UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MAURO HENRIQUE MULATI

Investigação da meta-heurística de otimização por colônia de formigas artificiais aplicada ao problema de cobertura de conjunto

Maringá
2009

Mauro Henrique Mulati

Investigação da meta-heurística de otimização por colônia de formigas artificiais aplicada ao problema de cobertura de conjunto

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual de Maringá, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.
Área de concentração: Ciência da Computação

Orientador: Prof. Dr. Ademir Aparecido Constantino

Maringá
2009

"Dados Internacionais de Catalogação-na-Publicação (CIP)"
(Biblioteca Setorial - UEM. Nupélia, Maringá, PR, Brasil)

M954i

Mulati, Mauro Henrique, 1982-
Investigação da meta-heurística de otimização por colônia de formigas artificiais aplicada ao problema de cobertura de conjunto / Mauro Henrique Mulati. -- Maringá, 2009.
140 f. : il.

Dissertação (mestrado em Ciência da Computação)--Universidade Estadual de Maringá, Dep. de Informática, 2009.
Orientador: Prof. Dr. Ademir Aparecido Constantino.

1. Meta-heurística - Aplicação. 2. Otimização por colônia de formigas artificiais. 3. Otimização combinatória. 4. Problema de cobertura de conjunto. I. Universidade Estadual de Maringá. Departamento de Informática. Programa de Pós-Graduação em Ciência da Computação.

CDD 22. ed. –003.3
NBR/CIP - 12899 AACR/2

Maria Salete Ribelatto Arita CRB 9/858
João Fábio Hildebrandt CRB 9/1140

FOLHA DE APROVAÇÃO

Mauro Henrique Mulati

Investigação da meta-heurística de otimização por colônia de formigas artificiais aplicada ao problema de cobertura de conjunto

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual de Maringá, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Aprovado em 26/02/2009.

BANCA EXAMINADORA

Prof. Dr. Ademir Aparecido Constantino
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Wesley Romão
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Robinson Samuel Vieira Hoto
Universidade Estadual de Londrina – DM/UEL

Local de defesa: Auditório da Incubadora Tecnológica de Maringá, bloco 14, *campus* da Universidade Estadual de Maringá.

“A grandeza é um caminho,
não um lugar.”
Adaptado de Donnie Berkholz

Dedico este trabalho
à minha família, à minha
querida namorada Daniele e
aos meus amigos.

Agradecimentos

Agradeço à Deus pelo dom da vida e pelas oportunidades que nela coloca, dentre elas aquela de realizar o presente trabalho. Agradeço à meus pais Mauro e Marlene, ao meu irmão Fábio, à minha irmã Thatiane e à minha namorada Daniele pela força e pelo apoio incondicional em todas os momentos. Ao caro professor e orientador Ademir meu muito obrigado pelas idéias, apoio e suporte durante esta jornada. Agradeço ao professor Ronaldo pelo apoio e discussões produtivas. Aos meus amigos Rogério e Maurílio, aos amigos de Nova Esperança e à todas as pessoas que de uma forma ou de outra contribuíram envio saudações e agradecimentos.

Investigação da meta-heurística de otimização por colônia de formigas artificiais aplicada ao problema de cobertura de conjunto

Resumo

O presente trabalho utiliza-se do problema de otimização combinatória denominado problema de cobertura de conjunto (PCC), classificado como NP-difícil. À tal problema são aplicados os algoritmos heurísticos baseados em *Ant Colony Optimization* (ACO) *Max-Min Ant System* (MMAS_L) e *Ant System RC_2* (AS_RC_2), além de se propor o *Adaptive Ant System* (AAS_MC), ressaltando que as colunas do PCC são referidas como componentes no contexto de algoritmos ACO. Objetiva-se calibrar os parâmetros de tais algoritmos de forma a conseguir soluções de qualidade em tempos computacionais aceitáveis, bem como fornecer algoritmos que contribuam com inovações. Dessa forma, os aspectos mais inovadores são dados pelo estudo do AS_RC_2 e pela proposta do AAS_MC. O AS_RC_2 possui um mecanismo de formação de conjunto de componentes candidatos para o passo de construção da formiga baseado em linhas da instância do PCC, que faz com que tal conjunto seja menor que as comumente utilizadas, enquanto que o AAS_MC apresenta mecanismo para evitar estagnação por meio da adaptação das importâncias do feromônio e da informação heurística em sua regra de decisão. Considerando que a ordem dos componentes de uma solução não é importante, o presente trabalho propõe diferentes maneiras de manipular o feromônio, com a representação, a consulta e a atualização de feromônio podendo ser feitos por componentes, seqüência de pares de componentes ou todos os pares de componentes. Assim, por componentes indica a maneira convencional de aplicação, por seqüência de pares faz com que se use as conexões entre os componentes e todos os pares indica o uso da conexão de todos com todos os componentes de uma solução. Por fim, são reportados e analisados resultados de experimentos realizados, destacando-se três modalidades de busca local: NBL que identifica nenhuma busca local em uso; JB2, que é basicamente uma perturbação da solução direcionada a colunas de um bom custo-benefício; como também usa-se o RFL, que faz uma busca na vizinhança da solução atual de modo a verificar todas as soluções com distância Hamming de até 3 em relação a esta.

Palavras-chave: Otimização por colônia de formigas artificiais. *Ant Colony Optimization.* ACO. Problema de cobertura de conjunto. PCC. Meta-heurística. Otimização combinatória.

Investigation of the ant colony optimization meta-heuristic applied to the set covering problem

Abstract

This thesis use the combinatorial optimization problem called set covering problem (SCP), which is classified as NP-hard. To that problem are applied the heuristic algorithms based on Ant Colony Optimization (ACO) Max-Min Ant System (MMAS_L) and Ant System RC_2 (AS_RC_2), besides the proposing of the Adaptive Ant System (AAS_MC), emphasizing that columns of the SCP are referred as components in the context of ACO algorithms. These investigations aim to calibrate the parameters of the algorithms in such a way to obtain solutions with quality in acceptable computational times, besides providing innovative algorithms. So, the more innovative aspects are the study of the AS_RC_2 and the proposition of the AAS_MC. The AS_RC_2 has a mechanism that make the set of candidate components for the construction step of the ant based on lines of the SCP instance, that mechanism makes such set smaller than the ones that are usually used, while the AAS_MC presents a way to avoid stagnation by the adaptation of the importances of pheromone and heuristic information in its decision rule. Considering that the order of the components of a solution is not important, this study proposes different manners to handle the pheromone, with the representation, the consulting and the updating of the pheromone being done by components, pair sequence of components or all pairs of components. Thus, by components we indicate the conventional way of utilization, by sequence of pairs we mean that is used the connections between the components and all pairs indicates the use of the connections from all to all components of a solution. Finally, results of experiments are reported and analyzed, emphasizing three ways of local search: NBL, which means that there is not a local search in use; JB2, that is basically a perturbation of the solution directed to columns with a good cost-benefit and there is also the RFL, which searches the neighborhood of the solution in such a way that all solutions with Hamming distance with up to 3 from the referred solution.

***Keywords*:** Ant Colony Optimization. ACO. Set covering problem. SCP. Meta-heuristic. Combinatorial optimization.

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

AAS_MC	*Adaptive Ant System* de Mulati e Constantino
ACO	*Ant Colony Optimization*
AS	*Ant System*
AS_HRTB	*Ant System* de Hadji *et al.* (2000)
AS_LM	*Ant System* de Leguizamon e Michalewicz (2000)
AS_RC_2	*Ant System* baseado em Reis e Constantino (2003)
CCB	Custo de cobertura
CCL	Custo de coluna
CMP	Componentes
GRASP	*Greedy randomized adaptive search procedure*
JB2	Algoritmo heurístico de busca local baseado em Jacobs e Brusco (1995)
MMAS	*Max-Min Ant System*
MMAS_L	*Max-Min Ant System* baseado em Lessing *et al.* (2004)
MMAS_LDS	*Max-Min Ant System* de Lessing *et al.* (2004)
NBL	Nenhuma busca local
NP	Não-determinístico polinomial
PCC	Problema de cobertura de conjunto
PCV	Problema do caixeiro viajante
PMC	Problema de encontrar o menor caminho em grafo
RFL	Algoritmo heurístico de busca local baseado em Yagiura *et al.* (2006)
RNA	Redes Neurais Artificiais
SA	*Simulated Annealing*
SEQ	Sequência de pares
TDN	Todos os pares de maneira normal
TOD	Todos os pares ou todos os pares de maneira híbrida

Sumário

1 Introdução

Problemas que se enquadram na classificação NP-difícil, como muitos problemas de otimização combinatória, demandam um poder computacional muitas vezes inviável para serem resolvidos de forma exata procurando uma solução ótima. Algoritmo exato para tal resolução normalmente tem complexidade computacional de tempo exponencial (Ramalhinho-Lourenço e Serra, 1998).

Assim, uma alternativa para se resolver tais problemas em tempo polinomial é a utilização de meta-heurísticas e algoritmos heurísticos. Meta-heurística é definida como uma estrutura genérica para desenvolvimento e aplicação de algoritmos heurísticos para resolução aproximada de problemas classificados como NP-difíceis ou NP-completos (Garey e Johnson, 1979). Normalmente, algoritmos heurísticos não garantem a solução ótima, porém, avaliações empíricas têm mostrado que tal classe de algoritmos têm cumprido com satisfação a missão de retornar uma solução boa o suficiente, ou até mesmo ótima, para determinados problemas em um tempo de computação aceitável. Além de algoritmos heurísticos, cada técnica normalmente define estruturas de dados próprias para resolução dos problemas. Meta-heurísticas – e seus algoritmos heurísticos derivados – podem se originar de: inspiração natural, como no caso de *Simulated Annealing* (SA) ou têmpera simulada, Algoritmos Genéticos (AG), Redes Neurais Artificiais (RNA), *Ant System* (AS) ou sistema de formigas, dentre outros; ou ainda, podem se originar a partir de inspiração na observação dos resultados, como Busca Tabu, GRASP dentre outros.

O presente trabalho se propõe a investigar a meta-heurística de Otimização por Colônia de Formigas Artificias, do inglês *Ant Colony Optmization* (ACO) (Dorigo e Stützle, 2004), com o objetivo de se obter algoritmos heurísticos mais rápidos e com melhores resultados. Tal investigação desenvolve-se de modo a expor várias ramificações da meta-heurística ACO, bem como casos de aplicação.

ACO pode ser aplicado tanto a problemas estáticos quanto dinâmicos. Problemas estáticos são aqueles que não se modificam enquanto estão sendo resolvidos, por outro lado, problemas dinâmicos se modificam enquanto estão sendo resolvidos (Dorigo e Stützle, 2004).

Nesse aspecto, o objetivo é contribuir na implementação e avaliação de resultados sobre o problema estático denominado problema de cobertura de conjunto (PCC).

Para fundamentar o presente trabalho, é apresentada uma revisão bibliográfica sobre o PCC e seus aspectos relacionados, sobre a meta heurística ACO (Dorigo e Stützle, 2004) de modo a apresentar várias de suas ramificações e aplicações, bem como é realizada revisão dos algoritmos heurísticos baseados em ACO para PCC.

Assim, os principais objetivos da presente dissertação são: a elaboração, extensão, implementação e análise de variações da meta-heurística ACO bem como modificações na estrutura de feromônio e na manipulação do mesmo, com as idéias sobre feromônio inspiradas naquelas utilizadas em Alaya *et al.* (2004) e em Solnon e Bridge (2006). As estruturas de depósito de feromônio contemplam as possibilidades de se depositar o feromônio em cada componente (coluna da matriz de restrição do PCC), cada par de componentes ou em todos os pares de componentes em uma solução.

Além disso, é apresentado o estudo e a extensão do AS_RC_2 proveniente do trabalho de Reis e Constantino (2003), bem como é proposto um novo esquema para ACO denominado ***Ant System* Adaptativo,** do inglês têm-se *Adaptive Ant System* (AAS), sendo que os parâmetros que definem os pesos dos fatores que compõem a informação de escolha são ajustados de acordo com a qualidade da solução em dada iteração. Esta proposta visa a eliminação do típico problema da estagnação normalmente encontrada nos algoritmos baseados em ACO e a diminuição dos parâmetros a serem definidos pelo usuário.

Também é parte do presente estudo utilizar e avaliar três modalidade de busca local, como segue: NBL que identifica nenhuma busca local em uso; JB2, que é basicamente uma perturbação da solução direcionada a colunas de uma boa relação custo-benefício, busca local esta que é baseada em Jacobs e Brusco (1995); como também usa-se o RFL, que faz uma busca na vizinhança da solução atual de modo a verificar todas as soluções com distância Hamming de até 3 em relação a esta, conforme idéias base expostas em Yagiura *et al.* (2006).

Para avaliar e validar, os algoritmos de interesse serão executados para resolver instâncias do PCC provenientes de uma base de dados utilizada em inúmeros trabalhos científicos, a OR-Library (Beasley, 1990), a fim de poder-se comparar os resultados de qualidade da solução e tempo de execução com outras implementações de ACO e ainda ser possível a comparação com outras técnicas existentes.

É importante notar que o AS – e ACO em geral – diz respeito à algoritmo heurístico construtivo, pois constrói soluções a partir dos componentes fornecidos, ao passo que possui também características de algoritmos heurísticos melhorativos, pois realiza a construção de tais soluções com base no feromônio que foi depositado por formigas em construções anteriores.

O texto está estruturado de forma que o Capítulo 2 expõe o PCC e seus aspectos relacionados de forma intrínseca como informação heurística, algoritmo *greedy*, buscas locais e questões de eliminação de colunas redundantes e redução do problema. O Capítulo 3 objetiva expor a meta-heurística ACO de forma geral, sendo no Capítulo 4 onde é feita a abordagem de algoritmos heurísticos baseados em ACO aplicados ao PCC. O Capítulo 5 apresenta as propostas do presente trabalho. O Capítulo 6 prima por expor os experimentos, seus resultados e respectivas análises sendo o trabalho concluído no Capítulo 7, este que também contém os trabalhos futuros. Após o Capítulo 7 tem-se as referências bibliográficas.

2 Problema de Cobertura de Conjunto e Aspectos Intrínsecos

Apresenta-se no presente Capítulo o Problema de Cobertura de Conjunto (PCC), sendo este um problema de encontrar um subconjunto de componentes dentre aqueles definidos, encaixando-se assim na classe de problemas de subconjunto. Classe a qual uma solução é composta por uma seleção de elementos de um conjunto dado respeitando as restrições do problema – constituindo assim uma solução factível – e que a ordem da disposição dos elementos na solução não é relevante. É possível que a instância do PCC passe por um processo de redução, que dependendo da instância em questão pode resultar em uma diminuição no tamanho desta, em tese tornando o processo de se encontrar uma solução para tal instância menos complexo no quesito tempo de processamento.

A informação heurística provém informações específicas a respeito da instância do PCC em questão. A partir da informação heurística pode-se definir um algoritmo heurístico construtivo guloso (*greedy*), que parte de um ponto inicial aleatório qualquer do problema e adiciona novos componentes que maximizem a informação heurística, sendo tal algoritmo o mais simples de ser criado dentre os construtivos.

A busca local também é inerente ao problema, consistindo esta de um algoritmo heurístico melhorativo que realiza buscas a partir de uma solução fornecida, com o objetivo de encontrar uma solução melhor variando partes desta. Como aspecto melhorativo há ainda o processo de eliminar os componentes redundantes que possam existir em uma determinada solução.

2.1 O Problema

O PCC pertence à categoria de problemas de subconjunto. Em tal categoria de problema, a solução é dada por um subconjunto de itens disponíveis, sujeita às restrições específicas do problema.

O PCC é definido como uma matriz $m \times n$ definida por $A=[a_{ij}]$, a qual cada um dos elementos podem ser 0 ou 1. Cada coluna da matriz A possui associado um custo não-

negativo c_j. Considera-se também que determinada coluna j cobre uma linha i se $a_{ij}=1$. Tem-se que o objetivo a ser alcançado no PCC é escolher um subconjunto $s \subseteq \{1,\ldots,n\}$ de colunas com custo mínimo, de modo que todas as linhas sejam cobertas. Introduzindo a variável binária x_j, que tem o valor 1 se $j \in s$ e tem valor 0 caso contrário, o PCC pode ser definido como:

$$min\ f(x)=\sum_{j=1}^{n} c_j \cdot x_j \tag{1}$$

sujeito a

$$\sum_{j=1}^{n} a_{ij} \cdot x_j \geqslant 1,\ \ i=1,\ \ldots,\ m, \tag{2}$$

$$x_j \in \{0,1\},\ \ j=1,\ \ldots,\ n, \tag{3}$$

onde a restrição definida pela Equação 2 mostra que cada linha deve ser coberta por pelo menos uma coluna. Pela Equação 3 tem-se que x_j assume o valor 0 ou 1, *i.e.* é uma variável de decisão. E ainda, tem-se que se x_j possui o valor 1, a coluna j se encontra no conjunto solução s e caso x_j tenha o valor 0, a coluna j não faz parte do conjunto solução s.

Define-se ainda em relação à instância do PCC:

- $M=\{1,\ldots,m\}$: conjunto de todas as linhas;
- $N=\{1,\ldots,n\}$: conjunto de todas as colunas;
- $M_j=\{i \mid i \in M \wedge a_{ij}=1\}$: conjunto de linhas que são cobertas pela coluna j; e
- $N_i=\{j \mid j \in N \wedge a_{ij}=1\}$: conjunto de colunas que cobrem a linha i.

Uma importante utilização do PCC é a sua aplicação ao escalonamento de tripulação em empresa de transporte (Desrochers e Soumis, 1989; Reis e Constantino, 2003). Como exemplo de PCC, tem-se a matriz binária a seguir:

$$A_{ij}=\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Considerando também o vetor de custo:

$$Custo=\begin{bmatrix} 23 & 12 & 30 & 51 & 5 \end{bmatrix}$$

Assim, com c_j denotando o custo da coluna j, é necessário minimizar a soma de $c_1+c_2+c_3+c_4+c_5$ de modo que todas as linhas sejam cobertas.

Supondo uma solução s_1 dada pelo vetor x que contenha as colunas 1, 3 e 4, tem-se a Figura 2.1.

A_{ij}	***1***	***2***	***3***	***4***	***5***	$a_{ij} \cdot x_j$
1	0	1	1	0	0	1
2	1	0	1	0	0	2
3	1	0	0	1	0	2
4	0	1	1	1	1	2
x	**1**	**0**	**1**	**1**	**0**	

Figura 2.1. Exemplo de instância do PCC e de uma solução

Desse modo, a restrição da Equação 2 é satisfeita. Este aspecto garante que todas as linhas serão cobertas pela solução, em outras palavras, a solução é dita factível. O custo da solução s_1 é 104. Por fim, resta analisar o valor dos custos, para verificar se esta é uma solução caraterizada como sendo mínima possível, tendo que para isso, é necessário verificar todas as possibilidades de solução para saber se a solução ótima foi encontrada.

2.2 Pré-Processamento para Redução do Problema

A redução do problema original em um problema equivalente menor é feita em alguns casos em uma etapa de pré-processamento, no intuito de facilitar sua resolução ou melhorar o desempenho do algoritmo de processamento. As regras para a redução empregadas foram: verificação de coluna nula, inclusão de coluna, domínio de linha e domínio de coluna.

A verificação de *coluna nula* elimina as colunas que não cobrem nenhuma das linhas consideradas para o problema, formalmente tem-se que se $\exists j \in N$ tal que $M_j = \emptyset$, então a coluna j deve ser eliminada do problema, pois como mencionado não cobre nenhuma linha.

Quando detectado que uma determinada linha é coberta apenas por uma das colunas, *a coluna pode ser incluída* de forma fixa como pertencente à solução do problema e a linha pode ser retirada, pois não será preciso trabalhar com ela novamente, já que a única coluna que a cobre já pertence à solução. Neste caso houve uma redução na dimensão linha para o problema considerado, formalmente tem-se que, se $\exists i \in M$ tal que $i \in M_j$ e $i \notin M_k \ \ \forall k \neq j$, então j deve estar em todas as soluções, e as linhas cobertas por M_j podem ser eliminadas, e o problema reduz-se a $M = M - M_j$ e $N = N - \{j\}$ (de Oliveira, 1999).

A *dominância entre as linhas* é checada verificando-se as linhas que são cobertas pelas mesmas colunas, a intersecção entre os pontos de cobertura permite que linhas sejam eliminadas, formalmente para o domínio de linha tem-se que se $\exists i, h \in M$ tal que $N_i \subseteq N_h$, então elimina-se a linha h. Diz-se, neste caso, que a linha h é dominada pela linha i, pois toda coluna que cobre a linha i também cobrirá a linha h (de Oliveira, 1999).

O *domínio de coluna* ocorre quando linhas cobertas por uma determinada coluna também são cobertas por uma outra coluna, porém com um custo menor ou igual.

Formalmente o domínio de coluna ocorre se $\exists j,k \in N$, com $M_k \subseteq M_j$ e $c_j \leqslant c_k$, então a coluna k deve ser eliminada, uma vez que a coluna j cobrirá todas as linhas de M_k a um custo não superior a c_k. Diz-se neste caso, que a coluna k é dominada pela coluna j.

A redução permite que instâncias de problemas sejam transformados em uma versão menor, podendo proporcionar um ganho de complexidade de espaço e de execução, desde que o custo de se aplicar os algoritmos de redução mais a aplicação de algoritmo heurístico não seja superior ao custo de processamento do algoritmo heurístico para o problema em sua forma original.

```
 InstanciaPcc::reduzir(){
1   houveColunaNula      =false;
2   houveInclusaoDeColuna=false;
3   houveLinhaDominada   =false;
4   houveColunaDominada  =false;
5   do{
6      houveColunaNula      =checarColunaNulaEEfetivar();
7      houveInclusaoDeColuna=checarFixacao__InclusaoDeColunaRemocaoDeLinhasEEft();
8      houveLinhaDominada   =checarDominioDeLinhaEEfetivar();
9      houveColunaDominada  =checarDominioDeColunaEEfetivar();
10  } while( houveColunaNula       or
             houveInclusaoDeColuna or
             houveLinhaDominada    or
             houveColunaDominada );
 }
```

Figura 2.2: Algoritmo geral de redução de instância do PCC

A implementação desenvolvida para utilizar redução de problema é apresentada na Figura 2.2, podendo-se notar que no pseudo-código principal é feita uma composição na utilização das verificações mencionadas. A checagem de existência de colunas nulas, a verificação de colunas que devem ser fixadas e possíveis inclusões de colunas, a checagem de domínio de linha e checagem de domínio de coluna.

2.3 Informação Heurística

A informação heurística desempenha o papel de conhecimento específico do problema, e está relacionada com o algoritmo heurístico utilizado em algoritmos construtivos, como por exemplo, no algoritmo guloso (*greedy* ou míope).

Em algoritmos heurísticos ACO a informação heurística desempenha também o papel de conhecimento específico do problema. No contexto da meta-heurística ACO, tal informação heurística é também denominada visibilidade, em alusão a capacidade de visão das formigas artificiais em relação aos dados do problema. A informação heurística pode ser do tipo estático ou dinâmico.

No tipo estático, o cálculo da informação heurística depende apenas da instância do problema em questão, sendo assim o calculo é efetuado apenas uma vez na inicialização do algoritmo mantendo seu valor até o fim da execução do mesmo. No tipo dinâmico, o cálculo é

dependente da solução parcial construída por cada formiga, sendo assim seu cálculo é computado a cada passo de construção de cada uma das formigas (Lessing *et al.*, 2004). Experimentos a respeito podem ser verificados no Capítulo 6.

As informações heurísticas utilizadas no presente trabalho são a informação heurística estática custo de coluna e a informação heurística dinâmica custo de cobertura, conforme segue descrição nas próximas Subseções.

2.3.1 Informação Heurística Estática Baseada no Custo de Coluna

A informação heurística baseada no custo de coluna é referenciada no presente trabalho por CCL e é calculada a partir do custo de cada coluna da forma mostrada (Lessing *et al.*, 2004) na Equação 4:

$$n_j = \frac{1}{c_j}, \tag{4}$$

de maneira que c_j é o custo da coluna j. Assim, quanto menor for o custo da coluna j, maior será o valor de sua informação heurística.

Pode-se facilmente verificar a condição de CCL como informação heurística estática pelo fato de que o valor dos custos das colunas não se alterarem durante a execução do algoritmo, não sendo necessário efetuar cálculos para obter novos valores de n_j no decorrer da execução do mesmo.

2.3.2 Informação Heurística Dinâmica Baseada no Custo de Cobertura

A informação heurística dinâmica baseada no custo de cobertura é abreviada no presente trabalho como CCB. Para seu cálculo é necessário definir a cardinalidade de j em relação a solução s, denotada por $card_j(s)$ que indica a quantidade de linhas cobertas pela coluna j mas não cobertas por nenhuma outra coluna na solução parcial s. Assim, define-se a informação heurística (Lessing *et al.*, 2004) conforme Equação 5:

$$n_j = \frac{card_j(s)}{c_j}, \tag{5}$$

com c_j sendo o custo da coluna j. É bastante interessante notar que n_j é diretamente proporcional a $card_j(s)$ e inversamente proporcional a c_j. Assim, quanto maior $card_j(s)$ em relação a c_j, maior será n_j. Outra leitura também é possível, sendo quanto menor o valor de c_j em relação a $card_j(s)$, maior o valor de n_j.

Vale notar o caráter dinâmico de CCB pelo fato de $card_j(s)$ ter de ser recalculado para cada coluna candidatada j à cada adição de coluna pelo passo de construção de cada

formiga, no sentido de que tais informações são necessárias para se decidir qual a próxima coluna a ser adicionada.

2.3.3 Considerações Sobre Informação Heurística

Outros tipos de informação heurística nos moldes estático e dinâmico podem ser verificados em Lessing *et al.* (2004), cujo objetivo principal do trabalho é a comparação de desempenho de diferentes tipos de informação heurística usando algoritmos heurísticos ACO para PCC. Em tal trabalho verifica-se os tipos de informação heurística estáticos baseados em custo de coluna (*column costs*) e custo lagrangeano (*Lagrangean costs*), bem como os tipos dinâmicos com base em custo de cobertura (*cover costs*), custo de cobertura lagrangeno (*Lagrangean cover costs*), custo de cobertura de Marchiori e Steenback (*Marchiori e Steenback cover costs*), custo de cobertura lagrangeano de Marchiori e Steenback com custos normalizados (*Marchiori e Steenback Lagrangean cover costs with normalized costs*) e limites inferiores (*lower bounds*), de forma que também traz experimentos comparando as diferentes abordagens.

2.4 Algoritmo Guloso

A informação heurística desempenha o papel de conhecimento específico do problema, e está relacionada com o algoritmo heurístico utilizado em algoritmos construtivos, como por exemplo, no algoritmo guloso (*greedy* ou míope). Tal algoritmo guloso é apresentado na Figura 2.3 e consiste de a cada passo escolher a coluna que maximize a função gulosa, que no caso é a informação heurística CCV.

```
 Solucao GreedyPcc::executar(){
1    S = Ø ;  // S é a solução inicial, que começa vazia.
2    U = M ;  // U representa as linhas a serem cobertas, que começa com todas.
3    card = 0 ;
4    valorFuncaoGreedyJMelhor = ∞ ;

5   while( U ≠ Ø ){
6      achou = false ;
7      for(each j∈N ){
8         if( j∉S ){
9            card = |M_j ∩ U| ;
10           if( card > 0 ){
11              valorFuncaoGreedyJ = card / custo[ j ] ;
12              if( valorFuncaoGreedyJ < valorFuncaoGreedyJMelhor ){
13                 jMelhor = j ;
14                 valorFuncaoGreedyJMelhor = valorFuncaoGreedyJ ;
15                 achou = true ;
16              }
17           }
18        }
19     }

20     if( not achou ){
21        escrever("Nao existe ao menos uma solucao factivel para este PCC, ");
22        escrever("pois a matriz possui linha que nenhuma coluna cobre.");
23        exit(EXIT_FAILURE);
24     }

25     U = U \ M_jMelhor ;
26     S = S U {jMelhor} ;
27  }
28  return( S );
 }
```

Figura 2.3. Algoritmo heurístico greedy para PCC

É importante notar na Figura 2.3 que a linha que faz a identificação do algoritmo contém o termo `Solucao` antes dos termos que designam o nome do algoritmo. Tal termo indica que o algoritmo retorna um valor que é do tipo `Solucao`, de forma que tal retorno é efetivado na linha 28. O mesmo ocorre com outros algoritmos que tenham sintaxe semelhante no presente trabalho.

