundo mecanismo participa: as formigas que aleatoriamente escolhem o caminho mais curto são as primeiras a chegarem ao alimento. Quando estas formigas voltam e encontram o ponto

de decisão 2 (Figura 2), encontram um nível mais alto de feromônios na ponte mais curta, que então é escolhida com maior probabilidade e recebe novamente feromônio adicional. Este fato aumenta a probabilidade de que as próximas formigas escolham este caminho em detrimento do outro.

Existe um modelo desenvolvido por [Goss et al., 1989] para este comportamento. Assumindo que em um dado momento,f_i formigas passaram pela i-ésima ponte ,a probabilidade de que uma formiga escolha a ponte i é:

$$p_i = \frac{m_i + k^h}{\sum_{j=1}^{B} m_j + k^h} \tag{3.1}$$

onde B é o número de pontes e os parâmetros k e h são próprios de cada experimento. Simulações de Monte Carlo encontraram como valores adequados $k \approx 20$ e $h \approx 2$.

Esta equação inspirou a equação usada no primeiro algoritmo de OCF ([Dorigo, 1992]).

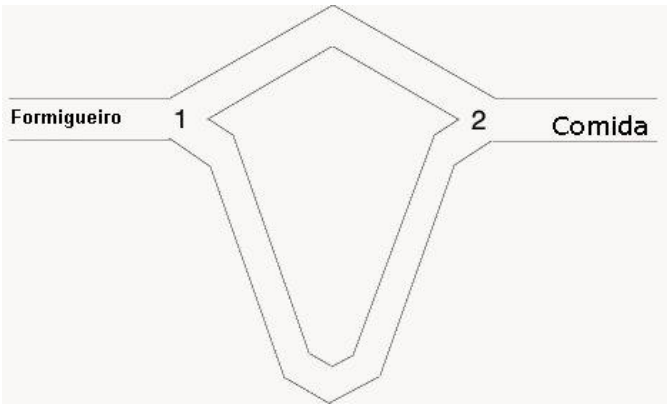

Figura 3.2. Caminhos de tamanhos diferentes

3.2 Fundamentação teórica da cobertura de um grafo por circuitos

Um *circuito* **C** pode ser definido como um subconjunto ordenado de arestas e arcos de G, onde o vértice de origem da primeira aresta/arco coincide com o vértice de destino da última aresta/arco. C_k é o k-ésimo arco ou aresta de C.

Lema 3.2.1 *Para todo grafo* G *misto não-euleriano e fortemente conexo, todo circuito* C *de* G *contém pelo menos um vértice* v *com uma aresta incidente não contida em* C *ou um arco com origem em v e não contida em C.*

Prova Seja C um circuito arbitrário. Como G é não-euleriano, existe uma ligação $\ell \notin$ C. Se C passa por todos os vértices de G, ℓ é incidente ou tem origem em um vértice de C e o teorema está provado. Caso contrário, seja v um vértice arbitrário tal que v não seja visitado por C. Considere um caminho de um vértice arbitrário u visitado por C para v, cuja existência é garantida pelo fato de que G é fortemente conexo. A última ligação do caminho u $\rightarrow$ v não pertence a C, pois v não é visitado por C. A primeira ligação deste caminho u $\rightarrow$ v não pertencente a C é evidentemente uma aresta incidente ou um arco com origem em u, provando o teorema.

O problema da cobertura de um grafo por circuitos (PCC) é NP-Díficil e definido como: Dado um grafo misto G fortemente conexo, deve ser encontrado um conjunto de circuitos de custo mínimo tal que cada aresta e arco de G esteja presente em pelo menos um circuito. Em [Lee & Wakabayashi, 2002], a relação entre o problema do carteiro chinês e a cobertura de um grafo por circuitos é discutida através de uma revisão da literatura sobre o problema. É provado que os problemas são equivalentes e possuem solução comum e bijetiva. De fato, a conclusão de que os problemas são relacionados é bem instintiva e deriva de um senso comum: a solução esperada para o MCPP é um passeio pelo grafo, que contenha todas as arestas e arcos e que comece e termine no mesmo vértice. De forma rigorosa, é necessário provar que esta cobertura por circuitos com duplicações de fato gera uma solução para o MCPP.

Teorema 3.2.2 *A solução da cobertura das ligações mínima de um grafo misto por circuitos é equivalente a solução do problema do Carteiro Chinês Misto, em um grafo fortemente conexo.*

Prova É preciso provar que podemos transformar a solução de um MCPP em um problema de cobertura por circuitos e vice-versa.

MCPP $\rightarrow$ PCC

A solução para o MCPP é naturalmente um circuito que contém todas as ligações e é mínimo. Portanto, este circuito é a solução ótima para o problema de cobertura mínima.

PCC → MCPP

A solução do problema de cobertura é um conjunto de circuitos de custo mínimo. É necessário provar que estes circuitos podem ser percorridos de forma a criar um único circuito. Assim, um circuito C_1 é escolhido deste conjunto. Do circuito C_1, um vértice v_{12} é escolhido, de forma tal que v_{12} esteja contido em outro circuito, C_2. Unindo esses dois circuitos, é possível imaginá-los como um só, pois partindo de um vértice qualquer de C_1, este é percorrido até o vértice V_{12} de intercessão, daí durante o circuito C_2 até o vértice de intercessão e finalmente, de volta ao vértice de origem pelo circuito C_1. Este processo pode ser repetido até que não hajam mais circuitos não inclusos na solução. Este processo está exemplificado na figura 3.3.

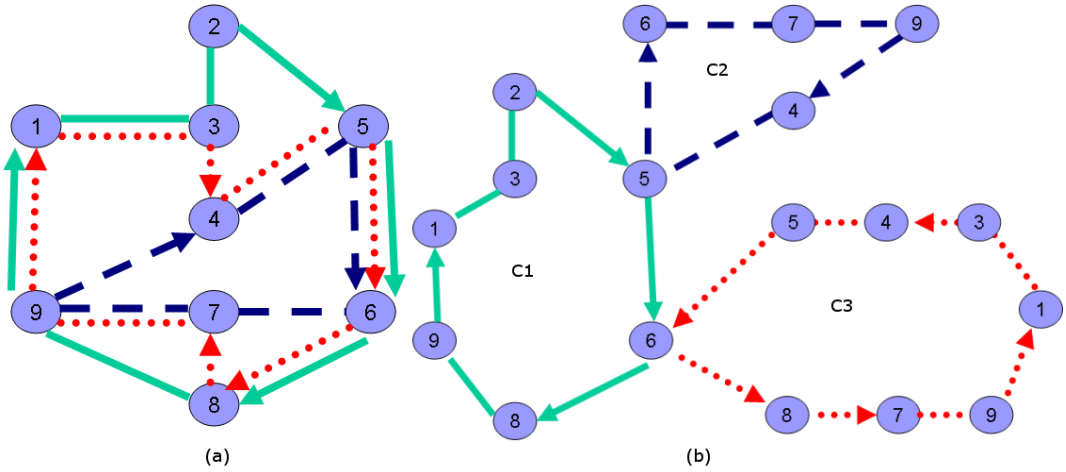

Figura 3.3. Exemplo para a prova

A figura 3.3a contém 3 circuitos, uma solução para o problema de cobertura. A figura 3.3b mostra os mesmos 3 circuitos, porém destacados e mostrando seus pontos de intercessão. Partindo do vértice 9 do circuito C_1, por exemplo, é possível gerar o circuito baseado em vértices 9-1-3-2-5-6-7-9-4-5-6-8-7-9-1-3-4-5-6-8-9.

3.3 Construção

A heurística começa com a geração de um banco de dados de circuitos, não-exaustivo, onde cada ligação (aresta ou arco) esteja presente em pelo menos um circuito. Este banco de dados pode ser gerado de qualquer forma, desde que seja possível conhecer todas as ligações que pertencem a um circuito e seu custo total.
Nesse trabalho, o banco de dados foi gerado por três métodos, detalhados abaixo, sempre com o objetivo de contemplar cada ligação com uma diversidade mínima, que permita uma execução produtiva da otimização da colônia de formigas.

3.3.1 Circuitos de tamanho definido

Este método de geração consiste de uma *busca em profundidade* (DFS, ou *Depth-First Search*). O grafo é percorrido em profundidade, por tentativa e erro, e se o vértice original é atingido em K passos, temos um circuito de tamanho definido K, que é armazenado no banco de circuitos. Caso contrário, é feito um *backtracking* e o algoritmo continua enquanto houverem caminhos não explorados.

No algoritmo 1, L é o conjunto de ligações e N é o conjunto de vértices de G.

```
1  Procedimento CircuitosTamanhoK (TamanhoK) ;
2  para cada n_i ∈ N faça
3  |  DFS(n_i,n_i, TamanhoK,1, -1, [lista vazia]);
4  fim para cada
5  Procedimento DFS (NoAtual, NoProcurado, Tamanho, Nivel,
   ArestaOrigem, Circuito) ;
6  para cada l ∈ L incidente a NoAtual faça
7  |  se (l != ArestaOrigem) então
8  |  |  Coloque l no fundo da lista Circuito; ;
9  |  |  se (Nivel = Tamanho) e (NoDestino(l) = NoProcurado) então
10 |  |  |  Armazene Circuito no banco de dados e retorne;
11 |  |  fim se
12 |  |  se Nivel < Tamanho então
13 |  |  |  DFS(NoDestino(l),NoProcurado, Tamanho,Nivel+1, l, Circuito);
14 |  |  fim se
15 |  fim se
16 fim para cada
17 Retorne ;
```

Algoritmo 1: Circuitos de tamanho definido K

O procedimento *CircuitosTamanhoK* (linhas 1-4) recebe o tamanho dos circuitos a serem gerados e para cada vértice do grafo, dá início ao DFS partindo desse vértice. No laço da linha 6, todas as ligações incidentes ao vértice atual são explorados. O teste da linha 7 é feito para que o DFS não volte pela Aresta de onde chegou. Isto só é permitido nos circuitos de tamanho 2, que são calculados de forma mais prática: basta incluir como circuitos todas as arestas e arcos paralelos de sentidos opostos. A linha 8 empilha a ligação percorrida na lista de ligações que irão compor o circuito. O teste da linha 9 verifica se o vértice atingido pela ligação percorrida é o nó final desejado com o número de K passos definido. Se for, o circuito é armazenado no banco de dados e o procedimento é abandonado. Caso contrário, o teste da linha 12 serve para verificar se ainda existem passos a serem executados e faz uma chamada recursiva á própria função, partindo do vértice atual, atingido pela ligação percorrida. A figura 3.4 ilustra alguns tipos de circuitos de tamanho definido. No grafo de exemplo, são criados os caminhos representados pelas arestas e arcos mais grossos. Os circuitos de tamanho 2 são as próprias arestas (ida e volta pela mesma aresta) e ligações paralelas (ida por uma ligação e volta por outro).

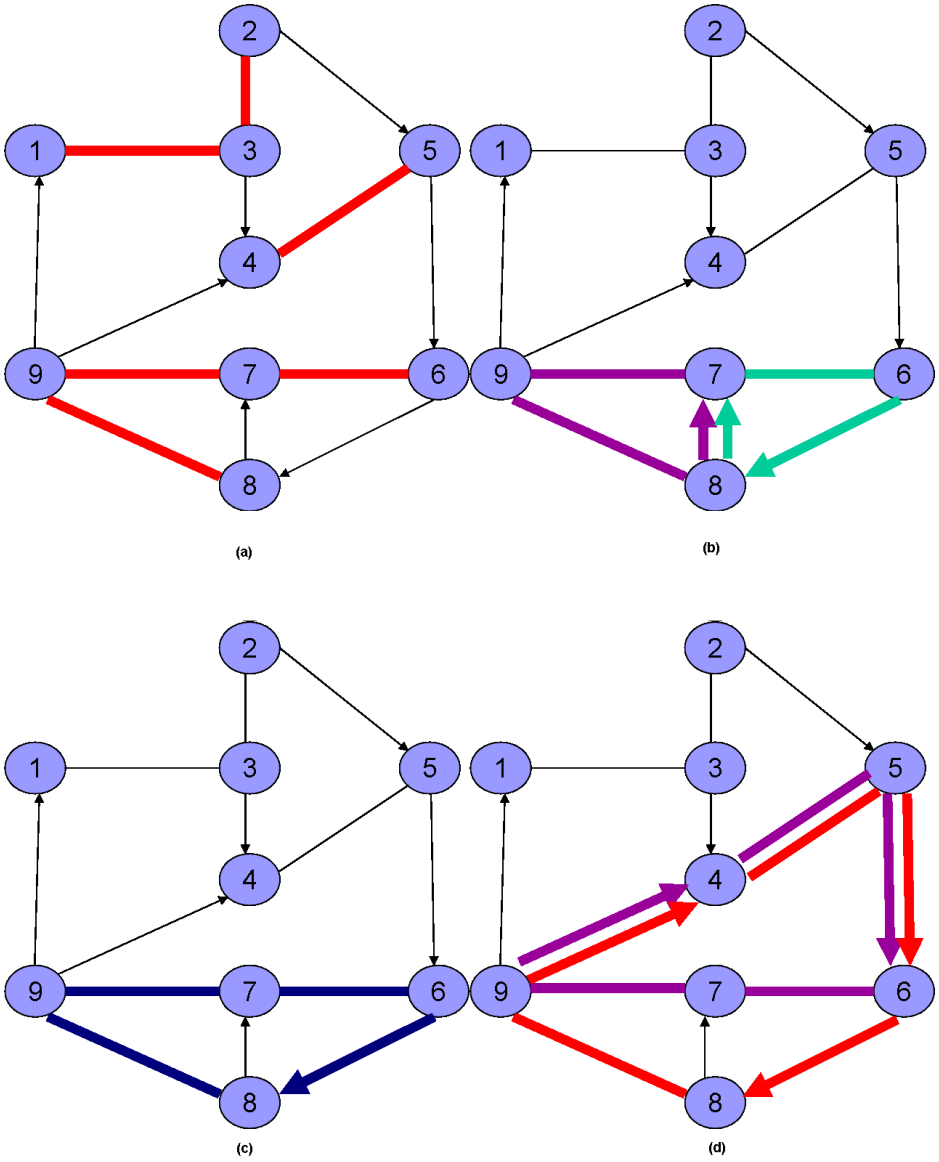

Figura 3.4. Exemplos de circuitos de tamanho 2(a),3(b),4(c) e 5(d)

3.3.2 Circuitos aleatórios

Este método também se baseia em uma busca em profundidade modificada. Uma ligação é escolhida por não pertencer a nenhum circuito ou apenas aleatoriamente, com fins de diversidade. Essa ligação é percorrida, atingindo um vértice. Este vértice recebe um número, que é o passo atual do DFS e uma ligação incidente é sorteada e percorrida.

Se não houverem mais arestas não usadas no vértice atual, volta-se ao vértice de origem até que se encontre uma nova saída. O circuito é encontrado quando um vértice visitado é atingido. O número encontrado no vértice representa o número de ligações

a serem descartadas do começo da lista temporária de ligações.

```
1  Procedimento GeraCircuitoAleatorio(Ligacao) ;
2  para cada N_i ∈ G faça
3  |  N_i.flag = -1;
4  fim para cada
5  Circuito ← [lista vazia] ;
6  N ← DFS2(Ligacao, 1, Circuito) ;
7  se Circuito != [lista vazia] e N != -1 então
8  |  Armazene Circuito no banco de dados a partir do item N ;
9  fim se
10 Função DFS2 (Ligacao, Nivel, Circuito) ;
11 se Nivel = 1 então
12 |  NoOrigem(Ligacao).Flag ← 0;
13 fim se
14 NoAtual ← NoDestino(Ligacao) ;
15 se NoAtual.flag != -1 então
16 |  Retorne NoAtual.flag;
17 fim se
18 NovaAresta ← SorteiaUmaAresta(NoAtual) ;
19 se NovaAresta = -1 então
20 |  Apague ultimo item da lista Circuito ;
21 |  Retorne -1;
22 fim se
23 Ligacao.usada ← Verdadeiro ;
24 Coloque Ligacao no fundo da lista Circuito ;
25 NoAtual.flag ← Nivel ;
26 Retorne DFS2(NovaAresta, Nivel+1, Circuito) ;
```

Algoritmo 2: Circuitos Aleatórios

No laço da linha 2 do algoritmo 2, todos os vértices são marcados com um valor de flag -1, representando que estes vértices nunca foram visitados. Em seguida, nas linhas 5 e 6, a função DFS2 é chamado, retornando o Circuito, caso encontrado e um valor N, que representa o flag do vértice que foi usado como condição de parada. Pela condição da linha 7, se a lista Circuito não for vazia, armazena-se o pedaço da lista Circuito do N-ésimo item até o final. Dentro da função DFS2 (linhas 10-26), ocorrerá uma variação de busca em profundidade, recebendo como parâmetro a Ligação de onde começará o circuito, o Nível atual de recursão e a Circuito formado até o momento. Caso este seja o primeiro nível de recursão (linha 11), o vértice de origem[1] é considerado

[1]Se a ligação é uma aresta, como dizer qual dos vértices é origem ou destino? A função NoOrigem, quando invocada pela primeira vez nesse contexto, sorteia uma direção para aresta e assim, induz os vértices de origem e destino. Assim, NoDestino, quando invocada, retorna exatamente o outro vértice da ligação

como visitado. Na linha 14, a ligação é percorrida. Caso o vértice de destino já tenha sido visitado (condicional da linha 15), um circuito foi formado e a função retorna em que ponto esta ligação se encontrou com o circuito, através do flag do vértice atingido. Caso contrário, uma nova ligação entre as não-utilizadas pelo circuito é sorteada (linha 18). Se o vértice não possuir uma ligação que permita abandoná-lo (condicional da linha 19), a função retorna -1, sinalizando erro e a incapacidade de formar um circuito. Ao final do algoritmo 2, a ligação é sinalizada como utilizada (linha 23), colocada no fundo da lista Circuito (linha 24) , o vértice de destino é marcado como visitado (linha 25) e o próximo nível de recursão é invocado, com a ligação sorteada e a lista Circuito atualizada.