2.5 Eliminação de Colunas Redundantes

O processo de eliminação de colunas redundantes consiste na retirada de colunas que cobrem linhas cobertas por outras colunas, desde que o custo de cobertura das linhas da coluna a ser excluída não aumente com a remoção de tal coluna. O algoritmo para efetuar esse processo é apresentado na Figura 2.4, o qual verifica cada linha i da coluna na matriz, se a cobertura dessa linha é maior que 2 e desde que a coluna não esteja classificada como imprescindível,

conforme a linha 11 do pseudo-código. Se a coluna não é imprescindível para a composição da solução, a coluna é descartada e a quantidade de linhas cobertas pela coluna é decrementada e a coluna descartada é removida da solução, caso contrário a coluna permanece compondo a solução corrente.

```
 Algoritmo Eliminar Colunas Redundantes
 Considere: // Definições.
1   I: conjunto de linhas.
2   J: conjunto de colunas.
3    α_i : conjunto de colunas que cobrem a linha i.
4    β_i : conjunto de linhas cobertas pela coluna j.
5    w_i : número de colunas em S que cobrem a linha i, i∈S .

 Faça: // Operações
6   // Inicialização do registro de cobertura.
7   for(i=1; i<=|M|; i++){
8       w_i=|S∩α_i| ;
9   }
10  for(i ∈ βj){
11    if(wi ≥ 2){
12      S ← S − {j};
13      wi ← wi − 1;
14    }
15  }
```

Figura 2.4. Pseudo-código do algoritmo de eliminar colunas redundantes

2.6 Busca Local

A fim de melhorar o desempenho dos algoritmos ACO pode ser utilizado uma busca local, sendo esta geralmente uma heurística melhorativa aplicada depois de quantidades de iterações determinadas de acordo com a situação em questão. Assim, o algoritmo ACO – que é construtivo – constrói soluções de acordo com uma análise abrangente no espaço de busca e, em seguida, passa tal solução para a busca local tentar melhorá-la de acordo com critérios que realizam uma análise local relativa a tal solução no espaço de busca.

Considerando as opções relativas ao PCC, as opções relativas à busca local seguem.

- NBL: indica que nenhuma busca local está em uso;
- JB2: indica que está em uso busca local baseada no trabalho de Jacobs e Brusco (1995); e
- RFL (versão beta): indica que se está usando busca local baseada no trabalho de Yagiura *et al.* (2006).

Assim, as seções subsequentes apresentam como as buscas locais foram adaptadas de suas origens, o funcionamento de tais buscas e os aspectos mais relevantes da utilização destas.

2.6.1 NBL

No *Ant System* original aplicado ao PCV (Colorni *et al.*, 1991; Dorigo, 1992; Dorigo *et al.*, 1996) não havia a utilização de busca local. Com a aplicação da busca local foi possível obter um ganho de desempenho no *Ant System* (Dorigo e Stützle, 2004). No que diz respeito à aplicação de ACO para o PCC tem-se que o trabalho de Leguizamon e Michalewicz (2000) aplica o AS ao PCC sem a utilização de busca local. Os trabalhos Lessing *et al.* (2004) e Lessing (2004) aplicam diversos algoritmos ACO ao PCC com e sem busca local, realizando comparações entre as diferentes configurações, sendo uma variação do R_FLIP a principal busca local utilizada.

2.6.2 JB2

A busca local JB2 é inspirada no trabalho de Jacobs e Brusco (1995). Tal trabalho apresenta um algoritmo completo para aplicação ao PCC baseado em *Simulated Annealing* (SA) possuindo um módulo principal chamado *Annealing Routine*, o módulo de construção chamado *Construct Module* e o módulo de busca local chamado *Search Module*. Dessa forma, o *Annealing Routine* chama inicialmente e uma única vez o *Construct Module* que constrói uma solução utilizando uma heurística *greedy* baseada no trabalho de Balas e Ho (1980). Depois de construída a solução inicial, o módulo principal define a heurística baseada em SA, a qual em determinada etapa – que é executada a cada iteração do algoritmo – procura por uma solução vizinha da solução em questão, o que é feito pelo *Search Module*.

Algoritmo baseado no *Search Module* com o nome de *Neighborhood Algorithm* também é utilizado na aplicação de AS para o PCC em Hadji *et al.* (2000), de forma que o *Neighborhood Algorithm* é chamado apenas uma vez ao final do algoritmo com o objetivo de melhorar a melhor solução encontrada pelo AS para o PCC utilizado – aspecto que também é citado em descrito em Dorigo e Stützle (2004). O trabalho de Hadji *et al.* (2000) possui uma descrição sucinta do *Neighborhood Algorithm* que pode facilitar o entendimento de forma abrangente do mesmo e do *Search Module* de Jacobs e Brusco (1995).

O algoritmo de busca local JB2 utilizado no presente trabalho consiste, de forma sucinta, de primeiramente remover-se uma determinada quantidade D de colunas da solução as quais são consideradas piores pelo algoritmo, o que provavelmente resultará em tornar várias linhas descobertas, sendo que em seguida o algoritmo efetua a cobertura de tais colunas de acordo com critério próprio de escolha de colunas. É importante notar que se trata de um algoritmo melhorativo e que não realiza a busca de forma a verificar completamente determinadas vizinhanças da solução original. Os mecanismos são descritos pelas seções subseqüentes.

O Algoritmo

Dessa forma, tem-se que o *Search Module* recebe uma solução e deve procurar por uma solução em sua vizinhança com o objetivo de melhorar a solução recebida. A busca local identificada neste trabalho por JB2 refere-se à algoritmo inspirado no *Search Module* de Jacobs e Brusco (1995) e no *Neighborhood Algorithm* de Hadji *et al.* (2000), com modificações em relação à ambos. A descrição dos parâmetros – e um possível protótipo de função – de tal algoritmo são apresentados na Figura 2.5, onde verifica-se que o nome utilizado para a função é `aplicarSearchModuleJb2`.

```
aplicarSearchModuleJb2(Solucao * S_original , double ρ1 , double ρ2 ,
                       InstanciaPcc *p_instanciaPcc);
/* Descrição dos parâmetros:
   * S_original : Solução a ser melhorada, passado por referência.
     ρ1 : A taxa de supressão de colunas da solução S_original .
     ρ2 : A taxa de redução/ampliação do maior custo dentre as cols. em S_original .
    *p_instanciaPcc: Instância do Pcc, passado por referência, contendo:
        M : O conjunto de todas as linhas.
        N : O conjunto de todas as colunas.
        I(j) : Linhas cobertas pela coluna j , ∀ j∈J . */
```

Figura 2.5. Apresentação do algoritmo de busca local JB2

Em tal Figura, tem-se que $S_{original}$ é a solução a ser modificada sendo passada por referência à função `aplicarSearchModuleJb2`. O parâmetro ρ_1 indica a taxa de remoção (supressão) de colunas da solução $S_{original}$, que culminará na definição de quantas colunas serão retiradas da solução na Etapa 2. Já o parâmetro ρ_2 é a taxa que influencia que valores de custo de colunas com base em $S_{original}$ serão consideradas quando a Etapa 4 de Re-efetuar cobertura referente colunas removidas for executada. O parâmetro `p_instanciaPcc` é um objeto passado por referência que contém estruturas de dados relativas a instância do PCC em questão, sendo relevante para `aplicarSearchModuleJb2` o conjunto de todas as linhas sendo representado por M, o conjunto de todas as colunas sendo representado por N e as linhas cobertas por cada coluna representadas por $I(j) \quad \forall j \in N$.

Com essa descrição dos parâmetros presente na Figura 2.5, passa-se para o algoritmo por meio do exposto na Figura 2.6 onde pode-se verificar o funcionamento de `aplicarSearchModuleJb2`.

```
aplicarSearchModuleJb2(Solucao * S_original , double ρ1 , double ρ2 ,
                       InstanciaPcc *p_instanciaPcc){
1   // 1. Inicializar.
2   S = S_original ;  OrdenarEmOrdemNatural( S );
3   ns = |S| ;                                  // Obter o número de colunas em S .
4   qs =custo[* S .ultimo()];                   // Q(S) = max(c_j | j∈S) .
5   D = teto(ρ1·ns) ;   E = teto(ρ2·qs) ;
6   inicializarVetorW();
7   qtLinhasCobertas=m;
8   R = N \ S ;                        // Atribuição feita de acordo com Figura 2.8.

9   // 2. Remover D colunas de S atualizando vetorW e qtLinhasCobertas.
10  d = 0 ;
11  while( d<D ){
12     k =Selecionar aleatoriamente uma coluna k , (k∈S) ;
13     S = S \ {k} ;  R = R ∪ {k} ;
14     for(each i∈I(k) ) vetorW[i]=vetorW[i]-1;
15     d=d+1;
16  }
17  atualizarQtLinhasCobertas();

18  // 3. Definir RE (recordadas) como RE={j|custo[j]≤E ∀ j∈R} .
19  RE=Ø ;  for(each j_R∈R ) if(custo[j_R] <= E)  RE = RE∪{j_R} ;

20  /* 4. Re-efetuar cobertura de colunas removidas atualizando vetorW e
          qtLinhasCobertas. */
21  while(qtLinhasCobertas ≠ m){
22     for(each j∈J ) card[j]=0;
23     for(each j_RE∈RE )
24        for(each i∈I(j_RE) )
25           if(vetorW[i]==0) card[j_RE]=card[j_RE]+1;

26     for(each j∈N ) beta[j]=0.0;
27     betaMin=SENTINELA_SUPERIOR;  jBetaMin=n;          // Uma coluna invalida.
28     for(each j_RE∈RE ){
29        if(card[j_RE]!=0) beta[j_RE]=custo[j_RE]/card[j_RE];
30        else beta[j_RE]=SENTINELA_SUPERIOR-1.0;
31        if(beta[j] < betaMin){ betaMin=beta[j]; jBetaMin=j; }
32     }

33     K = Ø ;                        // Definir K={j|beta[j]=betaMin ∀ j∈RE} .
34     for(each j_RE∈RE ) if(beta[j_RE]==betaMin)  K = K∪{j_RE} ;
35     k = Selecionar aleatoriamente uma coluna k ( k∈K );
36     S = S∪{k} ;   R = R\{k} ;   RE = RE\{k} ;

37     for(each i∈I(k) ) vetorW[i]=vetorW[i]+1;
38     atualizarQtLinhasCobertas();
39  }

40  // 5. Eliminar colunas redundantes atualizando vetorW.
41  eliminarColunasRedundantes( S , vetorW);

42  // 6. Finalizar.
43  if( S<S_original )  S_original = S ;
 }
```

Figura 2.6. Algoritmo de busca local JB2

Nota-se na linha de identificação do algoritmo da Figura 2.6 que o parâmetro S_{original} possui um "*" antecedendo seu nome, o que indica que tal parâmetro é passado por referência, sendo que o mesmo ocorre em algoritmos que tenha construção semelhante. Ainda por meio da referida Figura 2.6 pode-se verificar o funcionamento de cada etapa, conforme segue.

1. Inicializar

Nesta Etapa, S recebe $S_{original}$ de forma que o valor de $S_{original}$ não é modificado no decorrer do algoritmo, mas apenas na Etapa Finalizar. Assim a variável utilizada e modificada pelo algoritmo é S. A complexidade de tempo da atribuição de uma solução à outra é $O(|S_{lado\ direito}|)$.

A Linha 2 traz a chamada para ordenar as colunas da solução S em ordem natural, o que significa as colunas de S serão ordenadas em ordem ascendente com base nos valores c_j, sendo os valores repetidos de c_j ordenados entre si de forma descendente em relação ao número de colunas cobertas pela coluna j, ressaltando que a ordenação deve ocorrer para todos os $j \in S$. Dessa forma, a coluna com menor custo estará na primeira posição e a de maior custo estará na última posição;

A Linha 3 define ns como sendo a quantidade de colunas em S e a Linha 4 define qs como sendo o valor do custo da coluna de maior custo, que é aquela que está na última posição da solução, após a ordenação de S em ordem natural realizada na Linha 2.

A Linha 5 define D como sendo proporcional à quantidade de colunas de S – representado por ns – com base no parâmetro ρ_1. O valor D representa quantas colunas serão retiradas da solução para as quais a cobertura das linhas que se tornar pendente será recoberta pela seleção de novas colunas na tentativa de melhorar a solução nesse processo. Assim se ρ_1 for 0, nenhuma coluna será retirada – e conseqüentemente nenhuma será adicionada –, caso ρ_1 seja 1, todas as colunas serão retiradas o que fará com que a cobertura seja totalmente nova, situação esta que não daria a importância nenhuma à solução original recebida para processamento, o que não seria adequado.

A Linha 5 também define E como sendo proporcional ao maior custo dentre as colunas de S – que é representado por qs – com base no parâmetro ρ_2. O valor D será utilizado para dizer o valor máximo do custo que terão as colunas que serão consideradas para inserção na solução a fim de re-efetuar a cobertura retirada. Assim, se o valor ρ_2 for 0, nenhuma coluna será considerada (pois geralmente nenhuma coluna tem custo 0) o que poderá causar problemas, se ρ_2 for 1, as colunas consideradas terão custo no máximo igual ao custo da coluna que possui maior custo na solução $S_{original}$, se o valor de ρ_2 for 2 o custo

das colunas consideradas terão custo de no máximo duas vezes o valor do maior custo da coluna que possui maior custo em $S_{original}$, e assim sucessivamente.

O `vetorW` é um vetor com uma posição para cada linha i das linhas M da instância do problema em questão, sendo sua função armazenar em cada posição i a quantidade de colunas que cobrem tal linha i. A Linha 6 realiza a chamada a função que vai inicializar o `vetorW`, função esta que é detalhada na Figura 2.7.

```
 inicializarVetorW(){
1   for(each i∈I ) vetorW[i]=0;            // Inicializar vetorW[i] ∀i∈I com 0.
2   // Inicializar vetorW[i] como nro.de cols.em S que cobrem a linha i, ∀i∈I.
3   for(each j∈S )
4      for(each i∈I(j) )
5         vetorW[i]=vetorW[i]+1;
 }
```

Figura 2.7. Inicializar `vetorW`

Nesta Figura 2.7 verifica-se que primeiramente realiza-se a inicialização de todas as posições de `vetorW` com 0. Após isso tem-se dois laços `for` aninhados, onde o mais externo percorre as colunas j em S, sendo para cada um desses j de S o laço mais interno percorre as linhas i cobertas por tal coluna j, de forma que para cada cobertura que ocorre por motivo de alguma coluna j à linha i, o valor de `vetorW[i]` (posição que representa a linha i no `vetorW`) tem seu valor acrescido de 1.

Com foco na Figura 2.6, tem-se que a Linha 7 inicializa o valor de qtLinhasCobertas (quantidade de linhas cobertas) como m, que é a quantidade de linhas existentes na instância do PCC, as quais devem ser cobertas. Inicializa-se com m porque a solução S no momento da atribuição está cobrindo todas as linhas, pois S recém-recebeu o valor de $S_{original}$, que é a solução recebida pelo algoritmo – e espera-se que seja uma solução completa.

A Linha 8 faz com que R receba as colunas que estão em N exceto aquelas que estão em S. Tal atribuição é realizada de acordo com a Figura 2.8, com as Linhas de 1 a 4 fazendo com forma que R receba N e se ordene R em ordem natural – conforme conceitualização já apresentado. Em seguida, as Linhas de 6 e 16, tendo como base as sequencias S e R em ordem natural, retira colunas de R que estão em S, e faz R ser o conjunto de todas as colunas que não estão na solução. A implementação foi feita dessa forma como uma tentativa de aproveitar a ordenação natural para realizar tal operação de forma a obter uma complexidade de tempo melhor em comparação com uma implementação trivial – onde a cada elemento j_S de S, percorreria-se cada j_R de R desordenado, onde, encontrando o j_S faria com que o laço voltasse para o laço mais externo.

```
1   R = Ø ;
2   for(each j∈N )
3       R = R∪{j} ;
4   OrdenarEmOrdemNatural( R );
5   /* Com as sequencias S e R em ordem natural, retirar colunas de R que estao
       em S, e fazer R ser o conjunto de todas as colunas que não estão na
       solução. Feito dessa forma para aproveitar a ordenacao. */
6    j_R =R.begin();
7   for(each j_S∈S ){
8      encontrou=false;
9      while((not encontrou) and ( j_R ≠ R.end()))
10        if( j_S == j_R ){
11            R = R\{j_R} ;
12           encontrou=true;
13        }
14        else
15            j_R =próximo elemento de R ;
16  }
```

Figura 2.8. Atribuição R=N\S

2. Remover D colunas de S atualizando vetorW

Nesta etapa, inicialmente d recebe 0 conforme Linha 10. Em seguida na Linha 11, tem-se um laço que é executado D vezes – conforme incremento de d que ocorre na Linha 15 –, sendo tal laço equivalente ao comando **for**(int d=0; d<D; d++) na linguagem C++. O conteúdo do laço começa com a Linha 12 fazendo k receber uma coluna selecionada aleatoriamente dentre as colunas de S, seguido pela remoção de k de S e inserção de k em R. Tal movimento da coluna k faz com que o vetorW fique em um estado desatualizado em relação à solução S, fazendo-se necessário que a atualização seja feita, o que ocorre na Linha 14 com a remoção de uma "cobertura" de cada uma das linhas i que são cobertas por k – que agora não mais faz parte da solução S.

Depois do laço, a Linha 17 efetua a chamada da função para atualizar a quantidade de linhas cobertas, dado que tal valor ficou desatualizado depois da execução do laço de remoção de colunas de S recém-executado. Tal função é executada fora do referido laço para não ser necessário realizar a verificação para linhas que são cobertas por diversas colunas. A Figura 2.9 contém o algoritmo.

```
 atualizarQtLinhasCobertas(){
1   qtLinhasCobertas=0;      // Atualizar qtLinhasCobertas= |i|vetorW[i]=0,∀i∈I| .
2   for(each i∈I ) if(vetorW[i] ≠ 0) qtLinhasCobertas=qtLinhasCobertas+1;
 }
```

Figura 2.9. Atualizar qtLinhasCobertas

Como pode-se verificar por tal Figura, a Linha 1 faz a quantidade de linhas cobertas assumir o valor 0 e depois, a Linha 2 realiza uma laço que percorre todas as linhas do vetorW,

fazendo com que ao final do laço, o valor da quantidade de linhas cobertas seja igual a quantidade de linhas que têm seu valor no `vetorW` igual ou maior que 1.

3. Definir RE (recordadas) como $RE = \{ j | custo[j] \leq E \, \forall \, j \in R \}$

Esta Etapa insere colunas no conjunto RE – o nome RE indica colunas recordadas – para serem utilizadas posteriormente como base para re-efetuar a cobertura retirada. Assim, passa a fazer parte do conjunto RE cada coluna j em R que possua $custo[j] \leq E$, conforme denotam as Linhas 18 e 19.

4. Re-efetuar cobertura referente colunas removidas atualizando `vetorW` e `qtLinhasCobertas`

Esta Etapa é realizada pelo laço `while` que vai da Linha 20 até a Linha 39, sendo o laço executado enquanto existem linhas descobertas na instância do PCC em questão, o que é feito no laço da forma $qtLinhasCobertas \neq m$, significando assim que deve-se executar o laço enquanto a quantidade de linhas cobertas não for igual à quantidade total de linhas a serem cobertas – o que no contexto é equivalente à dizer que o laço deve ser executado enquanto $qtLinhasCobertas < m$, que expressa de forma mais intuitiva a condição do laço.

Assim, a cada iteração do laço são feitas tentativas de cobrir mais linhas da instância do PCC e, para tal, são utilizadas informações a respeito das colunas. Primeiramente, as Linhas de 22 a 25 realizam o cálculo da cardinalidade das colunas existentes em RE. A primeira dessas Linhas faz com que a cardinalidade de todas as colunas em N sejam inicializadas com 0, em seguida percorre-se todas as colunas existente em RE e, para cada uma dessas colunas, percorre-se todas as linhas da instância do PCC cobertas por tal coluna, e para cada uma dessas linhas, se elas cobrem alguma linha que ainda não é coberta pela solução S, o valor da cardinalidade de sua respectiva coluna em RE é incrementado em 1 unidade. Assim, a cardinalidade de j em relação a S indica "a quantidade de linhas cobertas pela coluna j fora da solução S que não são cobertas por nenhuma coluna em S". De posse deste cálculo, pode-se passar para o passo seguinte.

O próximo passo é dado pela definição de um vetor `beta`, que contém uma posição para cada coluna em N. A Linha 26 realiza a inicialização de todos os seus valores como 0.0. Em seguida é inicializado `betaMin` como um valor positivo máximo possível representável pela tipo de variável na plataforma computacional sendo utilizada, logicamente considerado $+\infty$ e expresso no algoritmo como SENTINELA_SUPERIOR. O valor de `jBetaMin` indica a coluna referente ao `betaMin`, inicializado com uma coluna inválida. Em seguida a Linha 28 inicia um laço que percorre todas as colunas j_RE em RE realizando o cálculo de `beta` para cada uma delas. Tal cálculo é feito pelas Linhas 29 e 30, onde se a cardinalidade de j_RE é

diferente de 0 (o que no contexto é equivalente a ser maior ou igual a 1) o valor de beta é dado pela Equação 6:

$$beta[\,j\_RE\,] = \frac{custo[\,j\_RE\,]}{card[\,j\_RE\,]} \quad (6)$$

E caso a cardinalidade de j_RE seja 0, o valor de `beta` fica sendo o dado pela Equação 7:

$$beta[\,j\_RE\,] = SENTINELA\_SUPERIOR - 1.0 \quad (7)$$

Dessa forma, garante-se que o a coluna com valor de beta sendo $\text{SENTINELA\_SUPERIOR} - 1.0$ seja selecionado para substituir o que tiver $SENTINELA\_SUPERIOR$, que possui uma coluna inválida e iria causar problemas. Feito isto, é escolhido o menor valor de `beta` – pela Linha 31 – para que as colunas que o possuam façam parte do conjunto de colunas K – feito pelas Linhas 33a 34, que são as candidatas a serem incluídas na solução S.

Após isso, a Linha 35 escolhe aleatoriamente uma coluna k dentre as existente em K. Tal coluna k é então incluída em S e removida de R e de RE, conforme Linha 36.

Depois, é necessário atualizar o `vetorW` para a coluna k recém adicionada fazendo com que para cada linha i coberta pela coluna k, o valor de `vetorW[i]` seja incrementado de 1, o que é feito pela Linha 37. Em seguida a Linha 38 realiza a chamada à função que atualiza a quantidade de linhas cobertas, função esta de acordo com a Figura 2.9 e sua descrição em textos adjacentes à mesma.

5. Eliminar colunas redundantes atualizando vetorW

A eliminação de colunas redundantes elimina da solução colunas que se retiradas apenas diminuirão seu custo sem que a solução passe descobrir alguma linha. A chamada ocorre na Linha 41 e acontece de acordo com o algoritmo descrito na seção 2.5 (página 37) e representado na Figura 2.4 (página 38).

6. Finalizar

A solução S é a que está sendo trabalhada pelo algoritmo, de forma que deve ser atribuída à variável recebida por referência $S_{original}$ para que seus valores estejam atualizados quando a função `aplicarSearchModuleJb2` terminar sua execução. Porém, antes de fazer isso, esta Etapa – conforme Linha 42 – verifica se a solução S trabalhada pela busca local em questão é melhor que a solução $S_{original}$ recebida e, caso seja, $S_{original}$ recebe o valor de S, e a solução trabalhada pela busca local terá seus valores atualizados na solução a ser utilizada pela função que chamou `aplicarSearchModuleJb2`. Tal funcionalidade se fez necessária a partir de experimentos preliminares que indicavam a existência de situações em que a busca local JB2 piorava a solução original fornecida.

Considerações sobre JB2

Um aspecto bastante interessante na comparação do JB2 com os de Jacobs e Brusco (1995) e de Hadji *et al.* (2000) é a forma que o laço de re-efetuar cobertura implementa a sua condição de execução, feita no JB2 de forma a utilizar a variável `qtLinhasCobertas`, verificando a cada iteração se esta variável indica que a quantidade de linhas cobertas pela solução S é igual à quantidade de linhas existentes em N.

2.6.3 RFL

A busca local referida no presente trabalho como RFL tem sua definição principal baseada no 3-FNLS proposto em Yagiura *et al.* (2006) e a inspiração de maneira de aplicação é baseada em Lessing *et al.* (2004) e Lessing (2004).

O algoritmo 3-FNLS – e adaptações, como é o caso do RFL – trabalha com o conceito de vizinhança. Considerando a solução representada pelo vetor binário $x=(x_1,\ldots,x_n)$, a vizinhança *r-flip* de x é definida por $NB_r(x)$. Assim, para um $r\geq 1$ tem-se (Yagiura *et al.*, 2006) a Equação 8:

$$NB_r(x)=\left\{x'\in\{0,1\}^n \middle| d(x,x')\leq r\right\}, \tag{8}$$

onde tem-se pela Equação 9:

$$d(x,x')=\left|\left\{j\in\{1,\ldots,n\} \middle| x_j\neq x'_j\right\}\right| \tag{9}$$

denota a distância Hamming entre $x=(x_1,\ldots,x_n)$ e $x'=(x'_1,\ldots,x'_n)$. Sendo assim, $NB_r(x)$ é o conjunto de soluções vizinhas de x, que é obtido ao trocar (*flip*, do inglês) o valor de máximo r variáveis do vetor binário x. Sendo assim, o nome *r-flip* vem dessa característica, valendo ressaltar que trocar o valor de x_j significa que se este for 0 deverá passar a valer 1 e se for 1 se tornará 0. Dado que quando x_j é 0 indica que a coluna j não faz parte da solução e quando x_j é igual a 1 denota que j está na solução, o fato de invertê-los representa operações de remoção ou adição de colunas da solução, de acordo com o caso em questão. É importante também notar que a vizinhança denotada por $NB_r(x)$ inclui também $NB_{r-1}(x)$, com $r\geq 1$ como já especificado. Assim, no caso do 3-FNLS e do RFL em que $r=3$, tem-se que $NB_3(x)\supseteq NB_2(x)$ e $NB_2\supseteq NB_1(x)$, valendo notar que a situação de igualdade dos conjunto existem pelo fato de o tamanho de x ser 2, não possibilitando uma vizinhança de $r=3$ e o caso de x ter tamanho 1, o que permitirá apenas a aplicação da parte referente ao 1-*flip*.

O trabalho de Yagiura *et al.* (2006) propõe o algoritmo 3-FNLS, ou *3-Flip Neighborhood Local Search* (*for the Set Covering Problem)* também chamado de *Iterated R-Flip* por Lessing *et al.* (2004), que a partir de uma solução *greedy* inicial realiza busca na

vizinhança de distância de no máximo 3 distâncias Hamming tentando encontrar a melhor solução existente em tal vizinhança primando por obter assim um ótimo local em relação à solução inicial. Faz parte de tal processo de busca a utilização da relaxação lagrangeana por meio dos multiplicadores de lagrange, que são fornecidos pelo método subgradiente, conforme pode ser visto em Fisher (1981), (Yagiura *et al.*, 2006) e de Oliveira (1999). A lógica geral de funcionamento do 3-FNLS pode ser vista na Figura 2.10 considerando-se fazendo parte do algoritmo o texto formatado de forma normal, o texto formatado com tachado simples e desconsiderando do algoritmo o texto que está formatado com sublinhado.

De forma geral, a busca pela vizinhança com distância Hamming, representada por r, de forma que seja $r>1$ é computacionalmente inviável devido ao fato de que uma implementação ingênua geraria um algoritmo de complexidade de tempo $O(n^r)$. Porém o referido trabalho de Yagiura *et al.* (2006) realiza a implementação com $r=3$ – utilizada nas bases do RFL – de maneira que sua utilização seja viável, conforme análise teórica e experimentos contidos em tal trabalho.

Tem-se assim que o RFL é uma heurística melhorativa que vasculha a vizinhança *r-flip* (no caso, 3-*flip*) em busca do ótimo local dessa região, sendo o RFL adequado para ser utilizada como busca local. De maneira sucinta, tem-se que o funcionamento do RFL parte de uma solução original fornecida pelo algoritmo construtivo (para o qual RFL desempenha o papel de busca local) – ao invés de escolher tal solução de forma *greedy*, como o 3-FNLS – e do valor dos multiplicadores lagrangeanos para fazer busca local na vizinhança com distância Hamming $r=3$ em relação à solução original com o objetivo retornar uma solução de melhor qualidade. Vale notar ainda que a busca local RFL descrita no presente trabalho foi utilizada em uma versão de implementação considerada instável pelo autor do presente trabalho – daí a denominação de "RFL (versão beta)".