3.3.3 Circuitos por Dijkstra

Este método se baseia na certeza de que o grafo G é fortemente conexo. Para cada ligação de interesse, é calculado o caminho de melhor custo (excluída a própria ligação) entre seu vértice de destino e seu vértice de origem. É possível afirmar que o caminho calculado pode substituir aquela ligação em qualquer circuito onde esta apareça. Ou ainda, junto com a ligação, forma um circuito. Tal procedimento é útil quando após os outros processos, ainda existam ligações que não pertençam a nenhum circuito.

```
1 para cada l ∈ L_1 tal que L_1 ⊆ L faça
2 |   Circuito ← [l] ;
3 |   Caminho ← CaminhoMaisCurto(NoDestino(l), NoOrigem(l), l) ;
4 |   Coloque Caminho no fundo da lista Circuito ;
5 |   Armazene Circuito no banco de dados;
6 fim para cada
```

Algoritmo 3: Circuitos por Dijkstra

No laço da linha 1, l é um subconjunto qualquer das ligações do grafo (L). A lista Circuito, que armazenará a lista de ligações é inicializada (linha 2) com a própria ligação. A seguir , na linha 3, o método de Dijkstra é utilizado na função CaminhoMaisCurto, que retorna a lista de ligações que formam o caminho mais curto entre o vértice de destino de l e seu vértice de origem, excluindo-se o caminho trivial que é percorrer de volta a própria ligação, caso ela seja uma aresta.

Adicionalmente, os circuitos gerados por este método são perturbados com a substituição de ligações aleatórias por caminhos alternativos.

A figura 3.5 a seguir demonstra a criação de circuitos através do uso de Dijkstra. Após a execução dos outros métodos anteriormente citados, sobraram os arcos que vão do vértice 9 a 1, do vértice 3 ao 4 e do vértice 2 ao 5 (Grafo a). Na primeira execução, é

gerado um circuito contendo os dois primeiros (Grafo b). Com a segunda execução, é gerado o segundo circuito (Grafo c).

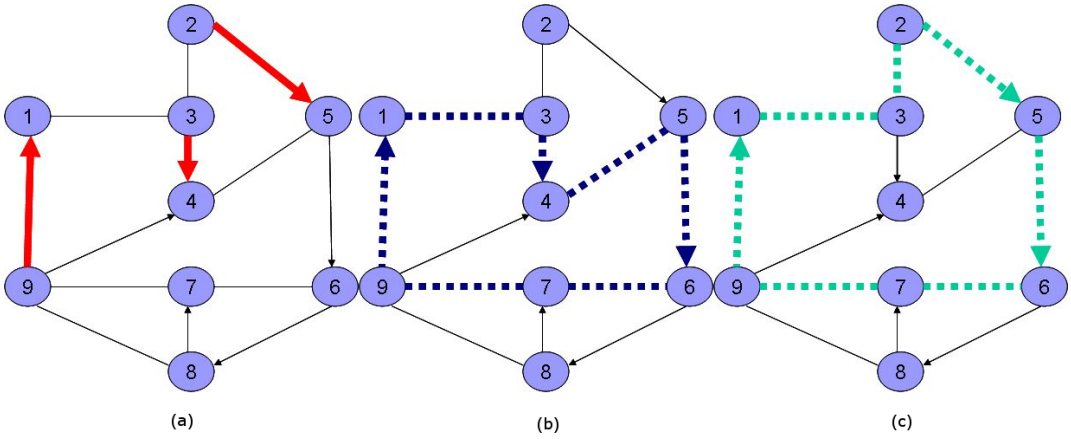

Figura 3.5. Exemplos de circuitos gerados por Dijkstra

3.3.4 Tratamento do banco de dados de circuitos

Após a execução desses métodos, o resultado será um banco de dados de circuitos onde cada ligação pertence a pelo menos um circuito. Entretanto, é bem provável que métodos diferentes gerem os mesmos circuitos. Tal situação é indesejada, pois alterará a distribuição probabilística na escolha do circuito a ser usado pelas formigas. Se um determinado circuito foi inserido várias vezes no banco de dados, a probabilidade desse circuito ser escolhido aumenta proporcionalmente ao número de inserções, assim como uma pessoa ter seu nome inscrito em um mesmo sorteio várias vezes aumenta a chance de que ela seja sorteada.

Portanto, faz-se necessária um tratamento dos circuitos desse banco. Circuitos iguais, que difiram apenas em relação á ligação inicial, devem ser eliminados, sob pena de atrapalharem a distribuição dos circuitos nas soluções. Para isso, seria ideal produzir um método que calcule, para cada circuito, um valor *hash*, identificando unicamente cada circuito e que circuitos iguais produzam sempre um valor *hash* igual, facilitando a eliminação das duplicações.
Um circuito é definido pela ligações que o formam e sua ordenação dentro do circuto e um método que calcule um valor hash certamente deve levar isso em conta. Cada ligação do grafo possui um número único que a identifica. Uma das formas intuitivas de se implementar uma função de hash (que será chamada **Método 1**) seria somando os

identificadores das ligações pertencentes a cada circuito , ou ainda, caso os valores ocupem muito espaço na memória, somando as diferenças entre identificadores de ligações (**Método 2**). Também é possível calcular um número de ponto flutuante identificador do circuito (**Método 3**), somando o valor da divisão de cada identificador de ligação pelo seguinte. Na Tabela 3.1, os três métodos estão expostos para casos específicos, em busca de suas características. Para 6 circuitos fictícios, estão calculados os 3 métodos, assim como seus identificadores.

Tabela 3.1. Análise de circuitos

#	**Custo**	**Tam.**	**Ident. das Ligações**	**Método 1 Soma Lig.**	**Método 2 Soma Difer.**	**Método 3 Ponto Fl.**
1	1500	5	45,76,123,55,78	377	202	5,8848166
2	1500	4	32,50,97,44	223	130	4,7350093
3	1500	5	123,55,78,45,76	377	202	5,8848166
4	3000	4	12,88,33,23	156	152	6,1544795
5	3000	4	12,88,30,26	156	152	6,3902097
6	1500	4	64,100,194,88	446	260	4,7350093

No circuito 2, por exemplo, calculamos por cada um dos métodos:

- **Método 1**: $32 + 50 + 97 + 44 = 223$
- **Método 2**: $|32 - 50| + |50 - 97| + |97 - 44| + |44 - 32| = 130$
- **Método 3**: $\frac{32}{50} + \frac{50}{97} + \frac{97}{44} + \frac{44}{32} = 4,7350093$

Observe que os métodos 1 e 2 de fato detectam circuitos iguais, com ligações iniciais diferentes. Porém, também consideram erroneamente como iguais os circuitos 4 e 5. Isso ocorre porque os dois métodos levam em conta apenas o valor absoluto dos identificadores das ligações e não sua posição relativa umas ás outras. O método 3 fornece uma identificação melhor que os outros métodos, mas também poderia incorrer em erro se os identificadores das ligações de um circuito fossem exatamente o dobro de um outro circuito, como nos circuitos 2 e 6. O ideal é utilizar um ou mais desses cálculos ,computacionalmente simples, e que economizarão tempo na fase de melhoria. este trabalho se utilizou dos valores gerados peos métodos 1 e 3.

Com esses números identificadores em mãos e o custo total calculado durante a produção do circuito, a probabilidade que esse identificador e o custo do circuito se repitam para circuitos diferentes é praticamente zero, atingindo zero em instâncias

reais. Agora, basta ordenar o banco de dados por esses três valores e excluir os casos de duplicidade, percorrendo-o do primeiro circuito ao último.

3.4 Melhoria - Otimização por Colônia de Formigas

A partir da obtenção do banco de dados construído pelos métodos citados, é possível começar a otimização por colônia de formigas. É bastante intuitivo que a qualidade dos circuitos presentes no banco influirão de forma importante na qualidade da solução encontrada.
Como o banco de dados está representado como uma lista de circuitos e ligações que os formam, é necessário explicitar também a relação inversa. Ou seja, que a relação entre as ligações e a que circuitos fazem parte, a partir da ligação, também seja acessada facilmente, já que esta é analisada a todo momento. Além disso, é importante que a lista de ligações ainda não inclusas na solução seja facilmente atualizada. Tais estruturas serão abordadas no momento em que forem citadas durante a heurística.

3.4.1 Inicialização

O primeiro passo consiste em carregar na memória o banco de dados de circuitos. Um vetor *vCircuitos* contendo os circuitos, com seus custos e ligações, é carregado. Alternativamente e conforme o tamanho do banco de dados de circuitos, pode-se usar uma técnica, onde o vetor contém apenas os custos e um ponteiro para a posição do circuito no arquivo. Nesse caso, toda vez que um circuito é lido, é feito um acesso ao arquivo em disco, impactando na velocidade do método.
Em um vetor **vFeromonios** com o tamanho da banco, será armazenada a quantidade atual de feromônios em cada circuito. Inicialmente, todos os circuitos tem a mesma quantidade de feromônio (valor de referência:1), para que a distribuição probabilística seja igual. Inicializa-se o valor ∞ para o custo da melhor solução encontrada até o momento.
Através do vetor *vCircuitos*, é possível explicitar a relação Circuito $\rightarrow$ Ligações, de forma que a partir de um circuito, é possível saber todas as ligações que o formam. De fato, esta é a forma original do banco de dados. No entanto, é necessária uma indexação para que seja possível chegar aos circuitos através das ligações (Ligação $\rightarrow$ Circuitos). Antes da carga do banco de dados de circuitos, um vetor **vLigações** de tamanho $|E| + |A|$ é declarado e conforme cada circuito é lido do arquivo, este circuito é incluído na lista referente as ligações nele contidos. Sendo assim, a k-ésima posição de vLigações contém uma lista de todos os circuitos que possuem a ligação k.

3.4.2 Gerando um Jardim

Um *Jardim* é um conjunto de formigas que são criadas e cujo sorteio de ligações ocorre sobre uma mesma distribuição probabilística, baseada na quantidade de feromônios acumulada nos circuitos pelos jardins anteriores. No primeiro jJardim, todos os circuitos tem a mesma probabilidade de serem escolhidos como parte de uma solução, pois foram inicializados como mesmo valor. Entretanto, dentro de um mesmo jardim, os dados heurísticos provocarão mudanças na distribuição probabilística de acordo com a conveniência em adicionar ou não um determinado circuito.Por definição, cada formiga em um mesmo jardim é independente das outras, fazendo suas escolhas de circuitos e arestas sem preocupação com outros resultados já encontrados por outras formigas no mesmo jardim.
Gera-se uma lista com todas as ligações do grafo e uma delas é sorteada, por exemplo, a ligação L_{22}. No vetor vLigações, na 22^o posição, estão todos os circuitos que contém esta aresta. Um circuito é sorteado conforme a seguinte distribuição probabilística:

$$p_i = \frac{f_i + a_i}{\sum_{j=1}^{B} f_j + a_j} \tag{3.2}$$

onde :

- p_i é a probabilidade de que o i-ésimo circuito da ligação L_{22} seja escolhido.
- B é o número de circuitos que a ligação L_{22} está contida (que ainda não fazem parte da solução).
- f_k é a quantidade de feromônio depositada no k-ésimo circuito da ligação L_{22}.
- a_k é o coeficiente de aproveitamento do k-ésimo circuito da ligação L_{22}. É calculado pela fórmula $1 - \frac{U_k}{C_k}$, com U_k sendo o custo total das ligações já presentes na solução e que pertençam ao k-ésimo circuito e C^k, o custo total do k-ésimo circuito. O objetivo deste coeficiente é beneficiar circuitos que tenha ligações formando pedaços maiores que ainda não entraram na solução.

Com o k-ésimo circuito sorteado, este é adicionado á solução, com seu custo somado ao custo total. Todas as ligações deste circuito saem da lista de ligações a serem adicionadas e os circuitos que contém cada uma delas tem seu contador de uso de ligações adicionado em uma unidade, através do vetor vLigações, estrutura de dados única de cada formiga. Isto é feito de forma a facilitar o cálculo do coeficiente a_k de uso. Uma nova ligação da lista de pendentes é sorteada e o processo se repete até que

todas as ligações estejam representadas na solução. Uma solução é gerada por uma formiga, e um conjunto dessas formigas formam um jardim. Na heurística proposta, o número de formigas por jardim pode variar e sua configuração é discutida no capítulo de Resultados Computacionais.

3.4.3 Aplicando feromônios

Ao fim da execução de cada Jardim, é preciso atualizar os feromônios associados a cada circuito do banco de dados. Essa atualização é feita de duas formas: *Evaporação* e *Impregnação*, ambos derivados de seus conceitos naturais. *Evaporação* é a retirada ou subtração de feromônio de um circuito de forma que as soluções mais antigas e fora de uso sejam gradualmente colocadas de lado. Já a *Impregnação* se refere ao acréscimo de feromônio nos circuitos, em quantidade proporcional á qualidade das soluções em que este participe.

No inicio de cada Jardim, multiplica-se todo o conjunto de feromônios por um fator ϵ menor que 1, simulando a degradação natural dos rastros das formigas reais. A diferença (1-ϵ) é o fator de degradação e a configuração desse fator também é discutida no capítulo de Conclusão. Cada solução gerada por uma formiga tem um custo final γ e ao fim de cada Jardim, calcula-se a média dos custos totais das formigas (γ_m). Todos os circuitos de cada formiga são atualizados conforme a seguinte formula:

$$f_i = f_i * \frac{\gamma_m}{\gamma_i} \tag{3.3}$$

onde f_i é a quantidade de feromônio depositada no k-ésimo circuito do banco de dados. Sendo assim, a quantidade de feromônios é aumentada ou diminuída em função da relação entre o custo médio do Jardim e o custo da formiga. Além disso, é dado um bônus de 20% nos feromônios dos circuitos contidos em formigas cujo custo γ seja melhor que a melhor solução encontrada em todos os jardins até o momento. Isto visa melhorar a velocidade de convergência. Com os dados feromônicos atualizados, é hora de executar um novo Jardim, dessa vez com as novas condiçoes probabilisticas. Esse processo se repete até que uma condiçao de parada seja atingida. No algoritmo proposto, a condiçao de parada ocorre quando 100 Jardins são calculados sem melhora da solução.

O método como um todo está exposto nos algoritmos a seguir.

```
1  Contagem = 0 ;
2  MelhorAtual.Custo ← ∞ ;
3  MelhorAtual.Ciclos ← ∅ ;
4  InicializaJardim;
5  enquanto Contagem < CondicaoParada faça
6  |  MelhorSolucao = NovoJardim(); Contagem ← Contagem + 1 ;
7  |  se MelhorSolucao.Custo < MelhorAtual.Custo então
8  |  |  MelhorAtual.Custo ← MelhorSolucao.Custo ;
9  |  |  MelhorAtual.Ciclos ← MelhorSolucao.Ciclos ;
10 |  |  Contagem = 0;
11 |  fim se
12 fim enqto
```

Algoritmo 4: Colônia de Formigas

O Algoritmo 4 representa o corpo do procedimento. Nas linhas 1 a 3, a variável Contagem é inicializada com 0 e a variável MelhorAtual, que guardará a melhor solução encontrada até o momento, é inicializada com um conjunto vazio e com o valor ∞. É feita a inicialização do Jardim na linha 4, com o uso do Algoritmo 5 a seguir. São gerados os Jardins (linha 6) e o procedimento pára quando *Contagem* atingir *CondicaoParada* (linha 5), ou seja, quando houver um certo número de Jardins sem melhora da solução. No corpo do procedimento, só interessa a melhor solução gerada no Jardim (*MelhorSolucao*), pois as outras soluções do jardim já serviram de base para a atualização dos feromônios ao final do procedimento *NovoJardim()*. A melhor solução corrente é comparada á melhor solução corrente (linha 7) e se foi gerada uma solução melhor que a corrente, a variável MelhorAtual recebe esta solução (linha 8-9).

```
1  Procedimento InicializaJardim ;
2  vCircuitos ← [Banco de dados de Circuitos → Ligações] ;
3  vLigacoes ← [Indexação Ligações → Circuitos de vCircuitos] ;
4  para cada C_i ∈ vCircuitos faça
5  |  vFeromonios[i] ← 1 ;
6  fim para cada
```