O Algoritmo

O algoritmo de busca local RFL apresentado neste trabalho é definido na Figura 2.10, de forma que a base do algoritmo intitulado `aplicarThreeFlip(...)` é formada pelo algoritmo chamado 3-FNLS em Yagiura *et al.* (2006) – algoritmo também referido como *Iterated R-Flip* por Lessing *et al.* (2004) – com as alterações propostas pelo presente trabalho para que o RFL possa ser utilizado como uma busca local, alterações estas em parte inspiradas no que é relatado por Lessing *et al.* (2004). Assim, pela referida Figura observa-se em plano de fundo o 3-FNLS com modificações na declaração da função e nos comandos, principalmente no que se refere aos laços de repetição, de modo que o sentido do algoritmo não é modificado. Em seguida, há comandos do 3-FNLS que foram retirados do RFL, conforme indica texto com a formatação tachado simples, ~~`como neste texto exemplo`~~, havendo também a distinção dos

comandos que foram inseridos especificamente para o RFL conforme indica texto sublinhado, conforme este outro texto exemplo.

		Solucao aplicarThreeFlip(Solucao solucao){
Passo 1; Iniciali-zação:	1	$x^* :=$ solucao; //- ~~$x^* := GREEDY;$~~
	2	UB := cost(x^*);
	3	$u_i^{(0)} = min\{c_j/\|S_j\| \| i \in S_j\} \quad \forall i \in M$;
	4	$u^+ :=$ SUBGRADIENTE(UB, $u^{(0)}$);
Passo 2; Primeiro estágio; ~~Fixação de variáveis~~:	5	$N_1 = \emptyset$; $N_0 = \emptyset$; $N_{free} = N$; //~~(N_1, N_0, N_{free}) := FIRST_FIXING(u^+);~~
	6	~~// counter := 0;~~
	7	$x := 0$;
	8	$p_i := min\{c_j \| i \in S_j\} \quad \forall i \in M$;
Passo 3:	9	**while**((time ⩽ time_lim) **and** ((UB-LB) ⩾ (1-EPSI)) **and** (trial < max_trials)){
Passo 4:	10	**while**(**true**){
	11	$x^{\sim} :=$ ONE_FLIP(x);
	12	**if**($x^{\sim} \neq x$){
	13	$x := x^{\sim}$;
	14	**continue**; /* Go back to Passo 4. */ }
Passo 5:	15	$x^{\sim} :=$ TWO_FLIP(x);
	16	**if**($x^{\sim} \neq x$){
	17	$x := x^{\sim}$;
	18	**continue**; /* Go back to Passo 4. */ }
Passo 6:	19	**if**(cost(x) ⩽ L(u^+)) **break**; // Go forward to Step 7.
	20	**else{**
	21	$x^{\sim} :=$ THREE_FLIP(x);
	22	**if**($x^{\sim} \neq x$)
	23	$x := x^{\sim}$;
	24	**continue**; /* Go back to Passo 4. */ }
	25	}
Passo 7:	26	~~// counter := counter + 1;~~
	27	**if**(feasible solutions x were found in Passos 4, 5 and 6){
	28	let x^+ be the one with the minimum cost(x);
	29	**if**(cost(x^+) < UB){
	30	$x^* := x^+$;
	31	UB:=cost(x^*);
	32	~~/* counter := 0 ; */~~ } }
Passo 8:	33	Update the penalty weights p_i by calling UPDATE_PENALTY(x);
	34	//~~ **if**(the penalty weights are increased) or (counter < minitr_ls)~~
	35	//~~ **continue**; // Go back to Passo 3.~~
Passo 9; ~~Estágio de modificação; Fixação de variáveis~~:	~~36~~	~~/* Modify (N_1, N_0, N_{free}) calling MODIFY_FIXING($x^*, x, u^+, N_1, N_0, N_{free}$);*/~~
	~~37~~	//~~ counter := 0;~~
	~~38~~	//~~ // **continue**; // Go back to Passo 3.~~
	~~39~~	trial := trial+1;
	40	}
	41	**return**(x^*);
		}

Figura 2.10. Algoritmo RFL usado como busca local

Nota-se também que o algoritmo RFL – contido na Figura 2.10 – possui chamadas as funções `SUBGRADIENTE(...)` na Linha Erro: Origem da referência não encontrado, `ONE_FLIP(...)` na Linha 11, `TWO_FLIP(...)` na Linha 15, `THREE_FLIP(...)` na Linha 21 e `UPDATE_PENALTY(...)` na Linha 33, sendo que a função de cada um é definida de forma geral como segue – acompanhadas da reprodução da Linha em que cada uma é chamada.

- u^{+} := `SUBGRADIENTE(UB,` $u^{(0)}$ `)`: realiza a chamada à função subgradiente utilizado para fornecer um "bom" vetor de multiplicadores u^{+} de acordo com a relaxação lagrangeana recebendo os parâmetros custo de uma solução inicial dado por UB e e um vetor de multiplicadores inicial, dado por $u^{(0)}$. Informações de como este processo funciona podem ser consultadas em Yagiura *et al.* (2006), Fisher (1981) e de Oliveira (1999);
- $x^{\sim}$:= `ONE_FLIP(` x `)`: identificado no 3-FNLS como 1-FLIP seu objetivo é procurar uma solução $x^{\sim}$ na vizinhança com distância 1 da solução x;
- $x^{\sim}$:= `TWO_FLIP(` x `)`: identificado no 3-FNLS como 2-FLIP tendo a função de procurar na vizinhança com distância 2, já considerando o que foi trabalhado na vizinhança com distância 1 por `ONE_FLIP(...)`;
- $x^{\sim}$:= `THREE_FLIP(` x `)`: identificado no 3-FNLS como 3-FLIP com a função de procurar na vizinhança com distância 3, já considerando o que foi trabalhado com distância 2 por `TWO_FLIP(...)`, e este por sua vez usou o que foi trabalhado em `ONE_FLIP(...)`; e
- `UPDATE_PENALTY(` x `)`: com a função de atualizar as penalidades, como pode ser verificado no em Yagiura *et al.* (2006).

A inicialização dos multiplicadores de Lagrange é feita apenas uma vez na inicialização do algoritmo pelo fato de os valores a serem utilizados serem os mesmos nesse caso, não sendo executada repetidamente a cada chamada à RFL.

Considerações Sobre RFL

Em relação ao JB2, é relevante ressaltar que existe algoritmo melhorado no trabalho Brusco *et al.* (1999), o qual descreve várias melhorias em relação à busca local de Jacobs e Brusco (1995), as quais poderiam ser aplicadas à desdobramentos futuros do presente trabalho.

Ainda sobre JB2, tem-se que seu mecanismo de busca – que consiste em retirar determinada quantidade de colunas da solução e substituí-las em seguida – não o faz referente à uma vizinhança bem definida em relação à solução recebida, o que pode fazer a busca um tanto esparsa no que diz respeito a posicionamento no espaço de busca, sugerindo que podem estar sendo valorizadas questões mais relativas à qualidade de cada coluna individualmente. Situação diferente da apresentada pelo RFL, que realiza a busca local de forma a verificar a

vizinhança de $r = 3$ de uma solução, fazendo com que a posição de uma solução no espaço de busca seja utilizada de forma positiva para o algoritmo, o que remete à possibilidade de poder-se guiar algoritmos por regiões de acordo com o potencial em uma busca do gênero.

A respeito da implementação gerada e utilizada pelo presente trabalho é importante observar que o JB2 é passível de algumas otimizações e o RFL ainda se encontra em versão instável, designado de "versão beta" pelo fato de sua implementação ainda não ter sido considerada pelo presente autor como completamente finalizada. O código-fonte da implementação do 3-FNLS foi cedido pelos autores em Yagiura *et al.* (2006), de forma que tal código foi adaptado dando origem ao RFL do presente trabalho.

No geral, mais comparações de desempenho entre os aspectos tempo de execução e qualidade da solução podem ser encontradas no Capítulo 6 (página 105) que contém os experimentos e resultados do presente trabalho.

2.7 Considerações

Diante do exposto no presente Capítulo, tem-se a fundamentação teórica a respeito do PCC e seus aspectos inerentes de forma agrupada e relacionada, permitindo que se tenha uma idéia clara do que é dependente do problema e pode ser aproveitado na confecção de algoritmos heurísticos para construção e melhoria de soluções para o PCC.

3 Meta-heurística ACO

O *Ant System* (AS) (Colorni *et al.*, 1991; Dorigo, 1992; Dorigo *et al.*, 1996) é um algoritmo heurístico inspirado no comportamento de formigas naturais e foi criado para ser utilizada na obtenção de soluções em problemas de otimização combinatória, tendo como sua primeira aplicação o problema do caixeiro viajante (PCV). Após sua criação, muitas variações do *Ant System* foram elaboradas para diversos tipos de problemas. A fim de classificar e generalizar as diferentes variações do AS foi criada a meta-heurística *Ant Colony Optimization* (ACO) (Dorigo e Stützle, 2004).

Surgiram sucessores e extensões do AS, como no caso do *MAX-MIN Ant System* (MMAS) (Stützle e Hoos, 1995), *Ant Colony System* (ACS) (Dorigo e Gambardella, 1997), *ANTS* (Maniezzo, 1999) , *MMAS-ACS-Hybrid* (Stutzle e Hoos, 1997), *HyperCube* (Blum *et al.*, 2001), *BeamACO* (Blum, 2005), dentre outros.

Têm-se a aplicação do ACO à diversas categorias de problemas, tais como: problema generalizado de atribuição (PGA) (Ramalhinho-Lourenço e Serra, 1998), problema quadrático de alocação (PQA) (Maniezzo e Colorni, 1999), problema de cobertura de conjunto (PCC) (Leguizamon e Michalewicz, 2000), (Hadji *et al.*, 2000), problema da mochila (PM) (Dorigo e Stützle, 2004), (Alaya *et al.*, 2004) e problema de roteamento de pacotes em redes de computadores (Dorigo e Stützle, 2004) para citar alguns.

Nas seções subseqüentes do presente Capítulo têm-se a inspiração e os princípios teóricos da meta-heurística ACO.

3.1 Inspiração e Surgimento do Conceito de Formigas Artificiais

Na natureza pode-se observar diferentes tipos de mecanismos, *e.g.* em comportamentos sociais de animais, funcionalidades corporais, ecossistemas, dentre outros, sendo que tais mecanismos podem ser atribuídos à evolução natural. Dentre estes, o comportamento das formigas chama a atenção, pois são animais bastante limitados – com pouca visão –, porém o comportamento denotado por toda uma colônia de formigas é bastante complexo e

organizado. As formigas se comunicam umas com as outras por meio do resíduo de feromônio que depositam nos objetos – ou o chão – que têm contato. Nesse cenário, um contexto que é base para o AS é o mecanismo que surge dessa comunicação na formação de caminhos que, em um determinado momento, a maioria das formigas passa a utilizar. Vale notar que a medida que o tempo passa, o feromônio depositado vai evaporando, necessitando sempre que a comunicação se mantenha ativa (Colorni *et al.*, 1991).

Dessa forma, as formigas conseguem obter um bom caminho entre dois pontos, útil por exemplo para determinar um caminho entre o formigueiro e uma fonte de comida, pois na primeira decisão, a quantidade de formigas a escolher o caminho mais curto e o caminho mais longo devem ser aproximadamente o mesmo, de modo que elas escolhem o caminho com base no feromônio encontrado, e como ainda não existe resíduo de feromônio nos lugares que estão passando, a probabilidade é igual. Porém, na segunda decisão, as formigas que percorreram o caminho mais curto já estão voltando, depositando ainda mais feromônio, enquanto as que foram pelo caminho mais longo ainda estão completando a primeira transição. Dessa forma, a decisão das formigas numa segunda iteração deverá tender para o caminho mais curto, conforme ilustra a Figura 3.1.

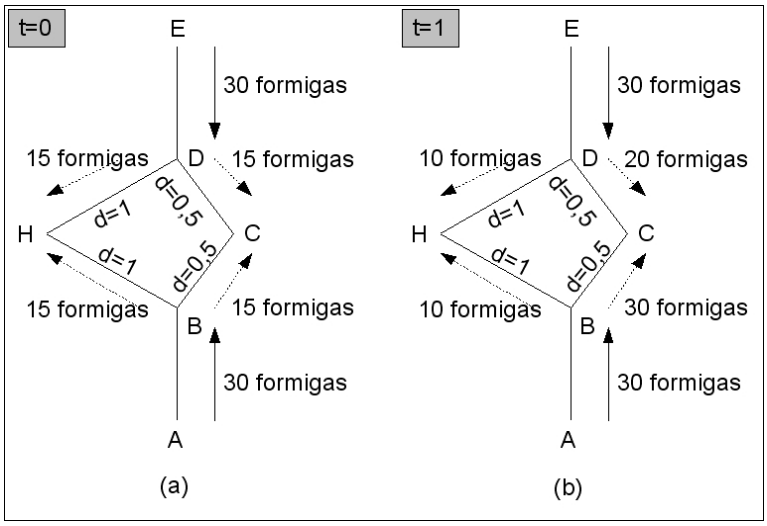


Figura 3.1. *Exemplo de movimento das formigas artificiais*

Assim, t representa o tempo do sistema, d é a distância do caminho, A e E são os pontos extremos que as formigas devem atingir e H e C são pontos intermediários, o qual as formigas devem decidir qual caminho tomar. O interessante é que escolham o mais curto, no caso, C.

A observação das características de colônia de formigas deu origem ao AS (Colorni *et al.*, 1991; Dorigo, 1992). O AS se utiliza dessas características para formular um sistema de colônia de formigas artificiais. Nesse sistema tem-se que o tempo é discreto e as formigas artificias podem possuir alguma memória (*e.g.* para armazenar o caminho de volta) e não são completamente cegas, pois haverá um fator visibilidade (*e.g.* informação heurística) do

caminho que influenciará probabilisticamente na decisão, além do feromônio depositado, que será representado por um número no caminho, sendo atualizado pelas formigas de acordo com as regras definidas pelo algoritmo utilizado. O resíduo de feromônio é o valor que influenciará probabilisticamente a decisão da formiga por meio da informação de escolha, que será formada juntamente com a informação heurística (o caminho que contiver mais informação de escolha, terá mais chance de ser escolhido). Isso resulta em um sistema que exibe comportamento **auto-catalítico**: as formigas vão reforçando as melhores soluções causando uma rápida convergência. Também se aproveita da sinergia de várias formigas trabalhando individualmente e se comunicando coletivamente, de forma que cada formiga ajusta seu trabalho levando em conta as informações provenientes dessa comunicação. É notório que o algoritmo tem como base uma população: um conjunto de soluções são construídas simultaneamente. O AS é considerado uma meta-heurística de caráter **construtivo** (constrói novas soluções) e **melhorativo** (melhora solução já existentes) ao mesmo tempo.

3.2 Princípios Teóricos das Formigas Artificiais

Para formalizar-se matematicamente as formigas artificiais, toma-se como base o problema de se encontrar o menor caminho (PMC) entre dois vértices (componentes) predefinidos em um grafo. Em Dorigo e Stützle (2004) existe a descrição do *Simple-ACO* para início de estudo. O *Simple-ACO* é um sistema de otimização por colônia de formigas artificiais simples, utilizado para encontrar caminhos de menor custo em um grafo, imitando assim de forma aproximada o comportamento das formigas naturais as quais inspiraram a meta-heurística em questão. Tal grafo é dado por $G=(N, A)$ onde G é um grafo conexo, N é o conjunto de n ($n=|N|$) vértices e A é o conjunto de arestas (não-direcionado) conectando os vértices. Os vértices os quais se deseja encontrar um menor caminho possível entre eles são dados no início do problema e denotados por ninho (*nest*) e fonte de comida (*food source*).

Para cada arco (i, j) do grafo G é associada a variável τ_{ij} chamada resíduo de feromônio (artificial). Estes resíduos são lidos e escritos pelas formigas, sendo a quantidade de feromônio proporcional a utilidade do arco associado a este. No começo do processo de busca um valor constante de feromônio é associado a cada arco (*e.g.* $\tau_{ij}=1$, $\forall (i, j)\in A$). Quando localizada em um vértice i uma formiga k usa o resíduo de feromônio para computar a probabilidade de selecionar j como vértice seguinte, conforme indica a Equação 10:

$$p_{ij}^{k}=\begin{cases}\dfrac{\tau_{ij}^{\alpha}}{\sum_{l\in N_i^k}\tau_{ij}^{\alpha}} & , se\ j\in N_i^k \\ 0 & , se\ j\notin N_i^k\end{cases}, \qquad (10)$$

onde N_i^k é o conjunto de vértices candidatos (suscetíveis à escolha) da formiga k quando no vértice i. No *Simple-ACO,* o conjunto de vértices candidatos contém todos os vértices conectados ao vértice i, com exceção do vértice predecessor, para evitar que se volte ao vértice recém-saído. A regra de atualização de feromônio é dada por meio da Equação 11:

$$\tau = \tau_{ij} + \Delta\tau^k \tag{11}$$

Com $\Delta\tau^k$ podendo ser um valor constante no caso mais simples. Deve haver também um mecanismo de evaporação, para que escolhas ruins possam ser "esquecidas" do sistema. Pode ser dado conforme Equação 12:

$$\tau_{ij} = (1-\rho)\tau_{ij}, \ \forall (i,j) \in A \tag{12}$$

É considerado uma iteração no sistema quando ocorre um ciclo composto de:

- movimentos das formigas;
- evaporação de feromônio; e
- depósito de feromônio.

3.3 Representação Genérica do Problema para ACO

Segundo Dorigo e Stützle (2004) um algoritmo da classe ACO pode ser aplicado a qualquer problema de otimização combinatória para o qual uma heurística construtiva possa ser definida. Para tal, define-se na mesma referência uma representação genérica de problema que podem ser utilizados com ACO a fim de que as formigas artificiais possam utilizar tal estrutura.

Considera-se o problema de minimização contido na Equação 13:

$$(S, f, \Omega), \tag{13}$$

onde:

- S é o conjunto de soluções candidatas;
- f é a função objetivo que associa um valor de uma função objetivo (custo) $f(s,t)$ para cada solução candidata $s \in S$; e
- $\Omega(t)$ é um conjunto de restrições.

O parâmetro t em $f(s,t)$ e $\Omega(t)$ indica que a função objetivo e as restrições podem ser dependentes do tempo. Como exemplo de sua relevância têm-se aplicações em problemas dinâmicos, os quais suas condições podem mudar no decorrer do tempo.

O objetivo é encontrar uma solução s^* que seja factível e globalmente ótima. Em um problema de minimização isto significa dizer que s^* é uma solução factível com o menor custo dentre todas.

Assim, o problema de otimização combinatória (S, f, Ω) é mapeado em um problema que pode ser caracterizado pela lista de itens:

- um conjunto finito $C=\{c_{1,}c_{2,}...,c_{|C|}\}$ de componentes, onde $|C|$ é a quantidade de componentes;
- os estados do problema são definidos em termos de sequências $x=\langle c_i, c_j, ..., c_h, ...\rangle$ de tamanho finito sobre os elementos de C. O conjunto de todos os possíveis estados é denotado por χ. A quantidade de componentes na sequência é expressa por $|x|$, sendo seu tamanho máximo limitado por uma constante positiva $n<+\infty$;
- o conjunto de soluções candidatas S é um conjunto de χ (*i.e.*, $S\subseteq\chi$);
- um conjunto de estados factíveis $\tilde{\chi}$, com $\tilde{\chi}\subseteq\chi$, definido via teste dependente do problema que verifica que não é impossível completar uma sequência $x\in\tilde{\chi}$ em uma solução satisfazendo as restrições Ω. Nota-se por esta definição, que a factibilidade de um estado $x\in\tilde{\chi}$ deve ser interpretado em um senso *fraco*. De fato, isso não garante que um completo s de x existe de tal modo que $s\in\tilde{\chi}$;
- um conjunto não vazio S^* de soluções ótimas, com $S^*\subseteq\tilde{\chi}$ e $S^*\subseteq S$;
- um custo $g(s,t)$ é associado com cada candidato $s\in S$. Na maioria dos casos $g(s,t)\equiv f(s,t)$, $\forall s\in\tilde{S}$, onde $\tilde{S}\subseteq S$ é o conjunto de soluções candidatas, obtido de S via as restrições $\Omega(t)$;
- em alguns casos o custo, ou a estimativa de um custo, $J(x,t)$ pode ser associado com estados outros que não sejam soluções candidatas. Se x_j pode ser obtido adicionando-se componentes a um estado x_i, então $J(x_i,t)\leqslant J(x_j,t)$. Nota-se que $J(s,t)\equiv g(s,t)$.

Dada esta formulação, formigas artificiais constroem soluções fazendo caminhadas aleatórias no grafo completo $G_C=(C,L)$, com os vértices sendo os componentes C, e o conjunto L conecta completamente os componentes C. O grafo G_C é chamado grafo de construção e os elementos de L são nomeados conexões.

Ainda segundo Dorigo e Stützle (2004) as restrições do problema $\Omega(t)$ são implementadas na política seguida pelas formigas artificiais, o que permite certo grau de flexibilidade. Na verdade, dependendo do problema de otimização combinatória considerado, pode-se implementar as restrições de uma maneira fixa, permitindo que as formigas construam apenas soluções factíveis, ou de forma mais flexível, permitindo que as formigas possam construir soluções infactíveis (*i.e.*, soluções candidatas em $S\setminus\tilde{S}$), sendo que podem ser penalizadas como uma função de seus graus de infactibilidade.

3.4 A Meta-heurística

Com base nas considerações das Seções anteriores e no trabalho de (Dorigo e Stützle, 2004), apresenta-se o algoritmo da meta-heurística ACO mais genérico, que pode ser especializado

em algoritmo heurístico aplicado a problemas dinâmicos como também pode ser convertido em algoritmo heurístico para aplicação a problemas estáticos. Tal algoritmo é descrito pelo pseudo-código da Figura 3.2.

```
 metaHeuristicaACO(){
1   escalonarAtividades{
2      construirSolucoesComFormigas();
3      atualizarFeromonios();
4      executarAcoesDaemon();  // Opcional.
5   }
 }
```

Figura 3.2. Pseudo-código do algoritmo ACO genérico retirado de Dorigo e Stützle (2004)

Com base nesta meta-heurística mais genérica é possível se definir a meta-heurística ACO para problemas estáticos, conforme ilustra Figura 3.3 a partir de trabalho de Dorigo e Stützle (2004).

```
 metaHeuristicaACOEstatica(){
1   Setar parâmetros; Inicializar resíduos de feromônio;
2   while( condição de parada não satisfeita ){
3      construirSolucoesComFormigas();
4      aplicarBuscaLocal();  // Opcional.
5      atualizarFeromonios();
6   }
 }
```

Figura 3.3. Pseudo-código do algoritmo ACO genérico para problemas estáticos retirado de Dorigo e Stützle (2004)

Assim, a partir da meta-heurística ACO para problemas estáticos pode-se definir algoritmos heurísticos para que possam efetivamente obter soluções para tais problemas.

3.5 Considerações

A meta-heurística ACO para problemas estáticos descrita na Figura 3.3 é a que é utilizada no decorrer do presente trabalho para aplicação de algoritmos heurísticos ACO para o PCC.

4 Algoritmos Heurísticos Baseados em ACO Aplicados ao PCC

Com base nos fundamentos da meta-heurística ACO exposta no Capítulo 3, o presente Capítulo apresenta algoritmos heurísticos baseados em ACO aplicados ao PCC existentes na literatura. Tal apresentação ocorre de uma maneira organizada de aspectos mais genéricos para mais específicos dos algoritmos. Tais fundamentações servirão também como base para as proposições realizadas no Capítulo 5.

É importante ressaltar uma questão de terminologia, a qual no âmbito de algoritmos heurísticos ACO para PCC têm-se que uma coluna está sendo referida pelo termo componente.

4.1 Algoritmo Heurístico ACO Genérico Aplicado ao PCC

Algoritmos heurísticos ACO possuem várias características comuns. Nesta seção são apresentadas as características mais comumente utilizadas em definições de algoritmo dessa classe, de forma que um algoritmo genérico é exposto. Não é definido aqui um algoritmo completo, mas sim uma base que serve seções subseqüentes com conteúdo geral.

4.1.1 Grafo de Construção

As colunas são representadas pelos componentes do conjunto C, com um componente extra que não está relacionado com nenhuma das colunas do PCC, sendo componentes os vértices do grafo. O componente extra geralmente serve de ponto de partida para as formigas, quando se quer que as formigas fiquem no grafo, mas em uma situação de neutralidade em relação às colunas. Este componente extra também é chamado de componente *dummy*. Estruturas equivalentes ao grafo de construção dessa forma também são utilizados em Leguizamon e Michalewicz (2000) e Hadji *et al.* (2000) como pode também ser visto em Dorigo e Stützle (2004), ao passo que em Lessing *et al.* (2004) também se faz uso de tal estrutura de forma semelhante, porém não é claro se existe ou não o componente extra (ou *dummy*).

4.1.2 Restrições

O algoritmo deve gerir como as restrições do problema serão tratadas, pois como não faz sentido para o PCC construir soluções com componentes (colunas) repetidos, não é permitido à formiga adicionar à solução um componente (coluna) que já se encontra nesta. Tal mecanismo é elaborado através de um conjunto de componentes candidatos da solução parcial s_k pertencente à formiga k, conjunto este denotado por $N(s_k)$. Assim, a formiga k deve procurar novos componentes para a solução s_k em $N(s_k)$, este que contém todos os componentes de C que não estão na solução s_k. Uma modificação que poderia ser útil é a diminuição do tamanho do conjunto de componentes candidatos de acordo com algum critério de avaliação de potencial dos componentes, possivelmente de acordo com algum critério baseado no tipo de problema – no caso o PCC –, porém, deve ser levado em conta o fato de que a diminuição do conjunto de componentes candidatos antes da escolha de um componente pode levar o algoritmo a desconsiderar solução ótima do problema. Assim, é importante verificar se tal modificação resultará em melhor comportamento do algoritmo.

No PCC, para que a solução seja factível, ela deve cobrir todas as linhas da instância do problema. Assim, a cada passo de construção a formiga deve verificar se sua solução em construção já cobriu todas as linhas. Nas implementações deste trabalho o mecanismo funciona exatamente como o exposto, porém, idéias de fazer com que o sistema possa trabalhar com soluções intermediárias infactíveis podem ser utilizadas, um exemplo disto é o algoritmo *R-Flip* iterativo em Yagiura *et al.* (2006). Isto significaria que a formiga poderia – de acordo com algum critério – encerrar a fase de construção da solução de forma a retornar uma solução que não cobrisse todas as linhas do problema. Nesse caso, algum tipo de controle deve ser feito de modo a garantir que apenas soluções intermediárias – que são produzidas pelo algoritmo, mas não é nenhuma delas que efetivamente é retornada pelo algoritmo – sejam infactíveis. A justificativa para se permitir tal flexibilidade é que soluções ótimas geralmente são mínimas, o que significa que se for retirado qualquer uma de suas colunas esta se tornará infactível. De fato, se uma coluna pode ser retirada sem que a cobertura seja modificada, isto significa que o custo de tal coluna na solução era desnecessário. Dessa forma, o que poderia ser feito é permitir a formiga que elaborou uma solução infactível depositar feromônio, mas não permitir que tal solução possa ser retornada como melhor solução pelo algoritmo, sendo que ainda é necessário garantir que o algoritmo produza alguma solução que seja factível. Uma abordagem poderia ser metade das formigas serem obrigadas a produzir soluções factíveis enquanto que a outra metade poderia elaborar soluções infactíveis, e – nos casos em que apenas uma formiga deposita feromônio a cada iteração – poderia-se permitir que a formiga que produziu a melhor solução factível deposite feromônio e uma das que produziram soluções infactíveis também deposite feromônio. Ficam em aberto as questões de

como deve ser a valoração da função objetivo, como escolher as formigas que podem produzir soluções infactíveis, como deve ser a escolha da formiga que irá depositar feromônio e qual seria o critério de parada do algoritmo de construção da solução dessas formigas. Indícios de respostas para essas questões podem ser encontradas no já referido trabalho de Yagiura *et al.* (2006), que se utiliza de mecanismos semelhantes.

4.1.3 Resíduos de Feromônio

Os resíduos de feromônio são associados com os componentes, desta forma o feromônio τ_j associado com o vértice j mede a desejabilidade ("o quanto é bom") de se incluir a coluna j na solução. A forma de cálculo dos valores de τ_j são dados pelos algoritmos heurísticos ACO específicos.

4.1.4 Informação Heurística

A informação heurística referente ao componente j é dada por η_j^β, onde β é o fator de adequação que indica a relevância a ser dada a tal informação na regra de decisão utilizada. A informação heurística pode ser definida de várias maneiras, conforme Seção 2.3 na página 34.