Algoritmo 5: Inicialização do Jardim

O Algoritmo 5 , na linha 2, lê o banco de circuitos de um arquivo, criando a relação Circuito → Ligações. Durante esta leitura, cria em memória (linha 3) a relação Ligações → Circuito. Embora o pseudo-código dê a entender que o preenchimento de vCircuitos e vLigacoes são procedimentos independentes, vLigacoes pode ser criado

durante a leitura do banco de dados com pequeno esforço computacional. Além disso, na linha 5, os feromônios de todos os circuitos são inicializados com o valor 1.

```
1  Função NovoJardim() ;
2  i ← 1;
3  repita
4  |  vFormigas[i] ← GeraFormiga() ;
5  |  i ← i + 1;
6  até i > NúmerodeFormigas;
7  media ← [Média dos custos das formigas em vFormigas] ;
8  para cada F_i ∈ vFormigas faça
9  |  para cada C_j ∈ F_i.Ciclos faça
10 |  |  vFeromonios[C_j] = vFeromonios[C_j]*media/F_i.custo ;
11 |  fim para cada
12 fim para cada
13 para cada C_i ∈ vCircuitos faça
14 |  vFeromonios[i] ← vFeromonios[i] * CoeficienteEvaporação ;
15 fim para cada
16 retorna Formiga de Melhor custo;
```

Algoritmo 6: Geração de um Jardim

O Algoritmo 6 gera cada um dos Jardins. São geradas *NúmerodeFormigas* formigas, através do procedimento GeraFormiga, na linha 4. Além disso, cada solução gerada por uma formiga é guardada no vetor vFormigas. O custo médio das soluções é calculado (linha 7) a partir desse vetor e os feromônios são atualizados na linha 10 conforme a fórmula proposta. No laço da linha 13, a evaporação é feita com o *CoeficienteEvaporação* sendo um número entre 0 e 1, onde uma evaporação de 15 % estaria representada com um coeficiente no valor de 0,85.

```
1  Função GeraFormiga() ;
2  F.custo ← 0 ;
3  F.ciclos ← ∅ ;
4  LigacoesNecessarias ← |vCircuitos| ;
5  para cada C_i ∈ vCircuitos faça
6  |  vCoeficienteUso[C_i] ← 0 ;
7  fim para cada
8  enquanto LigacoesNecessarias > 0 faça
9  |  L ←[Sorteio Aleatório de uma ligação dentre as não usadas] ;
10 |  C ←[Sorteio Probabilistico de um circuito dentre os não usados e que
   |  contém L] ;
11 |  F.ciclos ← F.ciclos ⋃ C;
12 |  para cada L_i ∈ C faça
13 |  |  se L_i não havia sido usada ainda então
14 |  |  |  LigacoesNecessarias ← LigacoesNecessarias - 1 ;
15 |  |  |  para cada K_i ∈ vLigacoes[L_i] faça
16 |  |  |  |  vCoeficienteUso[K_i] ← L_i.custo + vCoeficienteUso[K_i] ;
17 |  |  |  fim para cada
18 |  |  fim se
19 |  fim para cada
20 fim enqto
```

Algoritmo 7: Geração de uma Formiga

O Algoritmo 7 representa a geração de cada formiga ou solução. A solução começa com custo 0 (linha 2) e nenhum circuito (linha 3). Os valores heurísticos (*vCoeficienteUso*) são zerados para cada circuito na linha 6, representando que nenhum dos circuitos contém alguma ligação que já pertença a solução. O laço da linha 8 termina quanto todas as ligações fizerem parte da solução. Na linha 9, a formiga sorteia aleatoriamente uma ligação que ainda não conste na solução. Na linha seguinte, um circuito é escolhido por um sorteio com a probabilidade já exposta, sendo incluído na solução (linha 11). A atualização dos valores heurísticos é feita entre as linhas 12 e 19. Se uma nova ligação entrou na solução, o critério de parada está mais próximo de ser alcançado(linha 14) e todos os circuitos que contenham esta nova ligação devem ter seu coeficiente de uso atualizado (linha 16).

Capítulo 4

Resultados computacionais

Este capítulo dedica-se a avaliar os resultados computacionais da heurística proposta. São expostas as instâncias utilizadas, suas características e valores ótimos, quando possível. O propósito dos resultados expostos neste capítulo é fornecer o maior número possível de evidências sobre os pontos fortes e fracos da heurística proposta, assim como ajudar a prover valores para os coeficientes configuráveis do método, tais como evaporação, bonificação e quantidade de formigas por jardim.

4.1 Metodologia

A meta-heurística proposta foi desenvolvida em C++, com uso da biblioteca de uso comum STL, em busca de melhor desempenho. Todos os algoritmos de terceiros citados foram implementados, de forma que existisse a independência de módulos implementados externamente.
O ambiente de testes é um computador desktop, com processador Intel Core 2 Duo de 2.4 GHz, 3 Gb de memória RAM e sistema operacional Ubuntu 10.04 LTS Server Edition. Cada instância foi executada 20 vezes e os resultados aqui demonstrados representam os valores médios de custo, salvo palavra em contrário.

4.2 Instâncias

Serão usadas neste trabalho as instâncias utilizadas em [Corberán et al., 2007], onde algumas delas tem valores ótimos assinalados, permitindo uma análise sobre a qualidade da solução. Tais instâncias são fortemente conexas por construção. Foi feita a tentativa de obtenção do software XNÊS [Negreiros et al., 2009] para geração de novas instâncias

e comparação de resultados, mas o contato se mostrou infrutífero. Nesse caso, foi criado um pequeno programa para a geração de algumas instâncias com uma maior quantidade de vértices e ligações. Estes grafos aumentados foram gerados através da junção entre as instâncias de Corberán. Para gerar uma instância de 5000 nós, por exemplo, foram escolhidas duas instâncias já prontas, uma de 2000 e outra de 3000 nós. Para que este novo grafo se mantenha fortemente conexo, os dois grafos originais são unidos por uma aresta ou por dois arcos de sentido inverso, onde o nó de origem está em um grafo e o nó de destino estará no outro grafo. Os nós de origem e destino em cada grafo também são sorteados. Esta união é feita através de $\frac{|V|}{10}$ sorteios que escolhem entre uní-los com uma aresta ou com dois arcos de sentido inverso, gerando diversidade das novas ligações criadas.
A tabela 4.1 contém as características destas instâncias.

Tabela 4.1. Instâncias

Conjunto	# de Inst.	vértices	# de Arestas			# de Arcos			Gerador
			Médio	Mín.	Máx.	Médio	Mín.	Máx.	
MA05	12	500	615,1	351	1112	542,4	193	1082	Córberan
MB05	12	500	626,3	320	1131	583,8	202	1174	Córberan
MA10	12	1000	1235,3	693	2235	1083,9	340	2253	Córberan
MB10	12	1000	1260,1	636	2293	1182,1	416	2327	Córberan
MA15	12	1500	1851,7	1011	3423	1627,2	576	3329	Córberan
MB15	12	1500	1880,2	964	3540	1750,5	615	3475	Córberan
MA20	12	2000	2462,6	1355	4528	2182,8	733	4515	Córberan
MB20	12	2000	2499,8	1243	4552	2329,3	850	4583	Córberan
MA30	12	3000	3704,8	1999	6795	3254,3	1097	6603	Córberan
MB30	12	3000	3746,3	1992	6799	3384,5	1206	6742	Córberan
MJ50	4	5000	5534,7	3506	10455	5106,8	2034	8761	Próprio
MJ10K	4	10000	13001,6	8010	16652	9863,0	5364	14113	Próprio
MJ20K	4	20000	23131,5	15001	33236	17643,2	13992	25622	Próprio

Na primeira coluna, estão os nomes dados aos conjuntos. A Notação seguida foi a de [Corberán et al., 2007], com a adição das instâncias criadas para este trabalho, cujos conjuntos começam com MJ. Na segunda coluna, está o número de instâncias diferentes que compõe o conjunto e em seguida, o número de vértices. O número de arestas está representado pelos valores médios, mínimos e máximo dentro de cada conjunto de instâncias, assim como o número de arcos.