4.1.5 Informação de Escolha

A informação de escolha é o resultado da multiplicação do fator feromônio pelo fator informação heurística, e na implementação do presente trabalho a informação de escolha é armazenada já multiplicada, seguindo linhas gerais do que Dorigo e Stützle (2004) faz para o PCV.

4.1.6 Algoritmo Heurístico

O algoritmo heurístico genérico do ACO para o PCC é apresentado na Figura 4.1. O algoritmo consiste de várias inicializações, seguidas por um laço de construção, melhoramento e avaliação das soluções, cuja melhor solução encontrada é retornada ao término do laço.

Por `setarParametros()` entende-se a obtenção e processamento dos parâmetros para que estes possam ser utilizados de forma adequada pelo algoritmo.

O procedimento `inicializarGerais()` faz a inicialização da melhor solução encontrada (como a pior possível), de marcações de tempo, dos critérios de parada e de outras questões de implementação que se fizerem necessárias.

Os procedimentos `inicializarLimitesFeromonio()` e `inicializarFeromonio()` trabalham para inicializar de forma correta os valores dos limites de feromônio e das demais questões relativas ao feromônio.

O procedimento `calcAllInfEscolhaEst()` realiza o cálculo de toda a informação de escolha estática através de seus dois fatores:

- fator feromônio; e
- fator informação heurística.

Ocorre que ambos podem ser classificados em estático e dinâmico, cujo significado depende do fator em questão. De forma resumida, têm-se que o cálculo do fator de feromônio estático deve ser realizado uma vez a cada iteração enquanto que o cálculo de feromônio dinâmico deve ser também a cada iteração, porém o tipo de consulta de feromônio utilizado implica em um cálculo que leva em consideração a solução parcialmente construída pela formiga fazendo com que a cada passo de construções cálculos tenham de ser realizados em relação ao fator feromônio – e conseqüentemente a informação de escolha – a cada aplicação do passo de construção.

Dessa maneira, o procedimento `calcAllInfEscolhaEst()` realiza o cálculo de todas as informações de escolha quando a informação heurística é denominada estática. Caso esta seja dinâmica o cálculo deve ser realizado a cada passo de construção e um cálculo no início da iteração não será capaz de amenizar tal complexidade algorítmica.

```
 Solucao AcoPcc::executar(){
1   setarParametros();
2   inicializarGerais();
3   inicializarFeromonio();
4   calcAllInfEscolhaEst();
5   while( criterioQtIteracoes           and
           criterioQtIteracoesSemMelhora and
           criterioTempo                 and
           criterioEncontrarOtima        and
           criterioEncontrarMc ){
6     construirSolucoesComFormigas();
7     atualizarFeromonio();
8     calcAllInfEscolhaEst();
9     calcularGerais();  // Dentre outros, recalcular critérios de parada.
10  }
11  return( melhorSolucaoEncontrada );
 }
```

Figura 4.1. Execução principal de algoritmo ACO genérico para PCC (denotado por AcoPcc no pseudo-código)

Em seguida tem-se o laço `while` e sua condição constituída de vários critérios, a saber:

- `criterioQtIteracoes`: verdadeiro se a quantidade de iterações for menor ou igual ao valor setado nos parâmetros, falso caso contrário;

- `criterioQtIteracoesSemMelhora`: verdadeiro se a quantidade de iterações em que não houve melhora na solução incumbente da heurística é menor que o valor estipulado nos parâmetros, falso caso contrário;
- `criterioTempo`: verdadeiro se o tempo de execução é menor que o tempo máximo para execução da heurística setado nos parâmetros, falso caso contrário;
- `criterioEncontrarOtima`: verdadeiro se não encontrou a solução ótima, falso caso contrário;
- `criterioEncontrarMc`: verdadeiro se não encontrou melhor solução conhecida, falso caso contrário;

Em relação aos critérios `criterioEncontrarOtima` e `criterioEncontrarMc` dois aspectos devem ser considerados:

- A solução ótima ou a melhor solução conhecida é informada no início do algoritmo e é expressa através do custo da solução e de sua quantidade de componentes (tamanho);
- A solução ótima ou melhor conhecida informada no início do algoritmo possui informação indicando se esta é comprovadamente ótima ou não;
- Para que os critérios sejam satisfeitos deve ser encontrada uma solução de custo igual ou melhor (menor) e tamanho igual ou melhor (menor) àquela que se deseja comparar;
- Através dos parâmetros `pararSeEncontrarOtima` e `pararSeEncontrarMc` é possível definir se o algoritmo deve considerar a solução informada como ótima ou melhor conhecida ou não. Assim:
    - Se `pararSeEncontrarOtima` for verdadeiro, o algoritmo pára caso a solução informada seja ótima e tal solução (ou melhor) for encontrada;
    - Se `pararSeEncontrarMc` for verdadeiro, o algoritmo pára mesmo no caso de a solução informada não for ótima e tal solução (ou melhor) for encontrada;

Assim, pode-se ler o laço e sua condição da forma como indica a Figura 4.2.

Executar laço enquanto
(*a quantidade de iterações for menor ou igual à quantidade máxima de iterações permitidas****) e***
(*a quantidade de iterações sem que haja melhora da solução incumbente for menor ou igual à quantidade máxima de soluções sem melhora permitidas****) e***
(*o tempo de execução for menor ou igual que o tempo máximo de execução permitido****) e***
(*se for para parar quando a solução ótima for encontrada, se a solução informada for ótima e a solução ótima (ou melhor) não for encontrada****) e***
(*se for para parar quando a melhor solução conhecida for encontrada, se a solução informada não for ótima e a melhor solução conhecida – ou melhor – não for encontrada****).***

Figura 4.2. Interpretação em linguagem natural do laço de execução e sua condição

O conteúdo do laço inicia-se pela execução de `construirSolucoesComFormigas()` que faz com que cada formiga construa uma solução. Em seguida viria a aplicação opcional de busca local – e no caso do PCC pode ser executado também o `eliminarColunasRedundantes()` – que geralmente aplica um algoritmo heurístico melhorativo sobre a melhor solução obtida pelas formigas, porém no presente trabalho tal algoritmo heurístico melhorativo está sendo aplicado pela própria formiga ao término da construção de uma solução. A execução de `atualizarResiduosFeromonio()` realiza todos os cálculos referentes ao feromônio necessários a cada iteração, contendo assim `evaporarFeromonio()`, `depositarFeromonio()`, `atualizarLimitesFeromonio()`, `adequarIntervaloFeromonio()` e `reInicializarFeromonio()`. Na seqüência tem-se `calcAllInfEscolhaEst()` que é idêntico ao executado antes da entrada no laço, valendo notar que será executado apenas se a informação heurística for estática e que – caso seja executado – devido ao valor de feromônio recém-atualizado essa chamada deve cuidar para que os valores da informação de escolha reflitam a nova situação de feromônio. Como última etapa do laço tem-se `calcularGerais()` que inclui, assim como `inicializarGerais()`, marcações de tempo, cálculo dos critérios de parada para a próxima iteração e de outras questões de implementação que se fizerem necessárias. Por fim o algoritmo retorna a melhor solução encontrada através do comando `return`.

4.1.7 Construção da Solução

O procedimento de construção de solução da formiga realiza a construção de uma solução por uma formiga e é dado genericamente por `Formiga::construirSolucao()`. Para cada iteração, este procedimento é chamado uma vez para cada formiga, como é denotado pela Figura 4.3, na qual vê-se o procedimento `construirSolucoesComFormigas()` que é chamado pelo algoritmo heurístico a cada iteração, conforme Figura 4.1.

```
 AcoPcc::construirSolucoesComFormigas(){
1   for( int k = 0; k < qtFomigas; k++ ){
2      formiga[k].construirSolucao();
3   }
 }
```

Figura 4.3. Construir soluções com formigas de AcoPcc

4.1.8 Busca Local

A busca local denotada pela função `aplicarBuscaLocal()` chamada dentro da função `construirSolucao()` em um algoritmo completamente definido pode ser uma das buscas locais indicadas na seção 2.6. Uma questão bastante relevante à aplicação de busca local para PCC é como ela é utilizada. Enquanto que abordagens mais genéricas utilizam aplicação de busca local apenas sobre a melhor solução obtida ao fim da execução do algoritmo, conforme

pode ser visto no algoritmo AS_LM, no presente trabalho a busca local é aplicada sobre cada solução construída.

4.1.9 Atualização de Feromônio

A atualização de feromônio é feita basicamente pela evaporação e pelo depósito de feromônio por uma ou mais formigas, sendo definida pelo algoritmo heurístico ACO em utilização.

4.1.10 Considerações Sobre Algoritmo Heurístico ACO Genérico Aplicado ao PCC

Com a abordagem genérica apresentada na presente seção têm-se uma base para aplicação de algoritmo ACO para PCC, sendo tal base aproveitada no decorrer do presente trabalho. Isso também é importante para implementação, facilitando abstrações para utilização de uma linguagem de programação orientada a objetos.

4.2 AS Aplicado ao PCC

Em Dorigo e Stützle (2004) é realizada uma discussão de algumas formas de se aplicar ACO ao PCC, sendo investigadas duas abordagens: a de Leguizamon e Michalewicz (2000) que é referida como AS_LM e a de Hadji *et al.* (2000) que será referenciada como AS_HRTB.

As características **grafo de construção**, **restrições**, **resíduos de feromônio**, **informação heurística**, **informação de escolha** e **algoritmo heurístico** são tomadas de Subseções equivalentes contidas na Seção 4.1, que aborda a aplicação de algoritmos heurísticos ACO para PCC de forma genérica. As demais características são abordadas pelas Subseções subseqüentes.

4.2.1 Construção da Solução

Na Figura 4.4 é ilustrado o procedimento `construirSolucao()` da formiga, este realiza um laço que se mantém enquanto a solução não está completa, cujos comandos adicionam um componente a solução sendo construída a cada iteração, através da chamada do procedimento `aplicarPassoDeConstrucao()` que se encontra com mais detalhes na Figura 4.5.

```
 FormigaPccAsLm::construirSolucao(){
1   while( not verificarSeSolucaoEstaCompleta() ){
2      aplicarPassoDeConstrucao();  // Chama aplicarRegraDeDecisao() internamente.
3   }
 }
```

Figura 4.4. Construir solução de formiga de AS_LM

Na Figura 4.5 vê-se o procedimento `aplicarPassoDeConstrucao()` constituído pelas chamadas de `aplicarRegraDeDecisao()` e `setEAddComptAtual()`, sendo este último responsável

por setar como componente atual aquele selecionado pelo procedimento `aplicarRegraDeDecisao()`, fazendo também com que o componente seja adicionado na solução da formiga sendo construída, além de realizar outras atividades necessárias ao correto funcionamento do algoritmo.

```
 FormigaPccAsLm::aplicarPassoDeConstrucao(){
1   aplicarRegraDeDecisao();  // Decidir qual é o próximo componente.
2   setEAddComptAtual();  // Mover a formiga para o novo componente.
 }
```

Figura 4.5. Aplicar passo de construção de formiga de AS_LM

Assim, na construção da solução, o AS_LM sabendo que cada componente j possui feromônio τ_j e informação heurística η_j, faz com que cada formiga k comece com uma solução vazia, construindo então a solução iterativamente adicionando componentes j até que todas as linhas estejam sendo cobertas, *i.e.* a solução esteja completa. Assim, cada próximo componente é selecionado de acordo com a **regra de probabilidade de decisão** p exeutada por `aplicarRegraDeDecisao()` chamado no algoritmo da Figura 4.5 e cuja definição é apresentada na Equação 14:

$$p_j^k(t) = \begin{cases} \dfrac{\tau_j^\alpha \cdot \eta_j^\beta}{\sum_{h \in N_{factível}(s_k)} \tau_h^\alpha \cdot \eta_h^\beta} & se\ j \in N_{factível}(s_k) \\ 0 & se\ j \notin N_{factível}(s_k) \end{cases} \quad (14)$$

Onde os valores dos parâmetros α e β indicam a importância do fator feromônio e da informação heurística, respectivamente. Têm-se que $N_{factível}(s_k)$ é o conjunto de componentes candidatos factíveis da solução s_k da formiga k antes da adição de novo componente até a iteração t, consistindo de todas os componentes que cobrem ao menos uma linha que ainda não foi coberta. O algoritmo AS_HRTB realiza o mesmo processo de seleção, com a diferença de que, ao invés de começar com uma solução vazia, uma solução parcial inicial é elaborada com componentes selecionados aleatoriamente, a fim de aumentar a diversificação da solução. Outra diferença é que o AS_HRTB possui uma remoção de componentes redundantes da solução ao fim de cada passo de construção.

4.2.2 Atualização de Feromônio

Para a atualização de feromônio, primeiro deve ocorrer a evaporação do feromônio para que então ocorra o depósito do mesmo. A evaporação é dada pela Equação 15:

$$\tau_i \leftarrow (1-\rho)\tau_i,\ \forall i \in C \quad (15)$$

E o depósito do feromônio é dado pela Equação 16:

$$\tau_i \leftarrow \tau_i + \sum_{k=1}^{m} \Delta \tau_i^i(t) \quad (16)$$

No AS_HRTB, o valor da atualização é dado por $l \in N^k = \frac{1}{f(s_k)}$, onde $f(s_k)$ é o valor da função objetivo da solução s_k da formiga k, se o componente i é um elemento de s_k, e 0 caso contrário. O algoritmo AS_LM utiliza a mesma regra, com a diferença de que o feromônio depositado é multiplicado pela soma dos custos de todas as colunas na definição do problema.

4.2.3 Busca Local

Com respeito ao aspecto busca local, o AS_HRTB aplica tal busca à melhor solução construída na última iteração do algoritmo. Tal busca local segue o esquema de Jacobs e Brusco (1995) detalhado na Subseção 2.6.2. Não é relatado busca local para AS_LM.

4.2.4 Considerações Sobre AS Aplicado ao PCC

Os resultados computacionais obtidos pelos algoritmos AS_LM e AS_HRTB são bons, porém não alcançam os resultados das técnicas em nível de estado da arte de Caprara *et al.* (1999) e Marchiori e Steenbeek (2000).

4.3 MMAS_L Aplicado ao PCC

A primeira aplicação do *Max-Min Ant System* (MMAS_L) ao PCC é feita pelo trabalho Lessing *et al.* (2004). A presente Seção descreve tal heurística sob a abordagem geral do presente trabalho.

Os aspectos **grafo de construção**, **restrições**, **resíduos de feromônio**, **informação heurística**, **informação de escolha** e **busca local** são os mesmos que os descritos em Subseções equivalentes a cada um destes na Seção 4.1, que expõe a aplicação de algoritmos heurísticos ACO para o PCC de forma genérica.

4.3.1 Algoritmo Heurístico

O algoritmo heurístico é tomado do algoritmo heurístico ACO aplicado ao PCC apresentado na Subseção 4.1.6 com a inserção do procedimento `inicializarLimitesFeromonio()`, como pode ser visto na Figura 4.6. Tal procedimento é explicado na Subseção 4.3.4.

```
 Solucao AcoPccMmasL::executar(){
1   setarParametros();
2   inicializarGerais();
3   inicializarLimitesFeromonio();
4   inicializarFeromonio();
5   calcAllInfEscolhaEst();
6   while( criterioQtIteracoes           and
           criterioQtIteracoesSemMelhora and
           criterioTempo                 and
           criterioEncontrarOtima        and
           criterioEncontrarMc ){
7      construirSolucoesComFormigas();
8      atualizarResiduosFeromonio();
9      calcAllInfEscolhaEst();
10     calcularGerais();
11  }
12  return( melhorSolucaoEncontrada );
 }
```

Figura 4.6. Execução principal de MMAS_L

4.3.2 Construção da Solução

Como exposto na Figura 4.7, o procedimento `construirSolucao()` da formiga realiza um laço que se mantém enquanto a solução não está completa, cujos comandos adicionam um componente a solução sendo construída a cada iteração, através da chamada do procedimento `aplicarPassoDeConstrucao()` que se encontra com mais detalhes na Figura 4.8. Após uma solução ter sido construída de forma completa por parte do algoritmo construtivo, são chamados procedimentos que objetivam realizar melhorias na solução recém-construída, como no caso do PCC chama-se os algoritmos heurísticos melhorativos `eliminarColunasRedundantes()` e `aplicarBuscaLocal()`.

```
 FormigaPccMmasL::construirSolucao(){
1   while( not verificarSeSolucaoEstaCompleta() ){
2      aplicarPassoDeConstrucao();  // Chama aplicarRegraDeDecisao() internamente.
3   }
4   eliminarColunasRedundantes();
5   aplicarBuscaLocal();
 }
```

Figura 4.7. Construir solução de formiga de MMAS_L

O procedimento `aplicarPassoDeConstrucao()` dado pela Figura 4.8 é constituído de duas etapas denotadas pelos procedimentos `aplicarRegraDeDecisao()` – que será explicado adiante – e `setEAddComptAtual()`. O procedimento o `setEAddComptAtual()` seta como componente atual aquele componente que foi selecionado pelo procedimento de decisão `aplicarRegraDeDecisao()`, faz com que o componente seja adicionado na solução da formiga sendo construída e realiza outras atividades necessárias ao correto funcionamento do algoritmo, como preparar o que é necessário para a correta detecção de que a solução é completa ou não.

```
 FormigaPccMmasL::aplicarPassoDeConstrucao(){
1   aplicarRegraDeDecisao();  // Decidir qual é o próximo componente.
2   setEAddComptAtual();  // Mover a formiga para o novo componente.
 }
```

Figura 4.8. Aplicar passo de construção de formiga de MMAS_L

Assim, o procedimento `aplicarRegraDeDecisao()` chamado na Figura 4.8 utiliza-se de uma **regra de decisão** para decidir de forma probabilisticamente guiada qual novo componente deve ser escolhido para ser incluído na solução. Como guia, a regra de decisão se utiliza da informação de escolha para saber quais as probabilidades de escolha de cada componente da vizinhança. Tal informação de escolha é composta dos valores de feromônio e de informação heurística de cada componente (outras possibilidades são propostas e descritas no Capítulo 5).

Para se obter a probabilidade de escolha de cada componente no momento de aplicar a regra de decisão utiliza-se da probabilidade p. Posto que cada componente j possui feromônio τ_j e informação heurística η_j, a probabilidade p de escolha do componente j pela formiga k é dada pela Equação 17:

$$p_j^k = \begin{cases} \dfrac{\tau_j^\alpha \cdot \eta_j^\beta}{\sum_{h \in N(s_k)} \tau_h^\alpha \cdot \eta_h^\beta} & se\ j \in N(s_k) \\ 0 & se\ j \notin N(s_k) \end{cases} \qquad (17)$$

Os valores dos parâmetros α e β indicam a importância do fator feromônio e da informação heurística, respectivamente. Têm-se também que $N(s_k)$ é a vizinhança considerada de s_k, esta que é a solução parcial da formiga k sendo construída até a referida iteração (iteração que geralmente é indicada por t, podendo ser representada na regra de decisão, porém isto não é feito por questão de simplicidade). Para o PCC, geralmente tem-se que $N(s_k)$ contém todos os componentes em C que não fazem parte de s_k.

4.3.3 Atualização de Feromônio

A atualização de feromônio consiste de dois passos básicos, a evaporação e o depósito de feromônio, conforme é ilustrado na Figura 4.9. Na mesma Figura vê-se outros procedimentos sobre o feromônio os quais serão explicados mais adiante nas subseções 4.3.4 e 4.3.5.

```
 AcoPccMmasL::atualizarResiduosFeromonio(){
1   evaporarFeromonio();
2   depositarFeromonio();
3   atualizarLimitesFeromonio();
4   adequarIntervaloFeromonio();
5   reInicializarFeromonio();
 }
```

Figura 4.9. Atualizar feromônio de MMAS_L

A evaporação de feromônio chamada pelo procedimento `evaporarFeromonio()` acontece da forma exposta na Equação 18:

$$\tau_j = (1-\rho)\cdot\tau_j \,,\ \forall\, j \in C \,, \tag{18}$$

de forma que ρ é um parâmetro que indica a taxa de evaporação do feromônio.

O depósito de feromônio é efetuado pelo procedimento `depositarFeromonio()`, de forma que apenas uma formiga deposita feromônio nos componentes contidos em sua solução. Ocorre da maneira indicada na Equação 19:

$$\tau_j = \tau_j + \Delta\tau \,,\ \forall\, j \in s_{best} \,, \tag{19}$$

onde $\Delta\tau$ é o inverso da função objetivo que deseja-se minimizar, sendo exposto na Equação 20:

$$\Delta\tau = \frac{1}{f(s_{best})} \tag{20}$$

Têm-se ainda que s_{best} indica solução da formiga *best*, formiga esta que pode ser:

- *ib* : do inglês *iteration best*, denota formiga com melhor solução da iteração; ou
- *bs* : do inglês *best so far*, indica a formiga com melhor solução desde o início da execução do algoritmo. Em uma terminologia voltada aos experimentos do Capítulo 6 diz-se "a melhor solução da tentativa".

4.3.4 Limites de Feromônio

Como característica específica do MMAS_L têm-se a delimitação dos valores de feromônio entre o limite mínimo τ_{min} e o limite máximo τ_{max}. Essa delimitação tem o intuito de não permitir que o algoritmo caia em estagnação em determinadas soluções pelo fato de terem um valor demasiadamente alto de feromônio, pois o valor de feromônio não poderá ser maior que τ_{max}. Também faz com que todos os componentes possuam chance significativa de serem escolhidos, pois nenhum terá valor de feromônio menor que τ_{min}. Dessa forma, objetiva-se permitir que as soluções possam se diversificar com probabilidade que seja adequada. Dessa forma tem-se os limites dados pela Equação 21:

$$\tau_j = \left[\tau_j\right]_{\tau_{min}}^{\tau_{max}} \tag{21}$$

Utilizando-se da definição do operador pela Equação 22:

$$\left[x\right]_b^a = \begin{cases} a & se\ x > a \,, \\ b & se\ x < b \,, \\ x & caso\ contrário \end{cases} \tag{22}$$

Dito isto, é necessário saber-se como definir de forma adequado os valores de τ_{min} e τ_{max}. Tais valores podem ser obtidos empiricamente e ajustados de acordo com o problema em questão, conforme pode ser visto em Solnon e Bridge (2006) para problemas de

subconjunto, sendo notório em tal trabalho a característica a qual os valores dos limites de feromônio permanecem o mesmo durante toda a execução do algoritmo.

Existem também considerações analíticas que podem servir de base para o estabelecimento de τ_{min} e τ_{max}, conforme apresenta o trabalho Stützle e Hoos (2000). Para o PCC tem-se a maneira utilizada em Lessing (2004), – esta por sua vez inspirada em descrição de Dorigo e Stützle (2004) – que denota a inicialização dos limites de feromônio da forma utilizada neste trabalho, cuja execução é chamada por `inicializarLimitesFeromonio()` conforme ilustrado na Figura Figura 4.6 e exposto nas Equações 23 e 24:

$$\tau_{max} = \frac{1}{\rho \cdot f(s_{ub})}, \tag{23}$$

onde s_{ub} é uma solução obtida por um algoritmo *greedy*, onde ub indica do inglês *upper bound*, ou limite superior. Tem-se também o valor de τ_{min} dado pela Equação 24:

$$\tau_{min} = \frac{\tau_{max}}{a}, \tag{24}$$

onde a é um parâmetro.

Calculados os limites, todos os valores de feromônio são inicializados como τ_{max}. Na execução de cada iteração, depois de se realizar as atualizações de feromônio, o valor de τ_{max} é atualizado de acordo com a Equação 25:

$$\tau_{max} = \frac{1}{\rho \cdot f(s_{bs})}, \tag{25}$$

com s_{bs} sendo a melhor solução obtida até o momento – conforme já mencionado – e com τ_{min} sendo atualizado da mesma forma que na Equação 24, sendo a atualização de τ_{max} e τ_{min} por iteração sendo realizada por `atualizarLimitesFeromonio()` contido na Figura Figura 4.9. Vale notar, no presente trabalho, que o valor dado por ρ para a finalidade exposta na Equação 25 é substituído por ρ_{base}, que é uma variável independente do ρ da evaporação de feromônio.

Após a atualização dos limites de feromônio, têm-se que fazer com sejam de fato limites para os resíduos de feromônio, o que é feito pelo procedimento `adequarIntervaloFeromonio()` contido na Figura 4.9.

4.3.5 Inicialização de Feromônio e Reinicialização de Feromônio

Calculados os limites de feromônio – conforme a Subseção 4.3.4 – todos os valores de feromônio são inicializados como τ_{max}, por meio de `inicializarLimitesFeromonio()` contido na Figura 4.6.

A cada determinada quantidade de iterações sem melhora da solução realiza-se a reinicialização dos resíduos de feromônio, a fim de evitar um possível comportamento de estagnação do algoritmo, complementando assim a ação os limites de feromônio no mesmo problema da estagnação. Vale notar também que o a reinicialização de feromônio ocorre com o valor de τ_{max} que é calculado a cada iteração. Tal funcionalidade é feita pelo procedimento `reInicializarFeromonio()` da Figura 4.9 no momento correto, conforme indicação da quantidade de iterações sem melhora, lembrando que não sendo o momento correto, o procedimento executa apenas a verificação necessária para obter esta informação.

4.3.6 Considerações sobre MMAS_L Aplicado ao PCC

O trabalho de Lessing *et al.* (2004) realiza tal implementação e apresenta resultados os quais são comparados com os do presente trabalho no Capítulo 6 que contém os experimentos e resultados.

4.4 AS_RC_2 Aplicado ao PCC

O AS_RC_2 é implementado com base no trabalho de Reis e Constantino (2003), que lança as fundações para tal algoritmo e executa testes em parte das instâncias de PCC de *OR-Library* (Beasley, 1990).

As características **grafo de construção**, **restrições**, **resíduos de feromônio**, **informação heurística**, **informação de escolha** e **algoritmo heurístico** são as mesmas que as descritas em Subseções equivalentes a cada uma destas na Seção 4.1, que expõe a aplicação de algoritmos heurísticos ACO para o PCC de forma genérica.

4.4.1 Construção da Solução

A construção de soluções com formigas é feita da mesma forma que a apresentada para a aplicação de algoritmos heurísticos ACO para PCC de forma genérica na subseção 4.1.7. A construção de uma solução por uma formiga é feita da forma ilustrada na Figura 4.10, de forma que é necessário inicializar o conjunto U de linhas não cobertas pelos componentes da solução atual e o vetor w_i, que para cada linha indica a quantidade de colunas na solução s que cobre tal linha.

Figura 4.10. Construir solução de formiga de AS_RC_2

```
 FormigaPccAsRc2::construirSolucao(){
1   S ← ∅ ;
2   U ← M ;  // U é o conjunto de linhas não cobertas.
3   w_i ← 0 , ∀i∈M ;  // w_i é a qt.cols.em S que cobrem a linha i , i∈S .
4   while( not verificarSeSolucaoEstaCompleta() ){
5      aplicarPassoDeConstrucao();  // Chama aplicarRegraDeDecisao() internamente.
6   };
7   eliminarColunasRedundantes();
 }
```

Tal procedimento chama então o laço para construir uma solução, que faz chamada à `aplicarPassoDeConstrucao()`, este ilustrado na .Figura 4.11.

```
 FormigaPccAsRc2::aplicarPassoDeConstrucao(){
1   aplicarRegraDeDecisao();  // Decidir qual é o próximo componente.
2   setEAddComptAtual();  // Mover a formiga para o novo componente.
 }
```

Figura 4.11. Aplicar passo de construção de formiga de AS_RC_2

No AS_RC_2, o procedimento `aplicarRegraDeDecisao()` chamado na Figura 4.11 utiliza-se de uma **regra de decisão** para decidir qual o novo componente que deve ser escolhido para ser incluído na solução. Tal decisão ocorre em duas etapas, sendo a primeira etapa probabilística e a segunda etapa determinística.

A primeira etapa seleciona aleatoriamente uma linha que ainda não é coberta por nenhum dos componentes existentes na solução parcial construída até o momento pela formiga. Vale notar que cada uma das referidas linhas têm a mesma probabilidade de ser selecionada, conforme Equação 26.

$$p_e^k = \begin{cases} \dfrac{1}{|U|} & se\ e \notin U(s_k) \\ 0 & se\ e \in U(s_k) \end{cases}, \ \forall e \in M, \tag{26}$$

onde e é uma linha e $U(s_k)$ é o conjunto de todas as linhas cobertas pelos componentes da solução parcial s_k.

A segunda etapa utiliza a linha selecionada para definir a vizinhança a ser utilizada para escolha do componente a ser adicionado à solução. Assim, define-se vizinhança contendo os componentes que podem ser selecionados como $N(s_k)$. Têm-se então que $N(s_k)$ é a vizinhança considerada de s_k, esta que é a solução parcial da formiga k sendo construída até a referida iteração. Para o PCC têm-se geralmente que $N(s_k)$ contém todos os componentes que estão em C mas que não fazem parte de s_k. No caso do AS_RC_2 para PCC, a vizinhança é definida de forma diferente, sendo mais claro dizer que fazem parte da vizinhança $N(s_k)$ todas os componentes que cobrem a linha atual e exceto aqueles que pertencem à solução parcial s_k sendo construída pela formiga k. Assim têm-se a definição formal na Equação 27:

$$N(e,s_k)=\{j|(j\notin s_k)\wedge(e\notin U)\wedge(a_{ej}=1)\}\ ,\ \forall\ j\in N \tag{27}$$

Por fim, a etapa diz que deve ser selecionado aquele componente da vizinhança que possua a melhor (maior) informação de escolha dentre todas os componentes da vizinhança. Tal informação de escolha é baseada no fato de que cada componente h possui feromônio τ_h e informação heurística η_h, de forma que a escolha do componente j pela formiga k é dado conforme Equação 28:

$$j=argmax_{h\in N(e,s_k)}\{\tau_h^{\alpha}\cdot\eta_h^{\beta}\} \tag{28}$$

Os valores informados através de parâmetros α e β indicam a importância do fator feromônio e da informação heurística respectivamente. O parâmetro ρ indica a taxa de evaporação de feromônio.

4.4.2 Busca Local

No trabalho Reis e Constantino (2003) não é relatado a utilização de busca local.

4.4.3 Atualização de Feromônio

A atualização de feromônio pode ser verificada na Figura 4.12.

```
 AcoPccAsRc2::atualizarResiduosFeromonio(){
1   evaporarFeromonio();
2   depositarFeromonio();
3   reInicializarFeromonio();
 }
```

Figura 4.12. Atualizar feromônio de AS_RC_2

A evaporação de feromônio é chamada por `evaporarFeromonio()` do algoritmo da Figura 4.12 da forma exposta na Equação 29:

$$\tau_j=(1-\rho)\cdot\tau_j\ ,\ \forall\ j\in C \tag{29}$$

O depósito de de feromônio é feito pelo procedimento `depositarFeromonio()` da mesma Figura da maneira apresentada na Equação 30:

$$\tau_j=\tau_j+\Delta\tau\ ,\ \forall\ j\in s_{best} \tag{30}$$

Com $\Delta\tau$ sendo expresso pela Equação 31:

$$\Delta\tau=\left(\frac{f(s_{bs})}{f(s_{best})}\right)^{y} \tag{31}$$

Onde y é um parâmetro utilizado para alterar a importância do feromônio a ser depositado nos componentes e s_{best} indica solução da formiga com índice $best$, índice este definido na seção 4.3.3.

4.4.4 Considerações sobre AS_RC_2 Aplicado ao PCC

Os testes apresentados por Reis e Constantino (2003) são interessantes no sentido de que mostram resultados bastante ajustados de forma a encontrar a solução ótima ou melhor solução conhecida para determinadas instâncias, porém o tempo de execução é elevado, fazendo com que outras abordagens sejam mais vantajosas. Tal demanda de tempo acontece principalmente pelo fato de a quantidade de formigas utilizada ter sido 1000. É interessante notar ainda que ao valor usado para o depósito de feromônio é diferente da maioria das variações de algoritmos heurísticos ACO para PCC e a delimitação da vizinhança pela seleção da linha pode mostrar um caminho promissor no sentido de diminuição da vizinhança de componentes a ser pesquisada a cada passo de construção do algoritmo construtivo.

4.5 Considerações

O presente Capítulo traz uma revisão bibliográfica das fundamentações a respeito de algoritmos heurísticos baseados em ACO aplicados ao PCC, os quais possuem resultados que podem ser usados para comparação em Capítulo oportuno. Também é importante ressaltar que tais fundamentações são parte da base para as proposições realizados no presente trabalho.

5 Proposta

O presente Capítulo apresenta as principais contribuições do presente trabalho, como a definição de novas variações de manipulação de feromônio e novos algoritmos heurísticos baseados em ACO para PCC. Tais contribuições são feitas no sentido de tentar melhorar o desempenho dos algoritmos existentes tanto na qualidade da solução quanto no tempo de execução.

5.1 Manipulação de Feromônio

Uma das características mais relevantes para algoritmos da classe ACO é a forma de utilização do feromônio. O trabalho de Alaya *et al.* (2004) define o algoritmo ACO chamado *Ant-knapsack* que trata do problema da mochila multidimensional (PMM) analisando a manipulação de feromônio de três maneiras distintas, referenciando outros trabalhos e apresentando abordagem própria, conforme segue:

- representação em uma dimensão (*e.g.* vetor ou VET), consulta por componentes (CMP) e depósito de feromônio em cada componente da solução s (CMP), para aumentar a desejabilidade de cada componente de s, de forma que no momento da construção de novas soluções, tais componentes terão mais probabilidade de serem selecionados (Leguizamon e Michalewicz, 1999);
- representação em duas dimensões (*e.g.* matriz de adjacência – MAT), consulta por seqüência de pares de componentes (SEQ) e depósito de feromônio na seqüência de pares de componentes – componentes sucessivos – da solução s (SEQ), com a idéia de aumentar a desejabilidade de se escolher o componente j_{f+1} depois que o último selecionado é j_f (Fidanova, 2002), com f denotando a posição de um componente j na solução s (considerando-se que a solução seja uma seqüência); e
- representação em duas dimensões (*e.g.* matriz de adjacência – MAT), consulta por todos os pares de componentes pertencentes à solução em relação ao componente candidato (todos os pares ou TOD) e depósito de feromônio em todos os pares de componentes distintos da solução s (TOD), para aumentar a desejabilidade de se

escolher juntos componentes de uma solução já encontrada quando uma nova solução está sendo construída (Alaya *et al.*, 2004).

A partir disso **propõe-se** de uma forma genérica a organização da manipulação de feromônio da seguinte forma:

- representação de feromônio (grafo de construção):
    - Uma dimensão (*e.g.* VET);
    - Duas dimensões (*e.g.* MAT);
- consulta de feromônio:
    - Componentes ou CMP;
    - Seqüência de pares ou SEQ;
    - Todos os pares ou TOD;
- depósito de feromônio:
    - Componentes ou CMP;
    - Seqüência de pares ou SEQ;
    - Todos os pares ou TOD.

Como um desdobramento teórico – e até intuitivo – mediante o apresentado, tem-se as combinações de parâmetros permitindo a formulação das configurações contidas na Tabela 5.1.

Tabela 5.1. Possibilidades de configurações de manipulação de feromônio

Código	Qt.de dimensões da repr. de feromônio	Consulta de feromônio	Depósito de feromônio	Certeza da possibilidade de implementação	Certeza de significado
1CC	1	CMP	CMP	SIM	SIM
1CS	1	CMP	SEQ	NÃO[#]	NAO
1CT	1	CMP	TOD	NÃO[#]	NAO
1SC	1	SEQ	CMP	NÃO[!]	NAO
1SS	1	SEQ	SEQ	NÃO[#]	NAO
1ST	1	SEQ	TOD	NÃO[#]	NAO
1TC	1	TOD	CMP	NÃO!	NAO
1TS	1	TOD	SEQ	NÃO[#]	NAO
1TT	1	TOD	TOD	NÃO[#]	NAO
2CC	2	CMP	CMP	SIM[$]	SIM
2CS	2	CMP	SEQ	SIM	SIM[*]
2CT	2	CMP	TOD	SIM	SIM[*]
2SC	2	SEQ	CMP	SIM[%]	NAO
2SS	2	SEQ	SEQ	SIM	SIM[&]
2ST	2	SEQ	TOD	SIM	SIM
2TC	2	TOD	CMP	SIM[%]	NAO
2TS	2	TOD	SEQ	SIM	NÃO
2TT	2	TOD	TOD	SIM	SIM

Com as informações teóricas contidas na Tabela 5.1 constata-se que nem todas as configurações formuláveis têm condições de serem afirmadas de imediato como válidas, pois podem:

- Não fornecer a certeza de que seja possível a implementação em um sistema de computação digital, como explicam os detalhes na Tabela 5.2; e

Tabela 5.2. Sobre a certeza da possibilidade de implementação

Sobre a certeza da possibilidade de implementação	Descrição
NAO[#]	Pois não é possível armazenar todas as dimensões de informação
NAO[!]	Pois não há informação disponível para consulta
SIM[$]	Mas é dispendioso, aproveitando apenas uma dimensão
SIM[%]	Mas com problema

- Mesmo possuindo a certeza de possibilidade de implementação, pode não ter a certeza de possuir um significado válido, ou mesmo possuindo tal significado, pode não ser o mais imediato, conforme coloca a Tabela 5.3.

Tabela 5.3. Sobre a certeza de significado

Sobre a certeza de significado	Descrição
SIM[*]	Pode ser alterada a regra de consulta
SIM[&]	Mas depende do problema

5.1.1 Representação de Feromônio (Grafo de Construção)

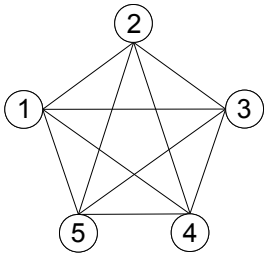
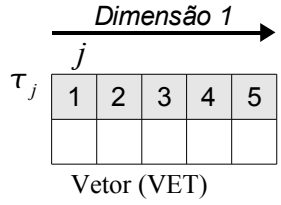
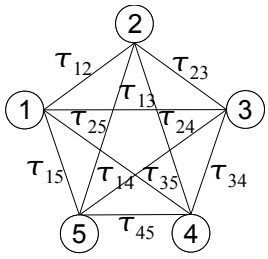
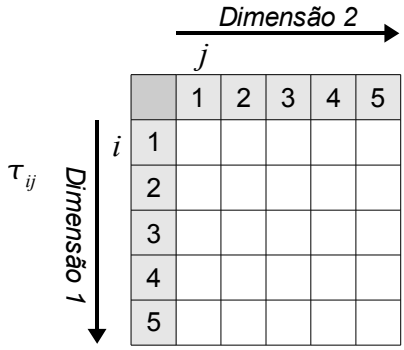
Os resíduos de feromônio podem ser associados com os componentes (seriam vértices no grafo de construção) ou com pares de componentes (que seriam as arestas no grafo de construção). No caso da associação com componentes, o feromônio τ_j referente ao componente j indica a medida de desejabilidade de se incluir o componente j na solução. No caso da associação com pares de componentes, o feromônio τ_{ij} indica a medida de desejabilidade de se escolher os componentes i e j para a mesma solução, porém a forma como isso é feito varia de acordo com o tipo de consulta de feromônio.

Na formulação teórica da aplicação de algoritmos ACO à um problema de otimização combinatória, o feromônio é representado por meio da estrutura matemática de grafo, chamada nesse contexto de grafo de construção, como já mencionado na referida Seção. Observa-se a partir disso e da expressão da representação de feromônio na Tabela 5.1 a partir da quantidade de dimensões utilizadas para se armazenar, consultar e depositar seus valores, que tal quantidade de dimensões é referente às necessidades das informações que devem estar contidas no grafo de construção.

Assim, tomando-se como exemplo o caso de código 1CC a partir da mesma Tabela 5.1, este possui uma dimensão que representa os valores de feromônio que estão relacionados à cada componente (cada vértice do grafo de construção). Dessa forma, o gráfico de construção está representado por tal vetor, sem ser necessário ter a representação de informações referente às conexões entre os componentes (arestas do grafo), pois tais informações não serão utilizadas no algoritmo ACO da variação em questão. Outros exemplos relevantes são 2SS, 2ST e 2TT de duas dimensões cada, nos quais se faz necessário que o feromônio seja armazenado nas conexões entre todos os componentes (vértices) – depósito de feromônio nas arestas do clique do grafo composto pelos referidos vértices – para tornar possível a consulta, o depósito e a representação dos valores nos referidos relacionamentos como pedem algoritmos ACO com tais configurações, porém, vale lembrar que embora com quantidades de dimensões equivalentes, os algoritmos têm comportamento de execução diferentes por conta das diferentes consultas e depósitos de feromônio.

Tendo essas duas informações como plano de fundo – as dimensões e o grafo de construção – sabe-se que elas expressam de forma genérica e teórica a representação de feromônio. Mas para que possam ser utilizadas de forma real em um sistema de computação, estas devem ser mapeadas em alguma estrutura de dados manipulável pelo computador. Nesse sentido, a Tabela 5.4 mostra um possível mapeamento.

*Tabela 5.4. Mapeamento de estruturas matemáticas de **representação de feromônio** em estruturas de dados computacionais*

Dimen-sões	**Modelagem matemática – *Grafo de construção***	***Exemplos* de estruturas de dados**	**Estruturas de dados *utilizadas* na implementação dos experimentos**
1		• Vetor • Lista • Qualquer estrutura que atenda aos requisitos de representar o grafo de construção em *uma dimensão* de forma a suprir as necessidades do algoritmo em questão	 Vetor (VET)
2		• Matriz de adjacência • Matriz de incidência • Lista de adjacência • Estrutura de conjunto • Qualquer estrutura que atenda aos requisitos de representar o grafo de construção em *duas dimensões* de forma a suprir as necessidades do algoritmo em questão	 Matriz de adjacência (MAT)

A mesma Tabela 5.4 contém as diferentes formas de representação de feromônio do grafo de construção que necessita a configuração do algoritmo ACO que está sendo utilizada.

A estrutura de dados utilizada para representação de feromônio (grafo de construção) define por si só a complexidade de espaço de memória e tem juntamente com as regras de consulta e depósito de feromônio importância capital na complexidade de tempo de execução do algoritmo ACO.

5.1.2 Consulta de Feromônio (Construção da Solução)

A informação armazenada na estrutura que representa o feromônio é utilizada para guiar as formigas no momento em que estas procuram novos componentes para adicioná-los em suas soluções. Essa orientação acontece por meio da regra de decisão – probabilística –, que na grande maioria dos algoritmos ACO utiliza-se dos referidos valores de feromônio depositados e de informações heurísticas. A partir das formas gerais das regras de decisão de cada algoritmo ACO referido no presente verifica-se que:

- τ é utilizado para representar valores de feromônio; e
- η é utilizado para representar informações heurísticas.

Para definir a que componente ou par de componentes se refere uma variável τ é necessário se definir o tipo de consulta de feromônio, que possui as variações CMP, SEQ e TOD. Utilizando tal característica como CMP, define-se um índice j para indicar a qual

componente refere-se a informação heurística em questão. Tem-se de forma bastante simples pela Equação 32:

$$\tau_j \tag{32}$$

Assim, utilizando consulta por CMP, realiza-se a consulta do valor de feromônio τ_j do componente candidato j do conjunto de componentes candidatos (sujeitos à escolha no momento).

No caso de se utilizar o tipo da consulta de feromônio como SEQ, define-se dois índices, i e j, para representar o par de componentes a que se refere o valor de feromônio como na Equação 33:

$$\tau_{ij} \tag{33}$$

Nas situações em que o algoritmo realiza uma consulta de feromônio por SEQ, convenciona-se que o primeiro índice, o i, refere-se ao componente dito atual, o que representa onde a formiga se encontra no momento da consulta. O segundo índice, o j, refere-se ao componente candidato da vizinhança (conjunto de componentes sujeitos à escolha no momento) cujo par formado com o componente atual está sendo consultado no momento. A princípio, esta consulta objetiva descobrir o valor de feromônio τ_{ij} do par de componentes (i, j).

Com o tipo da consulta de feromônio como TOD deve-se ter em mente que o valor do feromônio consultado refere-se – a princípio (pois poderia-se fazer a média, por exemplo) – à soma dos valores de feromônio de todos os pares de componentes existentes entre cada componente da solução parcial s sendo construída pela formiga k e o componente candidato j da vizinhança (conjunto de componentes sujeitos à escolha no momento) sendo analisado. Dessa forma, teria-se pela Equação 34:

$$\tau_{s_k j} = \sum_{i \in s_k} \tau_{ij} \tag{34}$$

Assim, com TOD a regra para consulta de feromônio se torna mais complexa que as outras formas do ponto de vista da complexidade de tempo de execução. Com o objetivo de amenizar tal complexidade, o trabalho de Alaya *et al.* (2004) faz o cálculo de maneira incremental da seguinte forma:

- cria-se uma estrutura extra na memória para cada formiga *k* chamada de T_{kj} com $j = 1 \ldots |C|$ para armazenar os valores das parcelas do somatório $\tau_{s_k j}$;
- tal estrutura pode ser implementada como, por exemplo, um vetor;
- essa estrutura não é o feromônio geral do algoritmo;
- inicializa-se todos os elementos da estrutura com 0;

- quando um componente j é escolhido para passar a fazer parte da solução, isto é, passar de $N(s_k)$ para o próprio s_k, o valor do somatório $\tau_{s_k j}$ deve ser recalculado para ser utilizado na próxima iteração para inclusão do próximo componente. Para que esse recálculo não tenha de ser refeito completamente, atualiza-se o os valores de T_{kj} da forma apresentada na Equação 35 (considerando l como o índice dos componentes do conjunto de componentes candidatos):

$$T_{kl} = T_{kl} + \tau_{jl}, \ \forall l \in N(s_k) \tag{35}$$

- assim, na próxima iteração, ao invés de se recalcular $\tau_{S_k j}$ para cada componente j do conjunto de componentes candidatos que se deseje consultar, é necessário apenas consultar o valor armazenado em T_{kj};

Uma característica relevante para os três tipos de consulta de feromônio – destaca-se especialmente o tipo TOD – é que a escolha do primeiro componente é feita de forma aleatória. Dado que o valor de feromônio antes da primeira iteração é 0, o algoritmo está preparado para escolher um componente aleatoriamente caso o valor de feromônio considerado seja 0.

A consulta de feromônio ainda pode ser caracterizada como estática ou dinâmica.

- estática (E): indica que o valor resultante da consulta do feromônio sob tal tipo mantém o seu valor – para qualquer das formigas consultando qualquer componente ou par de componentes – dentro de uma mesma iteração, sendo o caso das consultas:
    - CMP: cujos valores consultados são diretamente os feromônios dos componentes, que só se alteram na atualização de feromônio ao fim da construção da solução e, dessa forma, se torna uma alternativa com a menor complexidade computacional neste quesito;
    - SEQ: situação análoga à anterior, os valores consultados são os valores de feromônios que estão nos relacionamentos entre dois componentes, que só se alteram na atualização do feromônio ao fim da construção da solução, consistindo também de complexidade computacional próxima do tipo anterior;
- dinâmica (D): denota que o valor resultante da consulta do feromônio é variável – leia-se: necessita de algum cálculo – dentro de uma mesma iteração, como no caso de tal consulta ser dependente da solução parcial sendo construída. Tal caso e tipo de consulta de feromônio que se encaixa na categoria é:
    - TOD: neste caso o resultado da consulta de feromônio é resultante da soma de todos os valores de feromônio dos pares formados entre o componente candidato e todos aqueles componentes que estão na solução parcial sendo construída, necessitando que a cada consulta haja um calculo inerente, como

pode ser visto na parte dessa seção relativa a tal tipo de consulta de feromônio. Do ponto de vista da complexidade computacional esta é a mais elevada.

A classificação do tipo de consulta de feromônio entre estática e dinâmica é importante do ponto de vista da implementação, tendo esta importância se tornado especialmente útil se combinado com a mesma análise sobre a informação heurística de forma que em alguns casos com consulta de feromônio e de informação heurística estática, não é necessário fazer cálculos em relação a elas dentro da mesma iteração. Tal observação torna explícito os pontos positivos dos algoritmos mais simples do ponto de vista da complexidade computacional.

5.1.3 Depósito (Atualização) de Feromônio

Para que o algoritmo ACO possa funcionar corretamente, as soluções consideradas de boa qualidade pelo sistema devem ser reforçadas. Esse processo é feito na atualização de feromônio que contém a evaporação, depósito e atualização de limites, quando for o caso. Porém a característica mais notória pode ser dada pelo depósito de feromônio.

O valor a ser depositado nos resíduos de feromônio geralmente são definidos com base na qualidade da solução encontrada pela formiga em questão, de forma que quanto melhor a qualidade da solução, maior o valor do feromônio a ser depositado nos elementos que compõem tal solução. A formulação de como essa definição é feita varia entre os algoritmo ACO. Vale notar ainda que existe a possibilidade de se definir o valor do feromônio a ser depositado nos resíduos de forma a não usar diretamente a qualidade da solução, como pode ser visto no caso do *Simple-ACO*.

Com relação ao que é referido como o tipo de depósito de feromônio, este trabalho utiliza – a princípio – as três formas distintas já apresentadas:

- CMP: o feromônio a ser alterado é dado por τ_j, $\forall j \in s_k$ e neste caso, o feromônio é depositado diretamente nos componentes que fazem parte da solução. Note-se que a ordem em que os componentes estão dispostos na solução não têm influência no resultado do depósito. O feromônio depositado indica a desejabilidade de escolha de um componente individualmente;
- SEQ: quando o tipo de depósito de feromônio é setado dessa forma, deve-se depositar feromônio na conexão entre pares de componentes da solução na ordem em que se apresentam nesta, começando pelo primeiro componente. Assim, considerando-se f como o índice das colunas j na solução s_k, tem-se formulação $\tau_{j_f j_{f+1}}$, $\forall j \in (s_k \setminus j_{|s_k|})$. Nesta situação a ordem em que os componentes se encontram é importante, pois o feromônio será depositado nos pares formados pelo primeiro

componente com o segundo, no par do segundo com o terceiro, e assim sucessivamente até que todos os componentes da solução sejam contemplados. Reforça-se assim a desejabilidade de se escolher uma determinada seqüência de componentes;

- TOD: dado por $\tau_{i,j}$, $\forall i,j \in s_k$, o feromônio é depositado em todos os pares formados entre os componentes existentes na solução. Assim, a ordem em que os componentes se encontram na solução não causa impacto sobre o resultado do depósito de feromônio. Esse tipo de depósito de feromônio reforça a desejabilidade de se escolher os componentes da solução em questão de uma só vez.

Para exemplificar, considera-se um possível problema representado pelo grafo de construção contido na Figura 5.1, supondo que a seqüência de componentes $s = \langle 1,3,4 \rangle$ seja uma solução para tal problema. Tal solução está representada utilizando-se de uma seqüência pelo fato de se estar trabalhando com informações teóricas que ainda não foram convertidas em um problema real. O PCV, por exemplo, se encaixaria de forma adequada ao conceito de seqüência, mas por outro lado, o PCC deve utilizar uma abordagem menos rígida, no caso, a seqüência seria convertida em um conjunto com os componentes sendo seus elementos. A Figura 5.2 ilustra o referido grafo de construção após uma formiga construir a citada solução s por meio dos movimentos indicados nas setas.

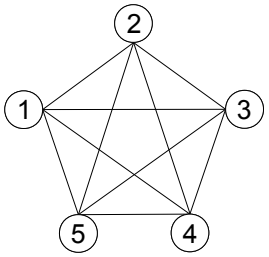


Figura 5.1. Conjunto de colunas mapeado em grafo de construção

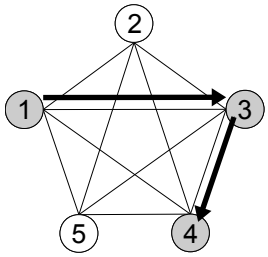


Figura 5.2. Movimentos de uma formiga no grafo de construção

Nota-se que o grafo e as características da solução são válidos para qualquer uma das representações, consultas e depósitos de feromônio citadas. As Figuras 5.3, 5.4 e 5.5 ilustram respectivamente para os tipos de depósito feromônio CMP, SEQ e TOD uma visualização de como fica disposto o feromônio depositado no grafo, o que é denotado por linhas pontilhadas.

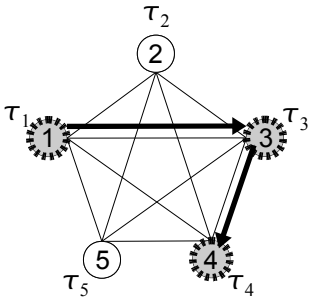


Figura 5.3. Depósito de feromônio por CMP

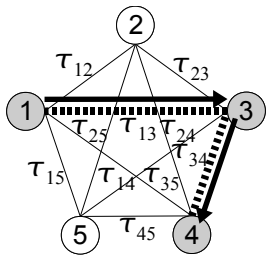


Figura 5.4. Depósito de feromônio por SEQ

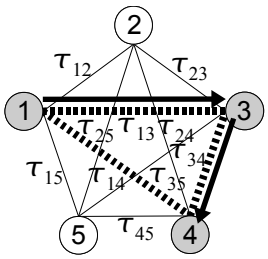


Figura 5.5. Depósito de feromônio por TOD

5.1.4 Estratégias de Manipulação de Feromônio para o PCC

O PCC é defino e exemplificado na Seção 2.1 e o grafo de construção – para o qual deve ser mapeado um problema ao qual um algoritmo ACO é aplicado – é apresentado na seção 3.3. Realizando tal mapeamento para PCC, tem-se que:

- as colunas do PCC são traduzidas nos componentes do grafo de construção;
- encontrar uma solução para o PCC consiste em selecionar um conjunto de colunas que satisfaça as restrições, tem-se que – pelo fato de tal seleção ser um conjunto – a ordem que as colunas estão dispostas não é importante.

Com base nisso e nas configurações 1CC, 2SS, 2ST e 2TT da Tabela 5.1, afirmadas com certeza de implementação e significados válidos, tem-se as considerações sobre as características de feromônio sobre:

- 1CC: esta configuração se adequa de forma bastante razoável ao PCC fazendo com que cada coluna tenha sua probabilidade de escolha expressa por meio do feromônio feita de forma individual. É a abordagem mais imediata e utilizada por vários trabalhos;
- 2SS: indica que a consulta de feromônio seja feita pela seqüência de pares e o depósito de feromônio realiza o reforço da desejabilidade de se escolher tal seqüência ou parte dela. Tal configuração não se mostra – a princípio – adequada ao PCC justamente por valorizar a seqüência dos elementos, principalmente no que tange os efeitos do depósito de feromônio. Porém como se está tratando de algoritmos heurísticos, não se pode afirmar que não dará resultados aproveitáveis, no entanto não é objetivo deste trabalho realizar tal análise. Alguns trabalhos se valem dessa abordagem para outros problemas;
- 2ST: nesse caso, existe a suspeita de que a consulta pela seqüência de pares poderia resultar em algum tipo de "memória indireta acumulada", dado que o depósito de feromônio ocorre entre todos os pares de uma solução encontrada. Assim a chance de escolha é reforçada entre todos os pares de uma mesma solução e a consulta é

realizada pelos pares do último componente adicionado à solução com os componentes candidatos. Assim, o raciocínio utilizado é: chegando-se a tal componente, os feromônios de suas ligações poderiam indicar qual seria um próximo desejável sem que seja necessário recorrer ao feromônio de todos aqueles que já fazem parte da solução. Uma questão a ser ponderada é o ganho na complexidade computacional em relação à consulta de feromônio (se comparada à configuração a seguir);

- 2TT: é a situação em que a consulta de feromônio para um componente candidato leva em consideração o feromônio contido em todos os pares entre os componentes contidos na solução parcial e o componente candidato em questão, fazendo com que todos os componentes da solução parcial construída até o momento influenciem o próximo componente a ser escolhido. O depósito de feromônio ocorre em todos os pares formados entre todos os componentes da solução encontrada, fortalecendo assim a chance de se escolher os componentes da solução juntos. Tal abordagem é a que fornece – intuitivamente – o maior poder teórico na construção de soluções de boa qualidade, porém é o que possui maior complexidade computacional.

Com base nessas características, as variações selecionadas para serem utilizadas em experimentos – no Capítulo 6 – para o PCC são apresentadas na Tabela 5.5, onde pode-se verificar as abreviaturas que serão utilizadas para se referir às suas respectivas características.

Tabela 5.5. Estratégias de manipulação de feromônio para o PCC

Código equivalente	**Feromônio**			
	Representação	**Consulta**		**Depósito**
		Tipo	**Estático (E) / Dinâmico (D)**	
1CC	VET	CMP	(E)	CMP
2ST	MAT	SEQ	(E)	TOD
2TT	MAT	TOD	(D)	TOD

5.1.5 Considerações Sobre Manipulação de Feromônio

De forma geral, tem-se ainda aspectos interessantes que podem ser analisados, como a comparação com trabalhos semelhantes, investigações do aspecto dimensões, idéias de implementação e questões empíricas referentes à experimentos que possam fornecer análises inerentemente realísticas.