4.3 Convergência

O gráfico 4.1 demonstra uma execução da meta-heurística proposta, conforme os testes feitos. Todas as instâncias tiveram comportamento semelhante em relação á forma (logarítmica inversa) de convergência. No gráfico 4.1, o algoritmo é executado sobre a instância MA0532, parte do conjunto de instâncias MA05, com 15% de evaporação. Essa instância foi escolhida para a maioria dos testes por ser uma das menores, minimizando o tempo de execução e permitindo um maior número de testes. O eixo X é o tempo de execução do algoritmo, exibido como o número de Jardins de 100 formigas geradas. O número de formigas é um dos parâmetros discutidos neste capítulo. O eixo Y representa o custo da solução e estão duas séries plotadas: a série clara representa melhor custo encontrado naquele Jardim, enquanto a série escura representa o melhor resultado até aquele momento, para que possa ser analisada a latência das melhores soluções. No gráfico 4.2, uma demonstração da execução e do mecanismo de melhora descontínua da solução nos jardins de número 100 a 200, com esse intervalo sendo escolhido aleatoriamente.

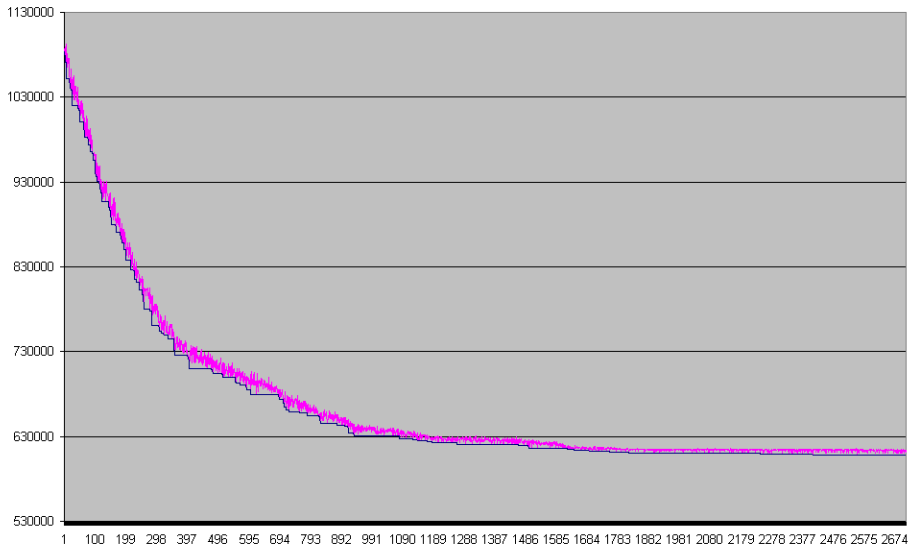

Figura 4.1. Execução na instância MA0532

O Gráfico 4.1 mostra a melhora da solução em função do número de jardins gerados. O custo inicial da solução é de cerca de 110000, sendo esta uma solução quase aleatória, pois os circuitos ainda não sofreram alterações em suas probabilidades. Em torno do 500^o jardim, a solução já tem custo 40% menor que o original e com a continuidade de execução do algoritmo, atinge 54% do custo original em torno do 1500^o jardim. Após essa fase, o gráfico demonstra que a solução ainda melhora, mas de

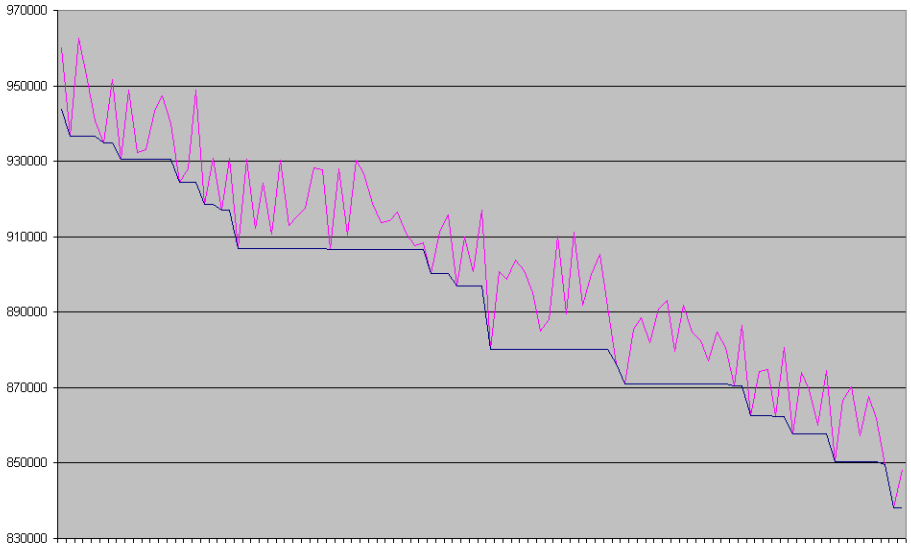

Figura 4.2. Detalhamento da execução nos jardins de # 100 a 200

forma mais lenta. O Gráfico 4.2 se propõe a demonstrar que cada Jardim pode gerar soluções melhores ou piores do que a solução corrente. Esse comportamento é esperado e mesmo as soluções piores do que a corrente contribuem para a solução, atualizando os feromônios dos circuitos envolvidos nesta.

4.4 Aproveitamento de formigas

Existe na literatura várias formas de aproveitamento das soluções geradas para atualização dos feromônios. Na grande maioria dos trabalhos, todas as soluções são aproveitadas na atualização dos feromônios,mas em alguns poucos, apenas uma parcela é utilizada. Se 20% das soluções forem aproveitadas, é usual nestes trabalhos selecionar as 10% melhores e as 10% piores soluções. Foi feito um experimento com as instâncias deste trabalho em busca desse aproveitamento.

Na Figura 4.3, está um gráfico da instância MA0532 em relação ao aproveitamento de formigas de um Jardim. A série de valores *Aprov. 10%*, no gráfico 4.3, significa que apenas as 5% melhores formigas e as as 5% piores formigas foram aproveitadas para a atualização dos feromônios. Os resultados para as outras instâncias se mostraram semelhantes, com a mais rápida convergência resultando do uso de 100% das formigas. Dessa forma, é feito o acúmulo e a retirada de feromônios dos circuitos na proporção da qualidade da solução. Além disso, existe um aspecto de esforço computacional a ser observado: é melhor que todo processamento de um determinado algoritmo seja aproveitado. Com o uso de todas as formigas, este aspecto é levado em consideração. Os valores expressos no gráfico 4.3 estão relativamente longe do custo ótimo devido ao fato de que a execução de cada teste foi restringida a 5 segundos, afetando a qualidade da solução final, mas permitindo um maior número de testes.

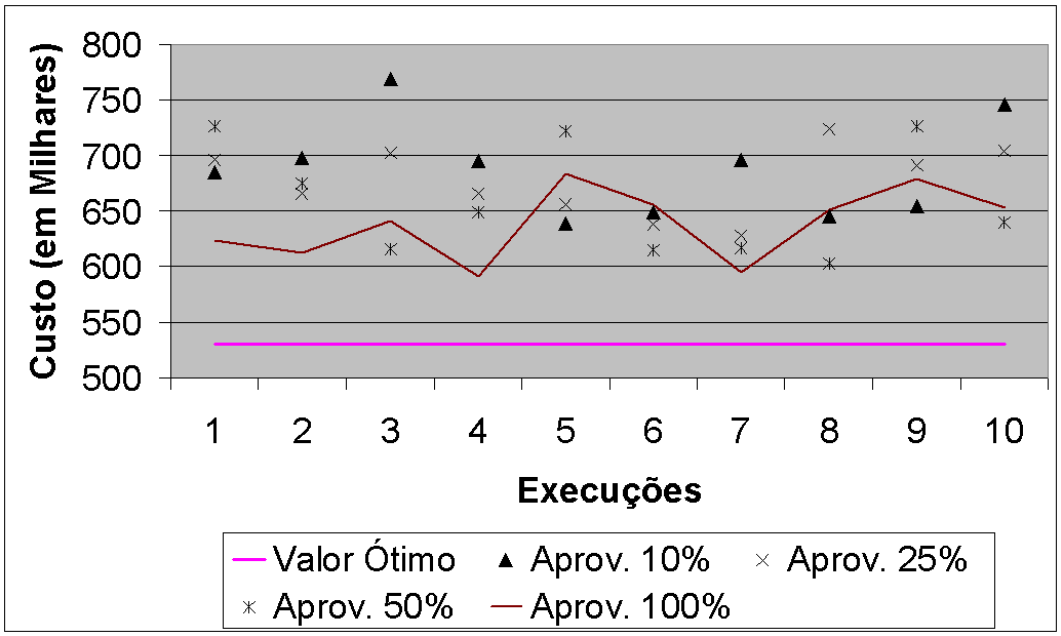

Figura 4.3. Aproveitamento do Jardim

4.5 Evaporação

A Figura 4.4 é um gráfico da instância MA0532 em relação ao coeficiente de evaporação em cada Jardim. A afirmação de semelhança entre instâncias citada na seção anterior se mantém. Foram utilizados 5 faixas de coeficientes (5% a 25%, variando 5 unidades percentuais). O coeficiente de evaporação 5% se mostrou melhor nos experimentos. Isto é facilmente observável no gráfico, onde a série 5% (representada pela linha, em contraste com os símbolos) passa quase sempre abaixo das outras séries, se aproximando mais do custo ótimo. Os valores expressos no gráfico 4.4 estão relativamente longe do custo ótimo devido ao fato de que a execução de cada teste foi restringida a 5 segundos, afetando a qualidade da solução final, mas permitindo um maior número de testes.