Outros trabalhos promovem definições, discussões e experimentos nas variações das características de manipulação de feromônio. Em Alaya *et al.* (2004) realiza-se estudo com base no PMM, existindo em tal trabalho indicações para trabalhos de terceiros – Leguizamon e Michalewicz (1999) e Fidanova (2002) – que abordam configurações de feromônio semelhante à 1CC e 2SS – respectivamente –, enquanto que o trabalho principal citado

compara tais abordagens com sua versão de 2TT. Já em Solnon e Bridge (2006) existe a definição de um *framework* geral para utilização de algoritmos ACO para problemas de seleção de subconjunto – como é o caso do PCC – parametrizando as variações da chamada estratégia de feromônio em *Vertex,* correspondente à 1CC, e *Clique,* que corresponde à 2TT. Diante desses estudo, não foi encontrado configuração correspondente à 2ST, que o presente trabalho contempla e reporta resultados de experimentos no Capítulo 6.

No que tange o aspecto dimensões da representação de feromônio verifica-se que existe ainda formas de desdobramento que podem ser assunto de investigação. Lançando-se a pergunta: e se for utilizado três ou mais dimensões para a representação do feromônio – e das conexões – entre os componentes? Como resposta, basicamente se pode dizer que o feromônio seria depositado em tuplas de três componentes, mas muitas questões ainda poderiam ser investigadas como qual seria a implicação na qualidade da solução, diferenças na complexidade computacional – provavelmente maior, mas seria necessário estudos para verificar se compensaria uma eventual melhoria na qualidade da solução –, à quais problemas ou tipos de problema a técnica beneficiaria ou prejudicaria, se poderia ser dispensada uma provável busca local além de muitos outros aspectos que poderiam trazer ao conhecimento questões relevantes. Tais questões foram lançadas a partir dos resultados obtidos em respostas às questões que motivaram a origem deste trabalho e pertencem à um escopo de trabalhos futuros.

Na tentativa de se obter estruturas de dados e algoritmos mais eficazes e eficientes, apresenta-se a idéia de implementação que pode ser vista partindo-se da estrutura de dados para representação de grafos chamada estrutura de conjunto, conforme possibilidade contida na Tabela 5.4 da seção 5.1.1, que poderia representar os pares de componentes – tais pares são arestas em um contexto de grafo – em uma forma de lista ligada na memória do computador, onde cada elemento da lista contém os números dos componentes a que tal par se refere. Assim, a partir de tal estrutura e do fato de que há variações nas configurações de feromônio que se utilizam da manipulação de dados – leia-se feromônio principalmente – em pares de componentes, lança-se a idéia de que a investigação de tal estrutura poderia agregar algum aspecto relevante à maneira de se implementar essa característica de algoritmos ACO. Talvez não da forma clássica, ou talvez incrementaria algo na idéia tridimensional do parágrafo anterior. Porém, tal investigação faz parte de um cenário de trabalhos futuros, assim como o referido parágrafo anterior.

Por fim, diante do exposto tem-se base teórica para composição e análise de experimentos de algoritmos ACO que venham a variar características de sua manipulação de feromônio, conforme pode ser verificado no capítulo 6. É possível que se avalie tais

experimentos de forma indireta, por meio de sua influência nos algoritmos ACO e de forma direta, pela observação das variações de feromônio em suas respectivas estruturas.

5.2 Algoritmos Heurísticos Baseados em ACO Aplicados ao PCC

De posse da proposta das variações da manipulação de feromônio, define-se nesta Seção tais variações aos algoritmos MMAS_L, AS_RC_2 e AAS_MC.

5.2.1 MMAS_L Aplicado ao PCC

As características **grafo de construção**, **restrições**, **resíduos de feromônio**, **informação heurística**, **informação de escolha, algoritmo heurístico** e **busca local** são tomadas por aquelas equivalentes a cada uma destas apresentadas para o algoritmo MMAS_L para PCC na seção 4.3.

Construção da Solução

Baseando-se na Subseção 4.3.2 é necessário aqui apenas se redefinir a regra de decisão a ser utilizada. Assim, para se obter a probabilidade de escolha de cada componente no momento de aplicar a regra de decisão utiliza-se da probabilidade p a seguir, classificadas de acordo com o tipo de consulta de feromônio (e, por extensão, de tipo de consulta de informação heurística e informação de escolha) conforme segue.

- CMP: consultando o feromônio em componentes – equivalente ao caso normal – têm-se que cada componente j possui feromônio τ_j e informação heurística η_j. Assim, a probabilidade p de escolha do componente j pela formiga k é dada pela Equação 17;
- SEQ: consultando o feromônio por seqüência de pares de componentes tem-se que cada par (i, j) de componentes possui feromônio denotado por τ_{ij} e a informação heurística é dada por η_{ij}. Dessa forma, a probabilidade p de escolha do componente j pela formiga k a partir do componente i é dada pela Equação 36:

$$p_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{h \in N(s_k)} \tau_{ih}^{\alpha} \cdot \eta_{ih}^{\beta}} & se\ j \in N(s_k) \\ 0 & se\ j \notin N(s_k) \end{cases} \tag{36}$$

- TOD ou TDN: nesse caso, o feromônio é consultado de forma que cada componente existente na solução parcial s sendo construída pela formiga k tenha uma influência direta (não apenas indireta, como na SEQ) na probabilidade p de escolha do próximo componente j. O fator de feromônio nesse caso é dado por $\tau_{s_k j}$, cuja definição é

dada pela Equação 34. Para a informação heurística são dadas duas formas de utilização neste tipo de consulta de feromônio, sendo a maneira dita híbrida (abreviada por TOD) e a maneira dita normal (abreviada por TDN), conforme segue:

- TOD: a informação heurística de um componente j, partindo-se do componente i e com a solução s parcialmente construída pela formiga k, é dada por η_{ij}. Assim tem-se a Equação 37:

$$p_{s_k ij}^{k} = \begin{cases} \dfrac{\tau_{s_k j}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{h \in N(s_k)} \tau_{s_k h}^{\alpha} \cdot \eta_{ih}^{\beta}} & se\ j \in N(s_k) \\ 0 & se\ j \notin N(s_k) \end{cases} \tag{37}$$

- TDN: nesse caso a informação heurística de um componente j, com a solução s parcialmente construída pela formiga k, é dada por $\eta_{s_k j}$ com definição pela Equação 38:

$$\eta_{s_k j} = \sum_{i \in s_k} \eta_{ij} \tag{38}$$

Dessa forma tem-se a Equação 39:

$$p_{s_k j}^{k} = \begin{cases} \dfrac{\tau_{s_k j}^{\alpha} \cdot \eta_{s_k j}^{\beta}}{\sum_{h \in N(s_k)} \tau_{s_k h}^{\alpha} \cdot \eta_{s_k h}^{\beta}} & se\ j \in N(s_k) \\ 0 & se\ j \notin N(s_k) \end{cases} \tag{39}$$

Os valores obtidos através de parâmetros α e β indicam a importância do fator feromônio e da informação heurística respectivamente. Têm-se também que $N(s_k)$ é a vizinhança considerada de s_k, esta que é a solução parcial da formiga k sendo construída até a referida iteração t. Para o PCC, geralmente tem-se que $N(s_k)$ contém todos os componentes em C mas que não fazem parte de s_k.

Atualização de Feromônio

Com a representação de feromônio em uma dimensão – equivalente ao caso normal – a evaporação de feromônio acontece da forma exposta na Equação 18.

No caso de duas dimensões, a evaporação de feromônio é dada pela Equação 40:

$$\tau_{ij} = (1-\rho) \cdot \tau_{ij}\ ,\ \forall\, i, j \in C \tag{40}$$

Com o tipo de depósito de feromônio como CMP – equivalente ao caso normal –, o feromônio a ser atualizado é dado por τ_j, $\forall\, j \in s_{best}$, de acordo com a regra de atualização da Equação 19 que se encontra na página 70.

Quando o tipo de depósito de feromônio é setado como SEQ, tem-se formulação $\tau_{j_f j_{f+1}}$, $\forall\, j \in (s_{best} \setminus j_{|s_{best}|})$. Para atualização tem-se a Equação 41:

$$\tau_{j_f j_{f+1}} = \tau_{j_f j_{f+1}} + \Delta\tau \ , \ \forall\, j \in s_{best} \tag{41}$$

O tipo de feromônio como TOD, dado por τ_{ij}, $\forall\, i, j \in s_{best}$ tem a regra de atualização pela Equação 42:

$$\tau_{ij} = \tau_{ij} + \Delta\tau \ , \ \forall\, j \in s_{best} \tag{42}$$

Com $\Delta\tau$ e *best* já definidos por meio da Subseção 4.3.3 na página 69.

Considerações

Em comparação aos limites de feromônio normais, aqui será necessário impor os limites ao feromônio depositado nos pares de componentes quando a representação de feromônio for SEQ e TOD.

As operações em inicialização de feromônio e reinicialização de feromônio também devem se adequar a SEQ e TOD.

5.2.2 AS_RC_2 Aplicado ao PCC

As características **grafo de construção**, **restrições**, **resíduos de feromônio**, **informação heurística**, **informação de escolha**, **algoritmo heurístico** e **busca local** são tomadas por aquelas equivalentes a cada uma destas apresentadas para o algoritmo AS_RC_2 para PCC na Seção 4.4.

Construção da Solução

A construção de soluções é feita de acordo com o apresentado na Subseção 4.4.1, com a construção de uma solução por uma formiga sendo feita da mesma forma como ilustrada na Figura 4.10, com a diferença de que a aplicação de uma busca local pode ser feita, conforme é ilustrado por meio da linha 8 da Figura 5.6.

```
 FormigaPccAsRc2::construirSolucao(){
1   S ← Ø ;
2   U ← M ;  // U é o conjunto de linhas não cobertas.
3   w_i ← 0 , ∀ i ∈ M ;  // w_i é a qt.cols.em S que cobrem a linha i , i ∈ S .
4   while( not verificarSeSolucaoEstaCompleta() ){
5      aplicarPassoDeConstrucao();  // Chama aplicarRegraDeDecisao() internamente.
6   };
7   eliminarColunasRedundantes();
8   aplicarBuscaLocal();
 }
```

Figura 5.6. Construir solução de formiga de AS_RC_2 com aplicação de busca local

Também modifica-se a segunda etapa da regra de decisão apresentada na Subseção 4.4.1 para se obter então a regra de decisão completa para todas as situações desejadas, assim, tem-se que o próximo componente j é dado de acordo com as regras a seguir, classificadas

de acordo com o tipo de consulta de feromônio (e, por extensão, de tipo de consulta de informação heurística e informação de escolha).

- CMP (equivalente ao caso normal): com o tipo de consulta de feromônio (e informação heurística) por componentes, tem-se que cada componente h possui feromônio τ_h e informação heurística η_h, de forma que a escolha do componente j pela formiga k é dado pela Equação 28;
- SEQ: consultando o feromônio (e informação heurística) por seqüência de pares, têm-se que cada par de componentes (i, j) possui feromônio τ_{ij} e informação heurística η_{ij}, de maneira que a escolha da coluna (componente) j pela formiga k é definido pela Equação 43:

$$j = argmax_{h \in N(e,i,s_k)} \{\tau_{ih}^{\alpha} \cdot \eta_{ih}^{\beta}\} \quad (43)$$

- TOD ou TDN: nesta situação, o feromônio é consultado de forma que cada componente existente na solução parcial s sendo construída pela formiga k tem uma influência direta (não apenas indireta, como em SEQ) no peso de escolha do próximo componente j. O fator de feromônio nesse caso é dado por $\tau_{s_k j}$, cuja definição é dada pela Equação 34. Para a informação heurística são dadas duas formas de utilização neste tipo de consulta de feromônio, sendo a maneira dita híbrida (abreviada por TOD) e a maneira dita normal (abreviada por TDN), conforme segue:
  - TOD: a informação heurística de um componente j, partindo-se do componente i e com a solução s parcialmente construída pela formiga k, é dada por η_{ij}. Assim tem-se a Equação 44:

$$j = argmax_{h \in N(e,i,s_k)} \{\tau_{s_k h}^{\alpha} \cdot \eta_{ih}^{\beta}\} \quad (44)$$

  - TDN: nesse caso a informação heurística de um componente j, com a solução s parcialmente construída pela formiga k, é dada por $\eta_{s_k j}$ definido pela Equação 38, constituindo a regra de decisão da forma dada na Equação 45:

$$j = argmax_{h \in N(e,s_k)} \{\tau_{s_k h}^{\alpha} \cdot \eta_{s_k h}^{\beta}\} \quad (45)$$

Os valores informados através de parâmetros α e β indicam a importância do fator feromônio e da informação heurística respectivamente. O parâmetro ρ indica a taxa de evaporação de feromônio. Têm-se também que $N(e, s_k)$ é dado pela Equação 27 da Subseção 4.4.1.

Atualização de Feromônio

Com a representação de feromônio em uma dimensão – equivalente ao caso normal – a evaporação de feromônio acontece da forma exposta na Equação 29.

No caso de duas dimensões, tem-se a forma dada na Equação 46:

$$\tau_{ij}=(1-\rho)\cdot\tau_{ij},\ \forall i,j\in C \tag{46}$$

O parâmetro ρ é a taxa de evaporação.

Com o tipo de depósito de feromônio como CMP, o feromônio a ser alterado é dado por $\tau_j,\ \forall j\in s_k$. Assim, tem-se a regra de atualização dada pela Equação 30.

Quando o tipo de depósito de feromônio é setado como SEQ, tem-se formulação $\tau_{j_f j_{f+1}},\ \forall j\in(s_{best}\setminus j_{|s_{best}|})$. Para atualização tem-se a Equação 47:

$$\tau_{j_f j_{f+1}}=\tau_{j_f j_{f+1}}+\Delta\tau,\ \forall j\in s_{best} \tag{47}$$

Com o tipo de feromônio como TOD, este é dado por $\tau_{ij},\ \forall i,j\in s_{best}$ e a regra de atualização é denotada pela Equação 48:

$$\tau_{ij}=\tau_{ij}+\Delta\tau,\ \forall j\in s_{best} \tag{48}$$

Com $\Delta\tau$ sendo mostrado na Equação 49:

$$\Delta\tau=\left(\frac{f(s_{bs})}{f(s_{best})}\right)^y \tag{49}$$

Onde y é uma constante utilizada para alterar a importância do feromônio a ser depositado nos componentes da solução e s_{best} indica solução da formiga com índice $best$, índice este que por sua vez pode denotar:

- *iteration best* (ib): formiga com melhor solução da iteração;
- *best so far* (bs): formiga com melhor solução desde o início da execução do algoritmo heurístico. Em um terminologia voltada aos experimentos do Capítulo 6 diz-se a melhor solução da tentativa.

Uma questão importante é a análise do valor utilizado para compor o depósito de feromônio. Cabe ressaltar que qualquer valor de bs referenciado nesse contexto inclui os dados da iteração em questão, de forma que se a bs for encontrada na iteração atual esta já é utilizada de forma atualizada, o que fará com que bs seja igual à ib na referida iteração atual em tal situação.

Pode-se verificar pela Equação 49 que o valor de depósito de feromônio é em relação à bs (ao invés de à 1, como no caso do MMAS_L aplicado ao PCC). O que pode ser visto pela formalização quando $best=ib$, tem-se a Equação 50:

$$\Delta\tau=\left(\frac{f(s_{bs})}{f(s_{ib})}\right)^y \tag{50}$$

Apenas recapitulando, quanto menor o valor de $f(s)$ melhor, pois se está tentando minimizar a função f. Dessa forma, se a iteração atual encontrar e atribuir à ib uma solução que não seja bs, ocorrerá que o valor de ib será maior (pior) que o valor de bs, como demonstra a Equação 51:

$$f(s_{ib})>f(s_{bs})\ \Rightarrow\ f(s_{bs})<f(s_{ib}) \tag{51}$$

Situação esta que fará com que $\frac{f(s_{bs})}{f(s_{ib})}<1$.

Caso a iteração atual encontre a solução bs pela primeira vez ou de forma repetida, tem-se que $\frac{f(s_{bs})}{f(s_{ib})}=1$, pois $f(s_{ib})=f(s_{sb})$, $\forall(f(s_{bs})\neq 0)\wedge(f(s_{ib})\neq 0)$.

É interessante notar que nesse esquema de atualização – com o uso do ib incluindo o bs – o valor de ib nunca será menor (melhor) que o valor de bs, o que acarretaria em $\frac{f(s_{bs})}{f(s_{ib})}>1$. Seria interessante verificar se tal possibilidade pode melhorar o desempenho do algoritmo no sentido da velocidade de convergência, dado que se a bs for encontrada na iteração atual, tal melhoria em relação ao bs anterior resultaria em um maior depósito de feromônio. Um ponto negativo poderia ser uma maior facilidade em um comportamento de estagnação em soluções ótimas locais.

E quando $best=bs$, tem-se o conteúdo da Equação 52:

$$\Delta\tau=\left(\frac{f(s_{bs})}{f(s_{bs})}\right)^{y} \tag{52}$$

Nesta situação quando $best=bs$, é notório que $\frac{f(s_{bs})}{f(s_{bs})}=1$, $\forall f(s_{bs})\neq 0$, fazendo com que o feromônio da solução bs seja reforçado.

Depois disso, o valor de y vem modificar o valor do feromônio a ser depositado, de forma que pode-se verificar pela Tabela 5.6 como é a influência de tal valor para situações válidas de acordo com o já discutido a respeito de $f(.)$. Os valores que efetivamente são utilizados no depósito de feromônio são os que estão em itálico na referida Tabela.

Tabela 5.6. Variação de best e de y para situações válidas com bs sendo atualizado antes do depósito de feromônio

Pode acontecer quando $best=$		Valor exemplo para $f(s_{bs})$	Relação de $f(.)$	Valor exemplo para $f(s_{best})$	$\frac{1}{f(s_{best})}$	$\frac{f(s_{bs})}{f(s_{best})}$ chamado x	x em relação à x^y	$\left(\frac{f(s_{bs})}{f(s_{best})}\right)^y$ chamado x^y	Valor exemplo para y	Intervalo de y
bs	ib									
NÃO, pois $best \neq bs$	**SIM,** $best=ib$ e encontrou uma solução pior que s_{bs}	10	$f(s_{bs})<f(s_{best})$	20	0,05	**0,00**	$x<x^y$	4,00	-2,00	$y<0$
							$x<x^y$	2,00	-1,00	
							$x<x^y$	1,68	-0,75	
							$x<x^y$	1,41	-0,50	
							$x<x^y$	1,19	-0,25	
							$x<x^y$	1,00	0,00	$y=0$
							$x<x^y$	0,84	0,25	$0<y<1$
							$x<x^y$	0,71	0,50	
							$x<x^y$	0,59	0,75	
							$x=x^y$	***0,50***	1,00	$y=1$
							$x>x^y$	***0,25***	2,00	$y>1$
							$x>x^y$	***0,21***	2,25	
							$x>x^y$	***0,18***	2,50	
							$x>x^y$	***0,15***	2,75	
							$x>x^y$	***0,13***	3,00	
SIM, $best=bs$	**SIM@,** $best=ib$	10	$f(s_{bs})=f(s_{best})$	10	0,10	**1,00**	$x=x^y$	***1,00***	0,00	$y\in\mathbb{R}$

Com o significado de SIM@ da Tabela 5.6 dado por:

- no caso de bs sendo atualizado **antes** do depósito de feromônio tem-se que SIM@ significa: **SIM@,** $best=ib$ e encontrou uma solução melhor ou igual que s_{bs} anterior (se s_{ib} for melhor que s_{bs}, bs foi atualizado, se tornado igual a ib); ou
- caso esteja definido que bs é atualizado **depois** do depósito de feromônio – e para fazer com que a Tabela 5.6 seja válida para tal variação, podendo ser usada em complemento à Tabela 5.7 –, deve-se ter que SIM@ significa apenas: **SIM@**, $best=ib$ e encontrou uma solução igual a s_{bs} anterior.

Também é interessante notar a situação em que o bs é atualizado depois da aplicação do depósito de feromônio, abrindo assim o possível caminho que ilustra a Tabela 5.7.

*Tabela 5.7. Situações de best de y para situações hipotéticas (não utilizadas no presente trabalho) com bs sendo atualizado **depois** do depósito de feromônio*

Pode acontecer quando $best=$		Valor exemplo para $f(s_{bs})$	Relação de $f(.)$	Valor exemplo para $f(s_{best})$	$\frac{1}{f(s_{best})}$	$\frac{f(s_{bs})}{f(s_{best})}$ chamado x	x em relação à x^y	$\left(\frac{f(s_{bs})}{f(s_{best})}\right)^y$ chamado x^y	Valor exemplo para y	Intervalo de y
bs	*ib*									
NÃO	**SIM**	10	$f(s_{bs})>f(s_{best}$	5	0,20	**2,00**	$x>x^y$	0,25	-2,00	$y<0$
							$x>x^y$	0,50	-1,00	
							$x>x^y$	0,59	-0,75	
							$x>x^y$	0,71	-0,50	
							$x>x^y$	0,84	-0,25	
							$x>x^y$	1,00	0,00	$y=0$
							$x>x^y$	1,19	0,25	$0<y<1$
							$x>x^y$	1,41	0,50	
							$x>x^y$	1,68	0,75	
							$x=x^y$	***2,00***	1,00	$y=1$
							$x<x^y$	***4,00***	2,00	$y>1$
							$x<x^y$	***4,76***	2,25	
							$x<x^y$	***5,66***	2,50	
							$x<x^y$	***6,73***	2,75	
							$x<x^y$	***8,00***	3,00	

Vale notar ainda que a coluna intitulada $\frac{1}{f(s_{best})}$ existente nas Tabelas 5.6 e 5.7 tem por finalidade ilustrar uma comparação entre os valores de depósito de feromônio utilizados pelo AS_RC_2 e pelo MMAS_L (vide Equação 20).

5.2.3 AAS_MC Aplicado ao PCC

Com o objetivo de fazer com que o AS tenha capacidade de se adaptar automaticamente em relação aos aspectos de diversificação (exploração) e intensificação (*exploit*) das soluções construídas é proposto o algoritmo heurístico *Adaptive Ant System* (AAS_MC), que faz com que cada formiga tenha condição de alterar dinamicamente a cada iteração os valores de α e β que serão utilizados por ela na composição da informação de escolha a partir dos valores de feromônio e de informação heurística para decidir qual próximo componente deve ser adicionado á solução corrente. Tais valores são denotados por α_k e β_k, com k sendo a formiga à qual pertencem os adaptadores. O AAS_MC tem como um de seus propósitos evitar a estagnação, por conta da adaptação que realiza, bem como objetiva a diminuição dos parâmetros a serem definidos.

As características **grafo de construção**, **restrições**, **resíduos de feromônio**, **informação heurística**, **informação de escolha, algoritmo heurístico** e **busca local** são

tomadas por aquelas equivalentes a cada uma destas apresentadas para o algoritmo ACO para PCC genérico na seção 4.1.

Construção da Solução

A construção da solução no AAS_MC é dada pelo algoritmo da Figura 5.7.

```
 FormigaPccAasMc::construirSolucao(){
1   somaFatorFeromonioSolucao     = 0.0;
2   somaFatorInfHeuristicaSolucao = 0.0;

3   contagemFatorFeromonio     = 0;
4   contagemFatorInfHeuristica = 0;

5   while( not verificarSeSolucaoEstaCompleta() ){
6      aplicarPassoDeConstrucao();  // Chama aplicarRegraDeDecisao() internamente.
7   }

8   eliminarColunasRedundantes();
9   aplicarBuscaLocal();
10  atualizarAdaptadores();
 }
```

Figura 5.7. Construir solução de formiga de AAS_MC

Para o AAS_MC os adaptadores são modificados ao longo das iterações (do tempo) – conforme Figura 5.12 – e cada formiga possui seus adaptadores específicos (individualizados), ao contrário do MMAS_L por exemplo, onde os adaptadores são válidos para todas as formigas. Assim, comparando-se os adaptadores das duas referidas heurísticas tem-se a Tabela 5.8.

Tabela 5.8. Comparação de adaptadores de MMAS_L e AAS_MC

Adaptador	MMAS_L	AAS_MC
De feromônio	α	α_k , $k = 1 \ldots qtFormigas$
De informação heurística	β	β_k , $k = 1 \ldots qtFormigas$

Onde verifica-se que a quantidade existente de α_k e β_k é a mesma da quantidade de formigas.

Assim, o procedimento `aplicarRegraDeDecisao()` utiliza uma **regra de decisão** para decidir de forma probabilisticamente guiada qual novo componente deve ser escolhido para ser incluído na solução, assim como também é ilustrado na Figura 4.8 para o MMAS_L aplicado ao PCC. Deve ser ressaltado que, neste procedimento, para cálculo da informação de escolha a partir do feromônio e da informação heurística são utilizados α_k e o β_k individual de cada formiga k.

De acordo com a Seção 5.1 (página 77), o tipo de consulta de feromônio pode assumir valores dentre CMP, SEQ e TOD. Ainda tomando como base as probabilidade definidas para o MMAS_L, define-se as probabilidade para o AAS_MC como baseadas neste com a

diferença de que os valores de α e β são substituídos por α_k e β_k respectivamente. Assim, de acordo com a regra de feromônio tem-se:

- CMP: consultando o feromônio em componentes, cada componente j possui feromônio τ_j e informação heurística η_j. Com isso a probabilidade p de escolha do componente j pela formiga k é dada pela Equação 53:

$$p_j^k = \begin{cases} \dfrac{\tau_j^{\alpha_k} \cdot \eta_j^{\beta_k}}{\sum_{h \in N(s_k)} \tau_h^{\alpha_k} \cdot \eta_h^{\beta_k}} & se\ j \in N(s_k) \\ 0 & se\ j \notin N(s_k) \end{cases} \tag{53}$$

- SEQ: consultando o feromônio por seqüência de pares de componentes, tem-se que cada par (i, j) de componentes possui feromônio denotado por τ_{ij} e a informação heurística é dada por η_{ij}. Dessa forma, a probabilidade p de escolha do componente j pela formiga k a partir do componente i é dada pela Equação 54:

$$p_{ij}^k = \begin{cases} \dfrac{\tau_{ij}^{\alpha_k} \cdot \eta_{ij}^{\beta_k}}{\sum_{h \in N(s_k)} \tau_{ih}^{\alpha_k} \cdot \eta_{ih}^{\beta_k}} & se\ j \in N(s_k) \\ 0 & se\ j \notin N(s_k) \end{cases} \tag{54}$$

- TOD ou TDN: nesse caso, o feromônio é consultado de forma que cada componente existente na solução parcial s sendo construída pela formiga k tenha uma influência direta (não apenas indireta, como na SEQ) na probabilidade p de escolha do próximo componente j. O fator de feromônio nesse caso é dado por $\tau_{s_k j}$, cuja definição é dada pela Equação 34. Para a informação heurística são dadas duas formas de utilização neste tipo de consulta de feromônio, sendo a maneira dita híbrida (abreviada por TOD) e a maneira dita normal (abreviada por TDN), conforme segue:
  - TOD: a informação heurística de um componente j, partindo-se do componente i e com a solução s parcialmente construída pela formiga k, é dada por η_{ij}. Assim tem-se a Equação 55:

$$p_{s_k ij}^k = \begin{cases} \dfrac{\tau_{s_k j}^{\alpha_k} \cdot \eta_{ij}^{\beta_k}}{\sum_{h \in N(s_k)} \tau_{s_k h}^{\alpha_k} \cdot \eta_{ih}^{\beta_k}} & se\ j \in N(s_k) \\ 0 & se\ j \notin N(s_k) \end{cases} \tag{55}$$

  - TDN: nesse caso a informação heurística de um componente j, com a solução s parcialmente construída pela formiga k, é dada por $\eta_{s_k j}$ com definição conforme Equação 38 na página 90. Dessa forma tem-se a Equação 56:

$$p_{s_k j}^{k} = \begin{cases} \dfrac{\tau_{s_k j}^{\alpha_k} \cdot \eta_{s_k j}^{\beta_k}}{\sum_{h \in N(s_k)} \tau_{s_k h}^{\alpha_k} \cdot \eta_{s_k h}^{\beta_k}} & se\ j \in N(s_k) \\ 0 & se\ j \notin N(s_k) \end{cases} \quad (56)$$

Os valores obtidos através de parâmetros α e β e atribuídos para os α_k e β_k indicam a importância do fator feromônio e da informação heurística respectivamente. Têm-se também que $N(s_k)$ é a vizinhança considerada de s_k, esta que é a solução parcial da formiga k sendo construída até a referida iteração t. Para o PCC, geralmente tem-se que $N(s_k)$ contém todos os componentes em C mas que não fazem parte de s_k.

Atualização de Feromônio

Vide Subseção 4.1.9 confrontando com a Figura 5.9.

```
 AcoPccAasMc::atualizarResiduosFeromonio(){
1   evaporarFeromonio();
2   depositarFeromonio();
3   reInicializarFeromonio();
 }
```

Figura 5.8. Atualizar feromônio de AAS_MC

Adaptadores, Inicialização de Adaptadores e Atualização de Adaptadores

A inicialização dos adaptadores α_k e β_k além do γ_k é executada no momento da criação de cada formiga k utilizando de base os valores recebidos através dos parâmetros α, β e γ. Tal procedimento é ilustrado por `inicializarAdaptadores()` contido na Figura 5.9. Um cuidado especial deve ser tomado na parametrização dos referidos valores, pois tem-se que logo no início do algoritmo o valor de α_k e β_k a ser utilizado pela formiga já não é mais o mesmo que foi parametrizado, mas sim são valores agora com base no valor de γ (ou γ_k, que neste contexto é equivalente a γ).

```
 FormigaPccAasMc::inicializarAdaptadores(){
1   γ_k = γ ;
2   α_k = α·γ_k ;
3   β_k = β·(1−γ_k) ;
 }
```

Figura 5.9. Inicializar adaptadores de formiga de AAS_MC

Para poder realizar a mudança dos adaptadores, ao final do procedimento `aplicarRegraDeDecisao()` são realizadas operações para embasar a atualização dos adaptadores, que estão colocadas de duas possíveis maneiras em relação aos componentes que vão sendo adicionados à solução corrente em questão:

- Pelas somas dos fatores de feromônio e da soma dos fatores de informação heurística; ou

- Por meio da contagem da quantidade de vezes que os fator de feromônio é maior que o fator de informação heurística ou da contagem da quantidade de vezes que a informação heurística é maior que o fator de feromônio (se forem iguais, o incremento é computado ao fator de informação heurística).

  Tais maneiras são ilustradas pelas Figuras 5.10 e 5.11.

```
 FormigaPccAasMc::aplicarRegraDeDecisao(){
1   /* (...) Corpo do procedimento para decidir o próximo componente a ser
             adicionado na solução. */

2   somaFatorFeromonioSolucao     += fatorFeromonio__repet;
3   somaFatorInfHeuristicaSolucao += fatorInfHeuristicaDin__repet;

4   /* (...) */
 }
```

*Figura 5.10. Trecho de aplicar regra de decisão de formiga de AAS_MC indicando modo **por soma** para atualização dos adaptadores*

```
 FormigaPccAasMc::aplicarRegraDeDecisao(){
1   /* (...) Corpo do procedimento para decidir o próximo componente a ser
             adicionado na solução. */

2   if( fatorFeromonio__repet > fatorInfHeuristicaDin__repet ){
3      contagemFatorFeromonio++;
4   }
5   else{
6      contagemFatorInfHeuristica++;
7   }

8   /* (...) */
 }
```

*Figura 5.11. Trecho de aplicar regra de decisão de formiga de AAS_MC indicando modo **por contagem** para atualização dos adaptadores*

Após a construção por completo da solução da formiga k (dado pela Figura 5.7, página 97) poderá ter duas maneiras de utilização classificadas de acordo com o tipo de base utilizado, conforme segue.

- por soma tem-se as Equações 57 e 58:

$$somaFatorFeromonioSolucao = \sum_{\forall j \in s_k} \left(\tau_j^{\alpha_k} \right) \tag{57}$$

$$somaFatorInfHeuristicaSolucao = \sum_{\forall j \in s_k} \left(\eta_j^{\beta_k} \right) \tag{58}$$

- por contagem tem-se as Equações 59 e 60:

$$contagemFatorFeromonioSolucao = \left| \left\{ j \middle| \left(j \in s_k \right) \wedge \left(\left(\tau_j^{\alpha_k} \right) > \left(\eta_j^{\beta_k} \right) \right) \right\} \right| \tag{59}$$

O lado direito da Equação 59 pode ser lido como a quantidade de componentes j da solução s_k cujos valores de $\tau_j^{\alpha_k}$ são maiores que $\eta_j^{\beta_k}$.

$$contagemFatorInfHeuristica = \left| \left\{ j \middle| \left(j \in s_k \right) \wedge \left(\left(\tau_j^{\alpha_k} \right) \leqslant \left(\eta_j^{\beta_k} \right) \right) \right\} \right| \tag{60}$$

De forma parecida, o lado direito da Equação 60 pode ser lido como a quantidade de componentes j da solução s_k cujos valores de $\tau_j^{\alpha_k}$ são menores ou iguais a $\eta_j^{\beta_k}$. Assim, os referidos valores de somas ou contagens, servem de base para a atualização dos adaptadores conforme é ilustrado nas Figuras 5.12 e 5.13.

```
 FormigaPccAasMc::atualizarAdaptadores(){
1   if( f(s_k)>f(s*_bsa) ){ // Se a solução da formiga k atual é pior que a bsa...
2       DR=[f(s_k)−f(s*_bsa)] / f(s_k) ;  // DR: Desvio relativo.
3     if( somaFatorFeromonioSolucao > somaFatorInfHeuristicaSolucao ){
4         γ_k=γ_k−(γ_k*DR) ;
5     }
6     else{
7         γ_k=γ_k+[(1−γ_k)*DR] ;
8     }
9      α_k=α*γ_k ;
10     β_k=β*(1−γ_k) ;
11  }
 }
```

*Figura 5.12. Atualizar adaptadores **por soma** de formiga de AAS_MC*

A Figura 5.13 ilustra a parte dada pela condição do **if** interno que deve ser substituída na Figura 5.12 para que esta passe a funcionar de acordo com a contagem.

```
 // (...)
3     if( contagemFatorFeromonio > contagemFatorInfHeuristica ){
 // (...)
```

*Figura 5.13. Parte modificada de atualizar adaptadores por soma de formiga de AAS_MC de forma que funcione **por contagem** para o mesmo tipo de formiga*

Analisando o algoritmo da Figura 5.12 se observa que $f(s_k)$ representa a função objetivo (custo) da solução encontrada pela formiga k. A função objetivo dada por $f(s^*_{bsa})$ é referente a solução *best-so-far* "anterior", que é a melhor solução encontrada até a iteração anterior a atual (diferentemente do AS_RC_2), o que implica que se uma melhor solução foi encontrada na iteração atual, esta não é considerada em s^*_{bsa} e ocorreria que $f(s_k)<f(s^*_{bsa})$ onde constataria-se que uma melhor solução global foi encontrada na iteração atual.

Ainda segundo a Figura 5.12, a variável DR vai dizer se a solução s_k construída pela formiga k é igual ou se mudou, e se mudou denota o quanto a solução s_k melhorou ou piorou em relação à solução s^*_{bsa}. Se a solução for a mesma, ocorrerá que $DR=0$, o que fará com que os adaptadores γ_k, α_k e β_k permaneçam os mesmos, porém o cálculo é evitado nessa condição por conta do primeiro **if**, pois não necessitaria efetuar alterações. Outra situação em que os adaptadores não são modificados é quando a solução s_k é melhor que a

solução s^*_{bsa}, o que se ocorresse, faria com que $DR<0$. No caso em que a solução construída pela formiga em questão seja pior que a global, será válido $f(s_k)>f(s^*_{bsa})$ o que fará com que a condição do primeiro **`if`** seja verdadeiro, fazendo executar seu conteúdo, fazendo com que DR denote o "quanto a solução s_k é pior que a solução s^*_{bsa}" de forma que quanto maior o valor de DR, pior é a solução s_k em relação à solução s^*_{bsa}.

Nas situações em que for necessário atualizar o valor dos adaptadores, calcula-se o valor de γ_k com base no valor de DR, de forma que γ_k será diminuído se o fator de feromônio contribuiu mais que o fator de informação heurística para a solução ruim e, de maneira análoga, o γ_k será aumentado se o fator de informação heurística contribuiu mais do que o fator de feromônio para a solução ruim.

Dessa forma, para facilitar o entendimento, supõe-se como parâmetros $\alpha=1$ e $\beta=1$, de forma que nesta situação pode-se entender $\alpha_k=\gamma_k$ e $\beta_k=1-\gamma_k$. Assim, é claro que a medida que α_k for diminuído, se está diminuindo a importância do fator de feromônio e aumentando a importância do fator de informação heurística, enquanto que aumentando α_k ocorre o aumento da importância do fator de feromônio e a diminuição da importância do fator de informação heurística. Assim, se agora forem considerados valores de α e β diferentes de 1, será possível verificar que estarão exercendo uma tendência na importância dos fatores, de modo que se um é aumentado, o outro é diminuído.

Uma questão que talvez possa ser repensada nesse modo de atualização de adaptadores é que estes somente são atualizados se a formiga encontrar uma solução pior que a global, fazendo com que não haja recompensa positiva aos adaptadores caso contribuições sejam feitas no sentido de encontrar uma solução melhor que a global até a iteração anterior.

5.3 Considerações

Assim, a Tabela 5.9 apresenta uma síntese do que foi exposto. Nesta Tabela é interessante observar que a coluna intitulada "**Feromônio: Consulta**" identifica todos os aspectos da manipulação de feromônio.

Tabela 5.9. Síntese das variações possíveis no presente trabalho (aqueles que não estão sombreados)

Tipo Heurística	**Feromônio**			**Tipo Busca Local**	**Tipo Inf. Heurística**
	(Representação,	**Consulta,**	**Depósito)**		
MMAS_L	(VET,	**CMP** (E),	CMP)	NBL	CCL (E)
AS_RC_2	(MAT,	**SEQ** (E),	TOD)	JB2	CCB (D)
AAS_MC	(MAT,	**TOD** (D),	TOD)	RFL	
	(MAT,	TDN (D),	TOD)		

Sendo assim, será esse o campo que identificará no Capítulo 6, que contém os experimentos, qual é a manipulação de feromônio que está sendo utilizada.

6 Experimentos e Resultados

Com base em toda a fundamentação teórica apresentada apoiada na literatura ou em proposições do presente trabalho é realizada no presente Capítulo a experimentação prática da mesma, de forma que dois simuladores foram implementados a fim de reproduzir experimentos já realizados por outros trabalhos e testar as novas proposições do presente estudo.

Os experimentos preliminares deram origem à implementação própria dos experimentos *double bridge* (DB) e *extended double bridge* (EDB) contidos em Dorigo e Stützle (2004), sendo que o DB possui resultados reportados na seqüência.

No decorrer do estudo houve implementação de AS_LM, de forma que seus resultados e implementações não são reportados, mas foram aproveitados para implementação do MMAS_L e AS_RC_2, esses sim tendo resultados expostos no presente Capítulo.

6.1 Aspectos de Implementação

Foi utilizada a linguagem de programação C++ sobre a plataforma GNU/Linux para a implementação dos programas simuladores *Leonidas300* e *Optimus*. O simulador *Leonidas300* faz a aplicação do algoritmo *Simple-ACO* ao problema da ponte dupla – *double bridge* (DB) – e o simulador *Optimus* faz a aplicação de algoritmos heurísticos baseados em ACO ao PCC.

6.2 Experimentos Preliminares e seus Resultados

O experimento realizado foi a implementação do algoritmo *Simple-ACO* (Dorigo e Stützle, 2004) descrito na Seção 3.2, para encontrar o caminho mais curto no problema da ponte dupla, onde existem duas opções para a formiga chegar ao outro lado: um caminho mais curto e um caminho mais longo. O experimento realizado é o *Simple-ACO* aplicado ao problema *double bridge* (DB). A experiência reportada é a mais básica dessas variações, utilizando o valor de depósito de feromônio como um valor constante de $1{,}0$. O valor dos resíduos de feromônio foi inicialmente setado para $1{,}0$.

A quantidade de formigas foi variada dentre os valores 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, como ilustra a linha de título da tabela 6.1. Cada formiga executou 1000 movimentos por tentativa, sendo que o algoritmo teve execução de 100 tentativas. Ainda deve ser ressaltado que não houve evaporação, pois o fator de evaporação ρ foi setado para $\rho = 0{,}0$.

O valor α, chamado de fator de adequação de feromônio, que modifica a importância do valor do feromônio, teve valor $2{,}0$ nas execuções.

Os resultados obtidos são ilustrados na tabela 6.1 e comparados com os resultados obtidos por Dorigo e Stützle (2004).

Tabela 6.1. Simple-ACO com depósito de feromônio com valor constante. Conteúdo representa a quantidade de vezes (dentre 100) que o caminho mais longo foi selecionado

Experimento/Qt. formigas	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**
Dorigo e Stützle (2004)	50	42	26	29	24	18	3	2	1	0
O presente trabalho	55	31	23	9	1	2	0	0	0	0

Os números dos resultados representam a quantidade de vezes que o caminho mais longo foi selecionado de forma que quanto mais perto de 0 for o valor, melhor é o resultado da seleção. A implementação foi realizada de acordo com as especificações de Dorigo e Stützle (2004) e como pode-se observar pela Tabela 6.1, os resultados do experimento preliminar foram melhores que os alcançados por Dorigo e Stützle (2004) pelo fato de se poder utilizar menos formigas para obter o mesmo resultado, o que pode indicar uma melhora na complexidade computacional de tempo para execução do algoritmo. Porém, vale ressaltar que tal melhoria não possui um significado importante na questão de alguma melhoria na meta-heurística ACO de modo geral.

6.3 Instâncias de PCC e Experimento de Pré-Processamento para Redução do PCC

Para avaliar e validar, os algoritmos de interesse serão executados para resolver instâncias de problemas do PCC provenientes da *OR-Library* (Beasley, 1990), que é uma base de dados utilizada em trabalhos científicos, o que possibilita comparar os resultados de qualidade da solução e tempo de execução com outras implementações de algoritmos ACO e outras técnicas que realizem testes com a mesma base.

São utilizadas 70 instâncias dessa base, sendo tal conjunto de problemas constituído pelas chamadas sub-classes de problemas, a saber: scp4 contendo 10 instâncias, scp5 com 10 instâncias, scp6 com 5 instâncias, scpa com 5 instâncias, scpb com 5 instâncias, scpc com 5 instâncias, scpd com 5 instâncias, scpe com 5 instâncias, scpnre contendo 5 instâncias, scpnrf com 5 instâncias, scpnrg com 5 destas e scpnrh também com 5 instâncias. Tal conjunto de

instâncias equivalem a parte dos testes realizados por Lessing *et al.* (2004). A Tabela 6.2 expõe tais instâncias separadas por suas classes e sub-classes, denotando também os nomes de cada instância de problema bem como suas dimensões em termos de linhas (*m*) e colunas (*n*). Tal tabela ainda é munida da coluna intitulada "Reduzir" que indica se é benéfico realizar o pré-processamento para reduzir as instâncias do PCC que se enquadram em tal sub-classe, dado que o tempo para realizar o pré-processamento de redução também é computado no tempo de execução do algoritmo. Essa informação de redução foi obtida a partir de testes realizados para esse fim específico, onde se executou somente o algoritmo de redução apresentado na Seção 2.2 para cada instância, e foi avaliado se houve alguma redução e se o tempo computacional a ser dispensado para tal finalidade seria compensatório. O resultado se encontra na referida tabela.

Tabela 6.2. Instâncias scp

Classe Problema	Sub-classe Problema	Instância do Problema	*m*	*n*	Reduzir
scp	scp4	scp401, scp402, scp403, scp404, scp405, scp406, scp407, scp408, scp409, scp410	200	1000	SIM
	scp5	scp501, scp502, scp503, scp504, scp505, scp506, scp507, scp508, scp509, scp510	200	2000	SIM
	scp6	scp61, scp62, scp63, scp64, scp65	200	1000	NÃO
	scpa	scpa1, scpa2, scpa3, scpa4, scpa5	300	3000	SIM
	scpb	scpb1, scpb2, scpb3, scpb4, scpb5	300	3000	NÃO
	scpc	scpc1, scpc2, scpc3, scpc4, scpc5	400	4000	SIM
	scpd	scpd1, scpd2, scpd3, scpd4, scpd5	400	4000	NÃO
	scpe	scpe1, scpe2, scpe3, scpe4, scpe5	50	500	SIM
	scpnre	scpnre1, scpnre2, scpnre3, scpnre4, scpnre5	500	5000	NÃO
	scpnrf	scpnrf1, scpnrf2, scpnrf3, scpnrf4, scpnrf5	500	5000	NÃO
	scpnrg	scpnrg1, scpnrg2, scpnrg3, scpnrg4, scpnrg5	1000	10000	NÃO
	scpnrh	scpnrh1, scpnrh2, scpnrh3, scpnrh4, scpnrh5	1000	10000	NÃO

Os testes de redução também foram executados para as instâncias da classe scpclr e scpcyc, conforme é ilustrado na Tabela 6.3.

Tabela 6.3. Instâncias scpclr e scpcyc

Classe Problema	Sub-classe Problema	Instância do Problema	*m*	*n*	Reduzir
scpclr e scpcyc	scpclr	scpclr10, scpclr11, scpclr12, scpclr13	511, 1023, 2047 e 4095	210, 330, 495 e 715	SIM
	scpcyc	scpcyc06, scpcyc07, scpcyc08, scpcyc09, scpcyc10, scpcyc11	240, 672, 1792, 4608, 11520 e 28160	192, 448, 1024, 2304, 5120 e 11264	NÃO

Têm-se ainda os mesmos testes de redução para as instâncias rail, estas divididas em duas sub-classes: rail- que agrega as instâncias menores e rail+ que abriga as maiores. Tais testes podem ser verificados na Tabela 6.4.

Tabela 6.4. Instâncias rail

Classe Problema	Sub-classe Problema	Instância do Problema	*m*	*n*	Reduzir
rail	rail-	rail0507, rail0516, rail0582	507, 516 e 582	63009, 47311 e 55515	NÃO
	rail+	rail2536, rail2586, rail4284, rail4872	2536, 2586, 4284 e 4872	1081841, 920683, 1092610 e 968672	NÃO

As instâncias de PCC de Steiner *data* são ilustradas na Tabela 6.5 com o respectivo parecer a respeito da redução. Tal tabela também ilustra os problema *ladata*, que são obtidos a partir das instâncias *data* com a ajuda de algoritmo de Yagiura *et al.* (2006) .

Tabela 6.5. Instâncias data e ladata

Classe Problema	Sub-classe Problema	Instância do Problema	*m*	*n*	Reduzir
data	data	data.009, data.015, data.027, data.045, data.081, data.135, data.243	12, 35, 117, 330, 1080, 3015 e 9801	9, 15, 27, 45, 81, 135 e 243	NÃO
	ladata	ladata.027 (a partir de data.009*3), ladata.045 (a partir de data.015*3), ladata.081 (a partir de data.027*3), ladata.135 (a partir de data.045*3), ladata.243 (a partir de data.081*3), ladata.405 (a partir de data.135*3), ladata.729 (a partir de data.243*3)	117, 330, 1080, 3015, 9801, 27270 e 88452	27, 45, 81, 135, 243, 405 e 729	NÃO

Existem ainda outras instâncias da literatura que podem servir para experimentos, como as de Wedelin (Caprara *et al.*, 1999) e de Balas e Carrera (Caprara *et al.*, 1999).

Dessa forma, vale notar que a maioria dos experimentos reportados no presente Capítulo utiliza redução nas instâncias do PCC antes de aplicação de algoritmo heurístico baseado em ACO.

6.4 Parâmetros

O simulador criado para executar os experimentos permite que as tentativas nele inseridas sejam configuráveis. A Tabela 6.6 ilustra os parâmetros, fornece uma descrição sucinta de cada um deles juntamente com um valor de exemplo para cada um dos parâmetros.

Tabela 6.6: Descrição dos Parâmetros

Nome do Parâmetro	Valor Exemplo	Descrição do Significado
tipoProblema	PCC	Tipo do Problema tratado no experimento
qtFormigas	10	Número de formigas utilizadas
alpha	1.0	Valor do coeficiente alfa
beta	5.0	Valor do coeficiente beta
rho	0.3	Valor do coeficiente rô
rhoValorBase	0.3	Valor base para o rô
qtIteracoes	1000000	Quantidade de Iterações
qtIteracoesReinicializar	100	A distância em iterações da última melhoria conquistada e a reinicialização
qtIteracoesSemMelhora	1000000	Quantidade de iterações permitidas sem que haja melhoria
cicloMmasL	4	
aceleracao	0.0	Coeficiente de Aceleração
numVelocity	1	
tipoFreqAtualizacaoFeromonio	FIXO	
qtTentativas	10	Quantidade de tentativas
tempoMaxPorTentativa	200	Tempo máximo de execução da tentativa
tipoHeuristica	MMAS_L	Tipo da heurística utilizada na execução
fatorY	-1.0	Fator Y
fatorAMmasL	10.0	Fator AMmasL
considerarNTauMin	NAO	Considerar o NTauMin
fatorB	1.0	Fator B
fatorC	1.0	Fator C
tipoBuscaLocal	JB2	Tipo de Busca Local utilizada
rho1Jb	0.4	
rho2Jb	2.0	
maxTrialsRFlip = 200		
tipoInformacaoHeuristica	CCB	Tipo da Informação Heurística
q0	0.0	
xi	0.0	
zeta	0.0	Coeficiente Zeta
thetaAlter	0.0	
tipoRepresentacaoFeromonio	MAT	Tipo da Representação do Feromônio
tipoConsultaFeromonio	TOD	Tipo de Consulta de Feromônio
tipoDepositoFeromonio	TOD	Forma de depositar Feromônio
tipoInicializacaoFeromonio	TAU_MAX	Tipo de inicialização de Feromônio
tau	-1.0	
tipoAtualizacaoFeromonioMmasL	MELHOR_DA_ITERACAO	
tipoBaseFeromonioInicialMmasL	GREEDY	
pararSeEncontrarOtima	SIM	Parâmetro que indica ao algoritmo se a execução deve parar quando a solução ótima for encontrada
pararSeEncontrarOmc	NAO	Parâmetro que indica ao algoritmo se a execução deve parar quando a solução ótima ou melhor conhecida for encontrada
reduzirProblema	SIM	Parâmetro que indica se haverá a aplicação de algoritmos de redução ao problema
tauMax	-1	
tauMin	-1	
selPrimComptAleatorio	NAO	

6.5 Estrutura de um Experimento

Para organizar os experimentos de modo a ser possível referenciar seus resultados de forma prática utilizou-se a abstração experimento, que recebe um nome. Um experimento contém várias simulações, uma simulação possui execuções, as execuções tentativas e as tentativas iterações.

A simulação indica várias execuções com configurações diferentes. Uma execução denota a execução de várias tentativas com a mesma configuração – no presente trabalho foi usado 10 como quantidade de tentativas por execução. Sendo que uma tentativa identifica a execução completa de um algoritmo heurístico ACO. Iteração refere-se à uma iteração do laço de repetição do algoritmo.

6.6 Experimento *TOWARD* e Seus Resultados

O experimento *TOWARD* trabalha com as instâncias de PCC da *OR-Library* (Beasley, 1990) de forma a verificar o comportamento do programa em um cenário hipotético o qual disponibiliza um "grande" intervalo de iterações sem melhora para MMAS_L, mas com tempo limitado em 200 segundos e um "grande" intervalo de tempo para os testes no AS_RC_2. A intenção de tal formulação é criar um experimento que proporcione condições para que o programa:

- Alcance a solução ótima para o problema; ou
- exiba comportamento de estagnação.

O objetivo principal é a possibilidade de que a análise dos resultados do experimento possa indicar critérios de parada mais bem ajustados que os critérios de parada exagerados intencionalmente atribuídos ao próprio experimento *TOWARD*. Como meta secundária coloca-se a tentativa de verificação de melhorias nas configurações gerais dos experimentos de forma que possa melhorar o desempenho em relação à qualidade de solução.

Tal experimento disponibiliza os algoritmos ACO designados por MMAS_L e AS_RC_2.

6.6.1 Parâmetros

Os parâmetros dos experimentos com algoritmo heurístico MMAS_L e AS_RC_2 diferem nas questões inerentes à cada algoritmo e na questão de critérios de parada. Para o MMAS_L o tempo máximo de execução de uma tentativa foi definido como 200 segundos, tomando como base o valor de 100 segundos utilizado por Lessing *et al.* (2004) e colocando um valor significativamente maior com o intuito de se obter resultados semelhantes mesmo se a implementação dos algoritmos divergem na demanda de tempo computacional devido à questões de programação e plataforma computacional.

Por outro lado, as simulações com AS_RC_2 tiveram seu tempo máximo de tentativa setado como 2592000 segundos, que equivale à 30 dias, pretendendo-se com isso setar o tempo como virtualmente ilimitado. O critério de parada é aquele que faz parar quando uma

determinada quantidade de iterações sem melhora da solução incumbente é executada, que no caso a média das configurações utilizadas é de 11,76.

As diferenças a respeito dos critérios de parada entre os algoritmos se dá pelo fato de o MMAS_L ser um algoritmo já conhecido e que oferece testes contundentes para comparação e embasamento de parâmetro em Lessing *et al.* (2004), ao passo que o AS_RC_2 provém uma nova forma de se definir vizinhança uma vizinhança menor, porém os testes executados por Reis e Constantino (2003) não foram suficientes para definir uma linha geral de configuração de parâmetros que fornecessem soluções de boa qualidade juntamente com tempo de computação aceitável.

Sendo assim, a Tabela 6.7 ilustra parte dos parâmetros utilizados no experimento, separados por algoritmo heurístico.

Tabela 6.7. Parâmetros utilizados no experimento TOWARD

Parâmetro	MMAS_L	AS_RC_2
qtFormigas	**10**	**1000**
alpha	1.0	1.0
beta	5.0	5.0
rho	0.3	0.3
qtIteracoes	**1000000**	20
qtIteracoesReinicializar	100	8
qtIteracoesSemMelhora	**1000000**	11,76 (cinquenta e cinco com 12 e quinze com 2)
tipoFreqAtualizacaoFeromonio	FIXO	MRU
qtTentativas	10	10
tempoMaxPorTentativa	**200**	**2592000**
tipoHeuristica	MMAS_L	AS_RC_2
fatorY	0.0	2.0
fatorAMmasL	10.0	1
tipoBuscaLocal	NBL | JB2 | RFL	NBL | JB2 | RFL
rho1Jb	0.4	0.4
rho2Jb	2.0	2.0
maxTrialsRFlip	200	200
maxTimePerRFlipCall	180	180
tipoInformacaoHeuristica	CCB	CCB
(tipoRepresentacaoFeromonio, tipoConsultaFeromonio, tipoDepositoFeromonio)	(VET,**CMP**,CMP) | (MAT,**SEQ**,TOD) | (MAT,**TOD**,TOD)	(VET,**CMP**,CMP) | (MAT,**SEQ**,TOD) | (MAT,**TOD**,TOD)
tipoInicializacaoFeromonio	TAU_MAX	INFORMADA
tau	-1.0	1.0
tipoAtualizacaoFeromonio	MELHOR_DA_ITERACAO	MELHOR_DA_ITERACAO
pararSeEncontrarOtima	SIM	SIM
pararSeEncontrarOmc	NAO	NAO

No MMAS_L a decisão sobre o tipo de frequência de atualização de feromônio foi realizada da forma FIXA sugerida por Lessing *et al.* (2004).

No AS_RC_2 a decisão sobre o tipo de frequência de atualização de feromônio é dado por *ib* ou *bs* e foi obtida através de um algoritmo baseado no movimento retilíneo uniforme (MRU), porém devido à baixa quantidade de iterações executadas, é considerado apenas a modalidade *ib* para fins de análise.

6.6.2 Resultados e Análise

A presente Subseção apresenta diretamente os resultados do experimento separados por algoritmo, bem como realiza comparações, comentários e análises a respeito do que é reportado.

Uma questão de organização e terminologia importante de se ressaltar para a presente Subseção é que quando se diz que um experimento, simulação ou execução tem a configuração CMP, SEQ ou TOD se está referindo a manipulação de feromônio como um todo.

MMAS_L

Os resultados do experimento *TOWARD* para as variações que possuem o MMAS_L como algoritmo heurístico ACO são ilustrados na Tabela 6.8. Tal Tabela contém as médias e somas dos dados mostrando assim tendências de comportamento para as configurações referente aos algoritmos em questão.