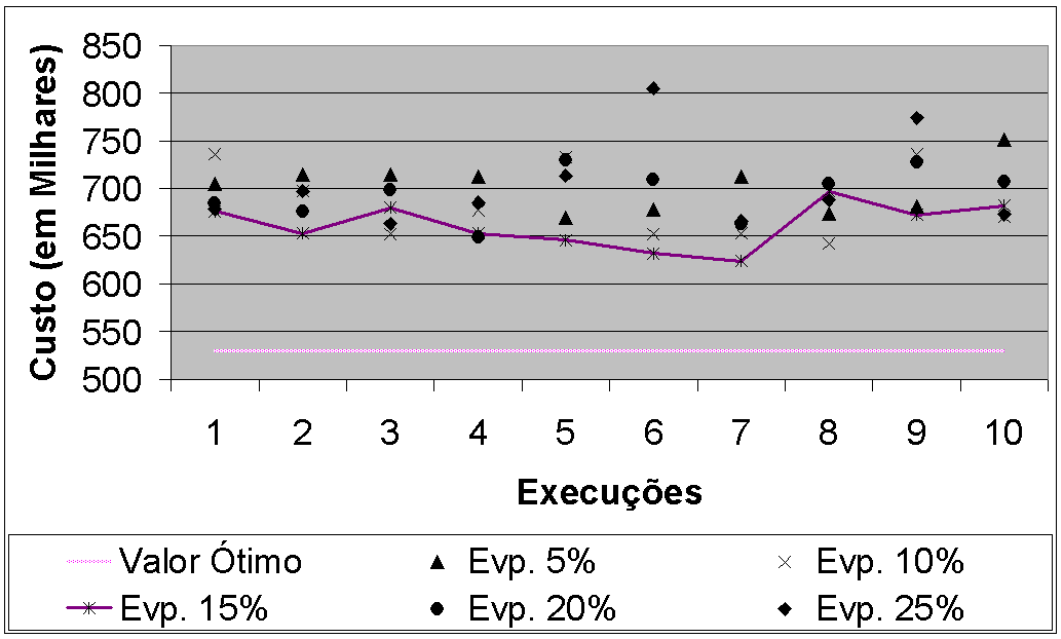

Figura 4.4. Evaporação

4.6 Geração de circuitos

Para a geração do banco de dados de circuitos, foi definida uma condição de parada simplista: Cada ligação (aresta ou arco) deve constar em pelo menos 10 circuitos diferentes. Esse número foi definido empiricamente, com base nos primeiros testes e na reincidência de ligação com baixa diversidade de circuitos. Essa condição garante diversidade na solução, impedindo que um mesmo circuito ou um pequeno conjunto de circuitos force a solução em uma direção única, por sua simples presença. A tabela 4.2 contém os resultados obtidos. A formação dos circuitos, conforme o capítulo 4,obedece a seguinte ordem:

- **Tamanho definido** : São gerados todos os circuitos de tamanhos 2 a 8.
- **Menor caminho**: Para as ligações que ainda não possuam circuitos e algumas escolhidas aleatoriamente, executamos o algoritmo.
- **Caminho aleatório**: São gerados para algumas ligação aleatórias e para as ligação que ainda não participam de 10 circuitos. Possui uma sub-condição de parada que é o limite de 60 segundos de execução.

Tabela 4.2. Geração de circuitos

Instâncias	Médias	
	Circuitos gerados	Tempo (s)
MA05	745,1	6,7
MA10	1330,3	12,4
MA15	1795,9	15,6
MA20	2888,8	27,7
MA30	3352,0	30,4
MJ50	6227,1	72,6
MJ20K	25755,4	230,4

Instâncias	Médias	
	Circuitos gerados	Tempo (s)
MB05	595,5	3,5
MB10	1369,7	7,5
MB15	1999,9	11,4
MB20	3233,1	20,7
MB30	3714,1	24,9
MJ10K	13012,6	128,9

4.7 Tempos de execução e qualidade da solução

Na tabela 4.3, estão os resultados por grupos de instâncias. Tal agrupamento foi feito em razão dos resultados em [Corberán et al., 2007] estarem expostos assim. A variação é a distância percentual entre a média das execuções dos algoritmos e o ótimo (onde exista) ou o *lower bound* da instância. Em [Corberán et al., 2007],os testes foram feitos em um processador AMD Athlon com Dual Core a 2.41GHz e 2GB de RAM. Como o conjunto de instâncias MJ foi criada apenas para este trabalho e a implementação do algoritmo de [Corberán et al., 2007] foge ao escopo deste trabalho, os valores de tempo de excução no algoritmo de Corberán foram calculados com base em uma interpolação, pelo método de Newton. Maiores detalhes deste método em [Campos, 2007].

A heurística exposta em [Yaoyuenyong & Charnsethikul, 2002], conhecida como SAPH (*Shortest Additional Path Heuristic*), foi implementada e executada sobre a primeira solução obtida para cada instância, haja visto que esta heurística necessita de uma solução inicial. Sua condição de parada é um limite de 1500 segundos(25 minutos) ou a execução de 200 iterações do algoritmo sem a inclusão ou exclusão de caminhos á solução.

Nos tempos de execução para a colônia de formigas estão inclusos os tempos de geração do banco de dados de circuitos e da execução da metaheurística. O critério de parada para a execução da metaheurística ocorre quando 200 jardins são calculados sem melhora da solução corrente. Para o cálculo da variação nas novas instâncias, foi utilizado como valor ótimo a soma dos ótimos dos dois grafos usados como base mais os valores das ligações inseridas para uní-los. Este não é um valor ótimo, mas é um lower bound.

Na primeira coluna, estão os nomes dos conjuntos testados. Nas duas colunas seguintes, estão a variação no custo encontrado pelo algoritmo de Branch & Cut de Corberán em relação ao custo ótimo e o tempo deste algoritmo. Estes valores estão em [Corberán et al., 2007]. As duas últimas colunas mostram esta mesma variação e tempo de execução para a metaheurística proposta.

Conforme a tabela 4.3 expressa, a metaheurística gera resultados aceitáveis apenas para instâncias grandes, onde o tempo de execução (em alguns maiores que uma hora e meia) se torna proibitivo. Entretanto, a heurística SAPH é pior em quase todos os casos. Em [Yaoyuenyong & Charnsethikul, 2002], a heurística é executada sobre uma

Tabela 4.3. Colônia de formigas - Resultados

Instâncias	SAPH		Branch & Cut		Colônia de Formigas	
	Variação	Tempo (s)	Variação	Tempo (s)	Variação	Tempo (s)
MA05	6,00%	16,5	0,00%	3,2	4,94%	31,8
MB05	4,69%	13,9	0,00%	7,5	3,58%	18,3
MA10	6,05%	56,8	0,00%	20,0	5,27%	63,7
MB10	3,24%	62,0	0,00%	78,6	2,93%	52,8
MA15	5,22%	146,9	0,00%	83,5	4,52%	72,6
MB15	4,57%	177,2	0,00%	139,9	3,60%	71,5
MA20	5,09%	322,3	0,00%	159,2	4,71%	140,8
MB20	6,78%	396,7	0,03%	119,6	3,42%	132,6
MA30	7,45%	963,1	0,03%	584,7	3,29%	164,6
MB30	6,03%	1020,9	0,24%	514	3,06%	173,9
MJ50	18,97%	1500,0	-	882,5	3,21%	278,3
MJ10K	27,03%	1500,0	-	1976,4	3,07%	446,9
MB20K	39,66 %	1500,0	-	5355,2	5,56%	800,3

solução inicial gerada pelo procedimento Misto ([Pearn & Chou, 1999]), mais próxima da solução ótima que a solução inicial proporcionada aqui.

Capítulo 5

Conclusão

Considerando os resultados demonstrados, o algoritmo resultados de boa aceptabilidade apenas nas instâncias maiores, onde o tempo de execução do algoritmo de Branh &Cut é algumas vezes, quase 7 vezes maior. A maior variação encontrada foi de 5,27 % do ótimo, bastante aceitável do ponto de vista prático. Do ponto de vista de tempo de execução, embora a média não tenha ultrapassado cerca de 7 minutos, algumas execuções de uma determinada instância maior chegaram a quase 20 minutos, enquanto outras execuções da mesma não ultrapassaram 2 minutos. Isso se deve ao caráter de sorteio de circuitos do algoritmo e, em teoria, á qualidade das escolhas iniciais feitas para ácumulo ou retirada de feromônios.

Além disso, o algoritmo proposto é adequado ás aplicações (coleta de lixo, inspeção de fios, limpeza de neve, ...) de forma bastante interessante. Se o algoritmo for aplicado ao problema de coleta de lixo urbano, por exemplo, e a autoridade de trânsito modifica o sentido em que determinada rua é trafegada, basta verificar se nos circuitos da solução que contém aquela rua, esta é trafegada em sentido proibido. Se não houverem outros circuitos na base de dados com a rua no novo sentido, utiliza-se os métodos de geração apenas para esta rua e o algoritmo de colônia de formigas começa da solução original apenas sem o circuito inválido, poupando tempo.

5.1 Trabalhos Futuros

O algoritmo possui certos aspectos a serem trabalhados. A qualidade e a velocidade de convergência das soluções acaba sendo um pouco dependente das escolhas feitas nos primeiros jardins. Uma futura melhoria deveria levar em conta esse aspecto, reaproveitando jardins de uma forma mais eficiente.

Além disso, existem outros modelos de distribuição probabilística que poderiam ser adaptados ao problema, principalmente em relação aos dados heurísticos.
Um aspecto que pode ser abordado é a forma de sorteio das ligações ainda não constantes da solução em cada formiga. O sorteio neste trabalho é aleatório, mas um parâmetro heurístico de escolha (o número de ligações adjacentes ainda não usadas, por exemplo) pode resultar em soluções melhores e de convergência mais rápida.
As estruturas de dados deste trabalho foram pensadas tendo em vista uma paralelização do problema, porém esta técnica não foi utilizada aqui e poderá render soluções rápidas e de qualidade para o problema. Muito embora pareça pequeno, merece um estudo o tamanho do impacto de mudanças pontuais de orientação em certas ligações, medindo o trabalho computacional para o recálculo da solução.
É necessária também a comparação do método com outras heurísticas da literatura para o problema.

Referências Bibliográficas

Aminu, U. & Eglese, R. (2006). A constraint programming approach to the chinese postman problem with time windows. *Computers & Operations Research*, 33(12):3423 – 3431. Part Special Issue: Recent Algorithmic Advances for Arc Routing Problems.

Bautista, J.; Fernández, E. & Pereira, J. (2008). Solving an urban waste collection problem using ants heuristics. *Computers & Operations Research*, 35(9):3020--3033.

Benavent, E.; Campos, V.; Corberán, A. & Mota, E. (1987). The capacitated chinese postman problem part i: An heuristic algorithm. pp. 16–22.

Benavent, E.; Campos, V.; Corberán, A. & Mota, E. (1992). The capacitated arc routing problem: Lower bounds. *Networks*, (22):669--690.

Bodin, L.; Golden, B.; Assad, A. & Ball, M. (1983). Routing and scheduling of vehicles and crews. *Computers & Operations Research*, (10):63–211.

Campos, F. F. (2007). *Algoritmos Numéricos. 2. ed.* LTC Editora.

Christofides, N.; Benavent, B.; Campos, V.; Corberan, A. & Mota, E. (1983). An optimal method for the mixed postman problem. *System modeling and optimization. Lecture Notes in Control and Information Sciences*, (59):641–649.

Corberán, A.; Martín, R.; Martínez, E. & Soler, D. (2002a). The rural postman problem on mixed graphs with turn penalties. *Computers & Operations Research*, (29):887–903.

Corberán, A.; Martín, R. & Sanchis, J. M. (2002b). A grasp heuristic for the mixed chinese postman problem. *European Journal of Operational Research*, (142):70–80.

Corberán, A.; Mota, E. & Sanchis, J. M. (2006). A comparison of two different formulations for arc routing problems on mixed graphs. *Computers & Operations Research*, (33):3384–3402.

Corberán, A.; Plana, I.; Reinelt, G.; Sanchis, J. & Theis, D. (2007). New results on the windy postman problem. *Technical Report*.

Desrochers, M.; Dror, M. & Galinas, G. (1991). The chinese postman problem with general precedence relations. *Technical Report*.

Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms (em Italiano)*. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milão, Itália.

Dror, M. (2000). *Arc routing: Theory, solutions and applications*. Boston: Kluwer Academic Publishers.

Edmonds, J. (1965). The chinese postman problem. *Operations Research*, (13, I):373–388.

Edmonds, J. & Jonhson, E. L. (1973). Matching, euler tours and the chinese postman. *Mathematical programming*, (5):88–124.

Ford, L. R. & Fulkerson, D. R. (1962). *Flows in Networks*. Princeton University Press.

Frederickson, G. N.; Hecht, M. S. & Kim, C. E. (1976). Approximation algorithms for some routing problems. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:216–227.

Goss, S.; Aron, S.; Deneubourg, J. & Pasteels, J. (1989). Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76(12):579--581.

Gross, J. & Yellen, J. (2003). *Handbook of Graph Theory (Discrete Mathematics and Its Applications)*. CRC Press, 1 edição.

Laporte, G. (1997). Modeling and solving several classes of arc routing problems as traveling salesman problems. *Computers & Operations Research*, 24(1):1057–1061.

Lee, O. & Wakabayashi, Y. (2002). On the circuit cover problem for mixed graphs. *Combinatorics, Probability & Computing*, 11(1):43--59.

Malandraki, C. & Daskin, M. S. (1993). The maximum benefit chinese postman problem and the maximum benefit traveling salesman problem. *European Journal of Operational Research*, (65):218–234.

Meigu, G. (1962). Graphic programming using odd and even points. *Chinese Mathematics*, (1):273–277.

Minieka, E. (1979). The chinese postman problem for mixed networks. *Management Science*, (25):643–648.

Negreiros, M.; Rodrigues, W.; Casto, A.; Coutinho, E. & Castro, G. (2009). O problema do carteiro chinês, algoritmos exatos e um ambiente mvi para análise de suas instâncias: Sistema xnÊs. *Pesquisa Operacional*, (29):323–363.

Neumann, F. (2008). Expected runtimes of evolutionary algorithms for the eulerian cycle problem. *Computers & Operations Research*, (35):2750–2759.

Nobert, Y. & Picard, J. C. (1996). An optimal algorithm for the mixed chinese postman problem. *Networks*, (27):95–108.

Papadimitriou, C. H. (1976). On the complexity of edge traversing. *J. of the Assoc. of Comptng. Machinery*, (23):544.

Pearn, W.-L.; Assad, A. & Golden, B. L. (1987). Transforming arc routing into node routing problems. *Computers & Operations Research*, 14(4):285 – 288.

Pearn, W. L. & Chou, J. (1999). Improved solutions for the chinese postman problem on mixed networks. *Computers & Operations Research*, (26):819–827.

Pearn, W. L. & Liu, C. M. (1995). Algorithms for the chinese postman problem on mixed networks. *Computers & Operations Research*, 22(5):479–489.

Raghavachari, B. & Veerasamy, J. (1999). A 3/2- approximation algorithm for the mixed postman problem. *SIAM Journal on Discrete Mathematics*, (12).

Sbihi, A. & Eglese, R. W. (2007). Combinatorial optimization and green logistics. p. Invited Survey.

Sherafat, H. (1988). Uma solução para o problema do carteiro chinês misto. In *Anais do IV CLAIO XXI SBPO*, pp. 157–170.

Wang, H.-F. & Wen, Y.-P. (2002). Time-constrained chinese postman problems. *Computers & Mathematics with Applications*, 44:375 – 387.

Win, Z. (1987). *Contributions to routing problems*. PhD thesis, University of Augsburg, Augsburg, Alemanha.

Yan, H. & Thompson, G. (1998). Finding postal carrier walk paths in mixed graphs. *Comput Optim Appl*, (9):229–247.

Yaoyuenyong, K. & Charnsethikul, P. (2002). A heuristic algorithm for the mixed chinese postman problem. *Optimization and Engineering*, (3):157–187.