Tabela 6.8. Resultados do experimento TOWARD para MMAS_L

Sum(Sim: (CsOMC/ Exec:AvgC) *Qt.Te.)	Sim: Tp. busca local	Sim: Tp.CF	Avg(Exec: Ti,s)	Avg(Exec:Avg (T:Ti,s))	Avg(Exec: %TOMC)	Avg(Exec: %AvgC)	Sum(Exec: QtOMC)	Avg(T: Ti,s)	Avg(T: QIT)	Avg(I: TIDT,s)	Avg(I: Iter)	Avg(S: %TOMC)	Avg(S: %OMC)
694,0739	NBL		1378,20	137,82	0,421	0,87	396	105,26	1632,17	42,26	52,21	-0,44	0,20
696,5315	JB2		1254,64	125,46	0,106	0,50	485	102,96	1278,97	35,96	55,04	-0,70	0,14
699,8958	RFL		773,79	77,38	-1,192	0,02	691	75,2	15,21	28,66	0,00	-1,77	0,00
2090,5012		CMP	1135,54	113,55	-0,222	0,46	1572	94,47	975,45	35,62	35,75	-0,97	0,11
693,8765	NBL		1411,40	141,14	0,655	0,90	409	113,41	1224,67	51,87	72,67	-0,21	0,32
695,8271	JB2		1305,30	130,53	0,452	0,61	473	108,39	1002,44	42,44	44,99	-0,36	0,25
699,9211	RFL		782,49	78,25	-1,170	0,01	695	75,96	14,99	29,61	0,01	-1,77	0,00
2089,6247		SEQ	1166,40	116,64	-0,021	0,51	1577	99,25	747,37	41,31	39,22	-0,78	0,19
691,1768	NBL		1481,10	148,11	1,418	1,31	353	123,41	985,24	60,91	71,06	0,36	0,50
695,1279	JB2		1372,96	137,30	0,636	0,72	442	110,81	818,66	53,51	47,60	-0,31	0,29
699,8389	RFL		768,27	76,83	-1,343	0,02	693	75,81	15,17	30,20	0,03	-1,67	0,00
2086,1437		TOD	1207,44	120,74	0,237	0,68	1488	103,35	606,36	48,21	39,56	-0,54	0,26
6266,2696		Total	1169,79	116,98	-0,002	0,55	4637	99,02	776,39	41,71	38,18	-0,76	0,19

Os títulos da Tabela 6.8 se utilizam dos operadores `Sum(...)` e `Avg(...)` para designar que o operando é uma soma ou uma média, respectivamente. Em tal Tabela também é utilizada a conceitualização de estrutura de experimento apresentada na Seção 6.5, de forma que o escopo de simulação – referente a cada linha – engloba várias execuções, onde cada execução pode ter configurações diferentes, e cada execução engloba várias tentativas, com cada tentativa tendo a mesma configuração se estas se encontram dentro de uma mesma execução. Outros termos e escopos são ilustrados na Tabela 6.9. Dessa forma, os dados representados na Tabela 6.8, referentes ao experimento *TOWARD* no que tange o algoritmo heurístico MMAS_L, mostram que cada simulação (linha) denota um conjunto de execuções com a variação mais significativa entre as configurações de suas diferentes execuções sendo o problema (dentre as 70 instâncias de teste). Assim, as várias execuções são agrupadas de acordo com configurações específicas de mais alta importância – nesse caso são `Tp.busca local` (sinônimo de `busca local`) e `Tp.CF` (sinônimo de `manip. fer.`) – para

formar uma simulação (linha), reportando desse modo a média do comportamento do algoritmo para o conjunto de problemas em questão nas referidas configurações.

Tabela 6.9. Significado dos títulos

Título	Operador mais geral	Escopo	Significado
`Sum(Sim : (CsOMC/Exec:AvgC)* Qt.Te.)`	Soma	Simulação	Soma de: custo da solução ótima ou melhor conhecida dividido pelo custo da solução média da execução multiplicado pela quantidade de tentativas da própria execução (utilizado por Lessing *et al.* (2004))
`Sim :Tp.b.local`	-	Simulação	Tipo de busca local da simulação
`Sim :Tp.CF.`	-	Simulação	Tipo de consulta de feromônio da simulação
`Avg(Exec:Ti,s)`	Média	Execução	Média do tempo (em segundos) da execução
`Avg(Exec:Avg(T:Ti,s))`	Média	Execução	Média entre as execuções de suas médias de tempo (em segundos) gasto por cada tentativa da execução
`Avg(Exec:%TOMC)`	Média	Execução	Média de percentual de distância do tamanho da solução média da execução em relação à solução ótima ou melhor conhecida
`Avg(Exec:%AvgC)`	Média	Execução	Média de percentual de distância do custo da solução média da execução em relação à solução ótima ou melhor conhecida
`Sum(Exec:QtOMC)`	Soma	Execução	Soma da quantidade de soluções ótimas ou melhores conhecidas encontradas na execução
`Avg(T :Ti,s)`	Média	Tentativa	Média de tempo gasto por cada tentativa da execução
`Avg(T :QIT)`	Média	Tentativa	Média de quantidade de iterações gastas para se encontrar a melhor solução da execução desde o início de sua respectiva tentativa
`Avg(I :TIDT,s)`	Média	Iteração	Média de tempo gasto para se encontrar a melhor solução da execução (para se chegar a sua iteração) desde o início de sua respectiva tentativa
`Avg(I :Iter)`	Média	Iteração	Média do índice da iteração em que a melhor solução da execução foi encontrada em sua respectiva tentativa
`Avg(S :%TOMC)`	Média	Solução	Média de percentual de distância do tamanho da melhor solução da execução em relação à solução ótima ou melhor conhecida
`Avg(S :%OMC)`	Média	Solução	Média de percentual de distância do custo da melhor solução da execução em relação à solução ótima ou melhor conhecida

Confrontando com trabalhos já existentes, é possível comparar parte dos resultados do experimento *TOWARD* com parte dos resultados dos experimentos obtidos por Lessing *et al.* (2004). As configurações das partes devem ter os mesmos parâmetros sendo que no *TOWARD* eles são dados por manipulação de feromônio setado como CMP e informação heurística com CCB com as variações de busca local NBL e RFL, sendo que sua correlação com parte dos resultados reportados em Lessing *et al.* (2004) pode ser verificada na Tabela 6.10, de modo que em tal Tabela `Sum(Exec:QtOMC)` equivale à n_{opt} em Lessing *et al.* (2004) e `Sum(Sim:(CsOMC/Exec:AvgC)*Qt.Te.)` equivale à *rel* na mesma referência.

Tabela 6.10. Comparação de resultados de MMAS_L e MMAS_LDS com informação heurística CCB

Busca local	**Algoritmo heurístico**	Sum(Exec: QtOMC) ou n_{opt}	Sum(Sim : (CsOMC/ Exec:AvgC)*Qt.Te) ou rel	**Tempo (s)**
NBL	MMAS_LDS (Lessing *et al.*, 2004)	492	696,4000	(**fixo**=min.=média=máx.) 100,00
	MMAS_L (do presente trabalho)	396	694,0739	(**média**) 137,82
RFL	MMAS_LDS (Lessing *et al.*, 2004)	685	699,6000	(**fixo**=min.=média=máx.) 100,00
	MMAS_L (do presente trabalho)	691	699,8958	(**média**) 77,38

Pelos dados ilustrados na Tabela 6.10, se for desconsiderado o tempo de execução, observa-se que:

- MMAS_L é pior que MMAS_LDS com NBL; e
- MMAS_L é melhor que MMAS_LDS com RFL.

Seguindo a análise existente em Lessing *et al.* (2004), a situação é que o experimento com MMAS_L possui resultados mais diversificados – e a princípio, piores na média – que os de MMAS_LDS, pois a aplicação de RFL no MMAS_L resultou em uma melhora sobre o MMAS_LDS, o que significa que as soluções mais diversificadas de MMAS_L forneceram subsídios para a busca local realizar um trabalho mais consistente.

Para verificar o comportamento dos algoritmos com variações dos parâmetros em relação à uma base experimental comum, no caso todas as simulações que têm MMAS_L como algoritmo ACO, a partir dos dados do experimento são gerados gráficos em que se pode observar e analisar os resultados do experimento de diferentes pontos de vista como segue no decorrer da presente seção.

Primeiramente são verificadas as características de busca local e da manipulação do feromônio com relação às duas questões mais importante em problemas de otimização:

- Qualidade da solução; e
- Tempo de execução.

A Figura 6.1 proporciona o confronto dessas questões a respeito dos resultados.

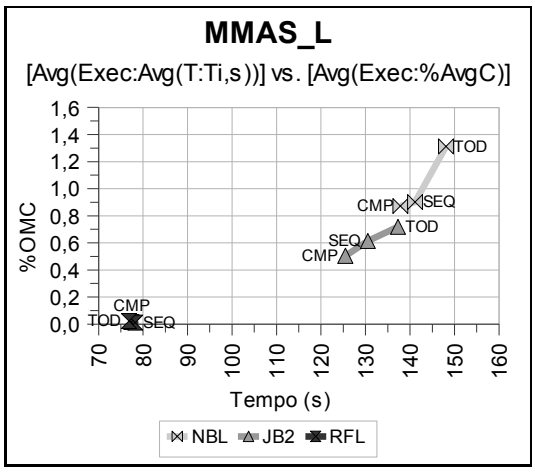


Figura 6.1. Qualidade da solução em função do tempo no MMAS_L

No eixo x está o tempo médio das tentativas dado por `Avg(Exec:Avg(T:Ti,s))`, enquanto que o eixo y denota a qualidade da solução em relação ao ótimo ou melhor solução conhecida, dado por `Avg(Exec:%AvgC)`. Assim, o gráfico permite uma análise bastante simples das características, seguindo a regra: **Quanto mais no canto inferior esquerdo está o ponto, melhor é tal configuração.**

A partir disso, verifica-se que:

- RFL tem o melhor desempenho, com pouca variação entre as diferentes formas de manipulação de feromônio;
- JB2 fica com o segundo melhor desempenho; e
- NBL é o pior dos três.

Deixando para análise subseqüente os casos do RFL, nota-se um padrão no comportamento dos diferentes tipos de manipulação de feromônio em NBL e JB2 de maneira que ficam elencados, do melhor para o pior, da forma:

1. CMP;
2. SEQ; e
3. TOD.

A Figura 6.2 coloca em escala mais aproximada apenas os resultados referentes à RFL. Dessa forma, é possível observar que:

- Comparando com o exibido em NBL e JB2, os resultados da manipulação de feromônio não segue os mesmos padrões;
- Não se tem uma configuração que seja melhor na qualidade da solução e no tempo de execução concomitantemente. Tem-se a configuração que oferte o melhor tempo por qualidade, que no caso é a CMP;
- A melhor qualidade fica por conta de SEQ porém com o pior tempo; e
- A pior qualidade é do TOD, porém com o melhor tempo.

Porém, vale notar que a diferença de qualidade é bastante pequena (termos da terceira casa da distância do ótimo em percentual) e o tempo também é pequeno (em torno de 2 segundos em 78, ou 0,026% de 78 segundos que é o tempo total). As análises são interessantes do ponto de vista teórico, para comparação com outras configurações e verificação das tendências do programa para uma possível modificação, porém na prática pode-se afirmar que haveriam resultados equivalentes na configuração em que se apresenta.

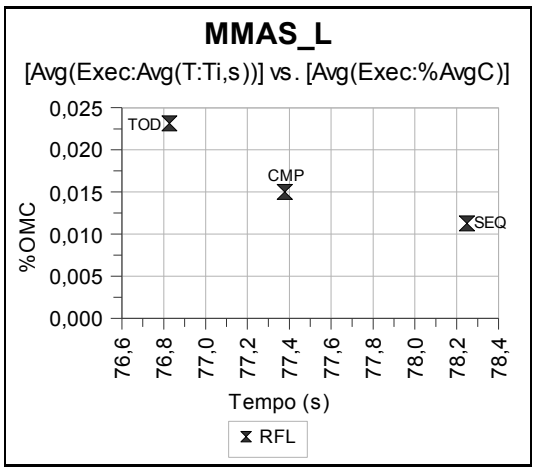


Figura 6.2. Comparação da manipulação de feromônio com RFL no MMAS_L

Comparando os desempenhos de cada manipulação de feromônio como uma linha no gráfico tem-se pontos para as buscas locais como pode ser visto no gráfico da Figura 6.3.

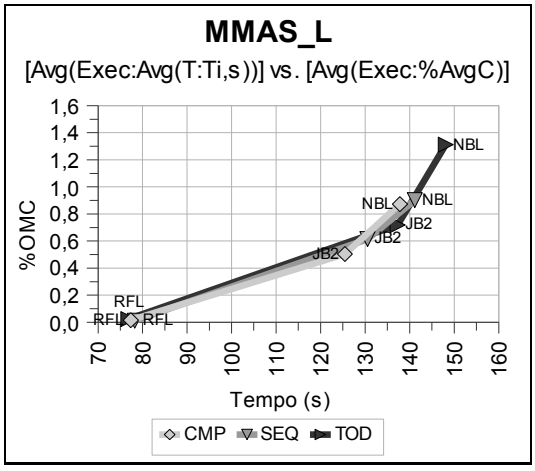


Figura 6.3. Qualidade da solução em função do tempo no MMAS_L (com linhas)

Tal abordagem revela que as relações entre os resultados das diferentes formas de manipulação de feromônio mudam entre diferentes buscas locais.

Como próximo passo verifica-se os resultados com a qualidade da solução e tempo de execução no mesmo gráfico de forma relacionada porém de forma a haver uma função para cada aspecto como pode verificar na Figura 6.4, e não vinculadas à mesma função como nos

exemplos anteriores. Assim, pode-se ter um subsídio mais ajustado para uma posterior calibração dos critérios de parada.

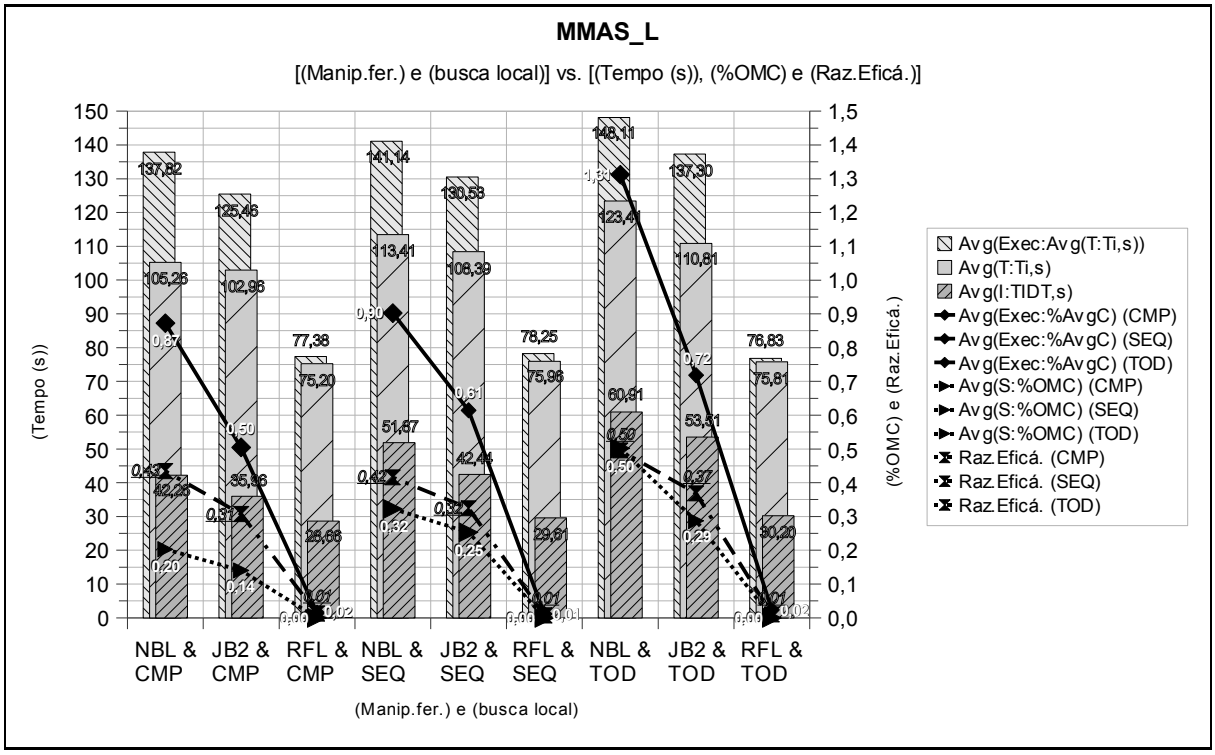


Figura 6.4. Tempo e qualidade da solução em função da manipulação de feromônio e da busca local

O valor razão de eficácia denotado nos gráficos por `Raz.Eficá.` diz respeito à razão entre a quantidade de vezes em que não se encontrou a solução ótima divida pela quantidade total de tentativas. Assim, um valor zero indica que o algoritmo encontrou a solução ótima em todas as tentativas que fez, sendo que todos os valores de `Raz.Eficá.` do gráfico da Figura 6.4 denotam que nenhum valor ótimo foi encontrado em nenhuma das tentativas.

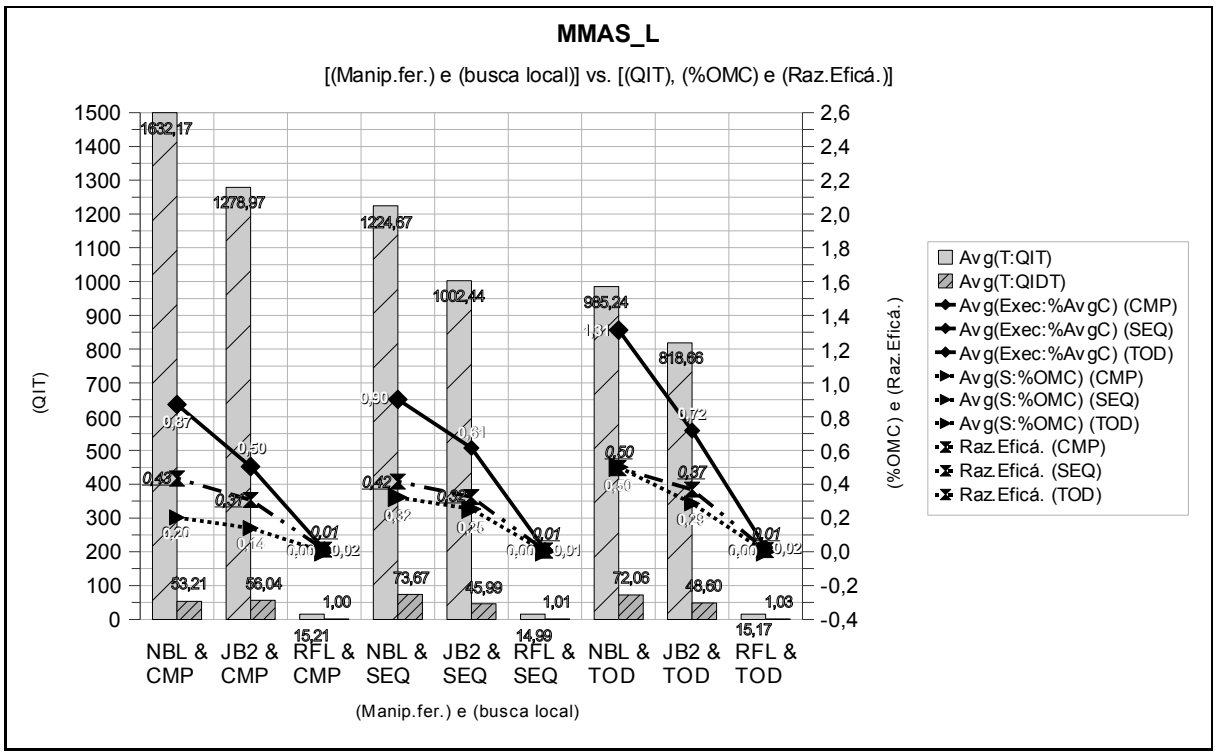


Figura 6.5. Iteração e qualidade da solução em função da manipulação de feromônio e da busca local

Com a pouca visualização possível a respeito do RFL na Figura 6.5, elaborou-se gráfico em escala adequada contendo apenas as questões do RFL, conforme pode ser visto na Figura 6.6.

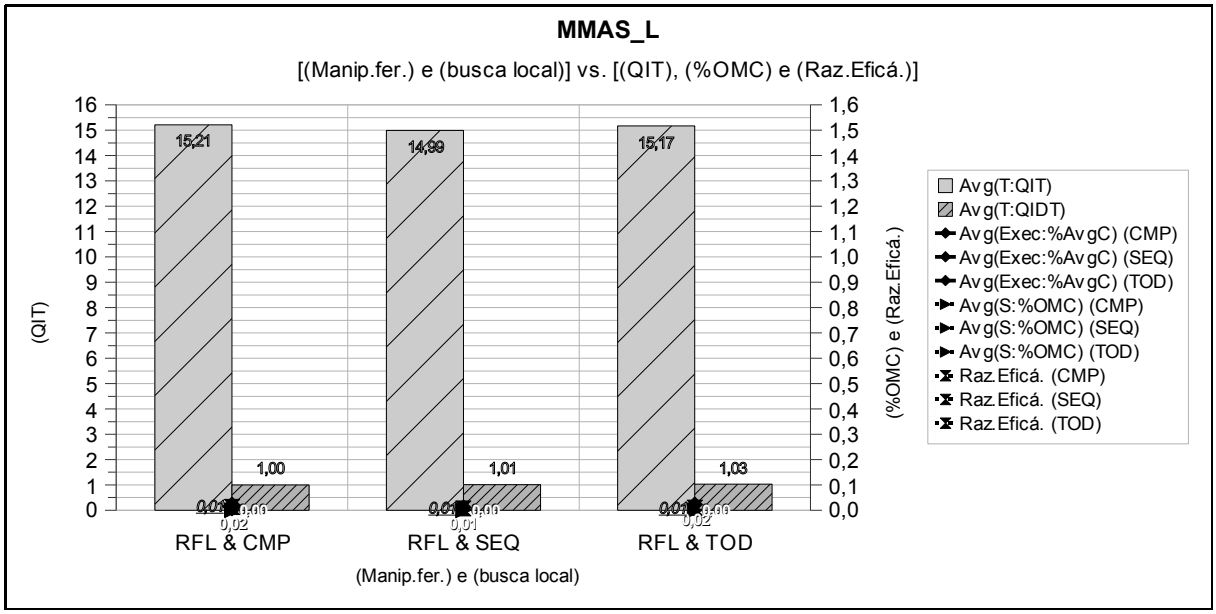


Figura 6.6. Iteração e qualidade da solução em função da manipulação de feromônio e do RFL

A mesma base de dados foi utilizada para a confecção da Figura 6.5, porém nesta ao invés da utilização dos tempos foram utilizadas as respectivas iterações. Assim, confrontando as Figuras 6.4 e 6.5 verifica-se como uma iteração da variação de cada algoritmo utiliza o tempo.

Na referida Figura 6.4, é possível ver duas classes de informações – cada qual vinculada a diferentes eixos y – e suas instâncias denotadas por seus respectivos tipos:

- (classe) qualidade de solução – eixo y secundário (da direita):
  - `Avg(Exec:%AvgC)` para `CMP`, `SEQ` e `TOD`; e
  - `Avg(S:%OMC)` para `CMP`, `SEQ` e `TOD`.
- (classe) tempo de execução – eixo x primário (da esquerda):
  - `Avg(Exec:Avg(T:Ti,s))`;
  - `Avg(T:Ti,s)`; e
  - `Avg(I:TIDT,s)`.

Do ponto de vista da qualidade da solução, o que poderia ser feito seria a mudança na configuração do experimento – gerais e específicas de critério de parada (que seriam válidas para instâncias com ótimos não conhecidos) – de forma a tentar fazer com que os dados da série `Avg(Exec:%AvgC)` se modifiquem de forma a tender ao comportamento de `Avg(S:%OMC)`. Porém tal calibragem fica em segundo plano no contexto do experimento *TOWARD* em questão.

A classe tempo de execução ilustra informações relevantes no que diz respeito à calibragem dos critérios de parada. É notório no gráfico da Figura 6.4 que `Avg(Exec:Avg(T:Ti,s))` está posicionado de forma que não é ultrapassado por `Avg(T:Ti,s)` nem `Avg(I:TIDT,s)`.

A comparação de `Avg(Exec:Avg(T:Ti,s))` com `Avg(T:Ti,s)` levanta a hipótese de que as tentativas consideradas a melhor em cada execução possuam tempo de execução sempre menor ou igual à média das tentativas de todas as execuções. Tal hipótese poderia ser comprovada pelo fato de que o tempo de execução é critério de desempate para o programa decidir entre soluções que possuam a mesma qualidade, selecionando aquela que tenha menor tempo como melhor da execução. Porém, deve ser levado em conta:

- O programa leva em consideração a **iteração** em que a solução é encontrada e não o tempo, e isso inclui na questão pequeno fator probabilístico do ponto de vista em que se analisa a situação, pois a mesma quantidade de iterações poderia denotar tempos diferentes de acordo com a formiga que encontra a solução, além de outras nuances de programação – de acordo com as observações empíricas, a probabilidade de a hipótese não se manter é verificada ser ínfima e não deve ter maiores implicações relevantes na prática.

É fato que `Avg(I:TIDT,s)` e `Avg(T:Ti,s)` pertencem à mesma tentativa, e sendo assim, `Avg(I:TIDT,s)` será sempre menor ou igual a `Avg(T:Ti,s)`, pois o tempo `Avg(I:TIDT,s)` está incluso em `Avg(T:Ti,s)`.

Com todas essas considerações, a prática muito provavelmente mostrará que `[Avg(Exec:Avg(T:Ti,s))]⩾[Avg(T:Ti,s)]⩾[Avg(I:TIDT,s)]` (a variante probabilística entra por causa da primeira inequação).

No experimento *TOWARD*, a instância de problema que possui a solução ótima tem sua tentativa terminada quando tal solução é alcançada – principalmente para não gastar tempo sem um sentido razoável nos testes. Assim, a análise deve ter enfoque diferente para cada um dos casos. Isso se dá pela configuração do parâmetro `pararSeEncontrarOtima=SIM`, que influencia diretamente a parada do algoritmo. Assim, são duas situações distintas:

- Instâncias de problemas que possuam solução ótima; e
- Instâncias de problemas que não possuam solução ótima.

O desafio nas duas é o mesmo: **obter um critério de parada que defina a hora certa de parar sem saber a solução ótima com antecedência**. A justificativa para tal desafio na segunda situação – que é clara – é que existe a possibilidade de se melhorar a melhor solução conhecida até o momento (além dos casos em que não se tem nenhuma informação a respeito da solução desejada), porém isso deve ser feito sem sacrificar o tempo de execução. Na primeira situação justifica-se considerando que o programa está sendo testado em instâncias de problemas conhecidas e com ótimos comprovados (e conseqüentemente conhecidos), mas a idéia é que o programa possa ser aplicado a problemas semelhantes, porém com soluções ótimas desconhecidas, além de servir como exemplo de comportamento para analisar o comportamento tendo o ótimo em mãos, daí a necessidade de se obter tal critério de parada.

Assim, o objetivo se converte em verificar o comportamento e inferir alguma espécie de regra que possa relacionar as variáveis relacionadas aos critérios de parada a seu efeito desejado. Tem-se os critérios de parada utilizados:

- **S**: Pára depois de uma determinada quantidade de iterações sem melhora;
- **I**: Pára depois de uma quantidade fixa de iterações a partir do inicio da tentativa;
- **T**: Pára depois de se ultrapassar desde o início da tentativa um tempo pré-definido;
- **O**: Se o ótimo for conhecido, pára se encontrá-lo;
- **M**: Pára quando a melhor solução conhecida for encontrada.

Com base nessas informações, o próximo passo é ir para as análises com base nos dados. Assim pode-se notar que `Avg(T:Ti,s)` pode ser maior que sua versão atual considerando que o programa pode – e provavelmente irá – gastar um tempo (que também pode ser medido em iterações) para perceber que deve parar, mas o valor de `Avg(T:Ti,s)` deve ser menor que `Avg(Exec:Avg(T:Ti,s))`, considerando as justificativas já apresentadas. Porém fica a pergunta: como se justifica a diferença bastante considerável das séries `Avg(T:Ti,s)` e `Avg(I:TIDT,s)` nos gráficos das Figuras 6.4 e 6.5? Parte da resposta é que o gráfico também inclui aquelas execuções em que o ótimo não é conhecido (e aquelas em que

o ótimo é conhecido e este não foi encontrado, e aquelas em que foi encontrado e o critério de parada foi outro), fazendo com que esses valores se afastem. Assim, a análise com as informações de forma separada denotam visão significativa de tal questão conforme é ilustrado pelas Figuras 6.7 e 6.9. Porém, é importante notar que o desafio nos dois casos é o mesmo: **obter um critério de parada que defina a hora certa de parar sem saber a solução ótima com antecedência**.

Assim, na Figura 6.7 é ilustrado o desempenho do algoritmo para os problemas em que a solução ótima não é conhecida.

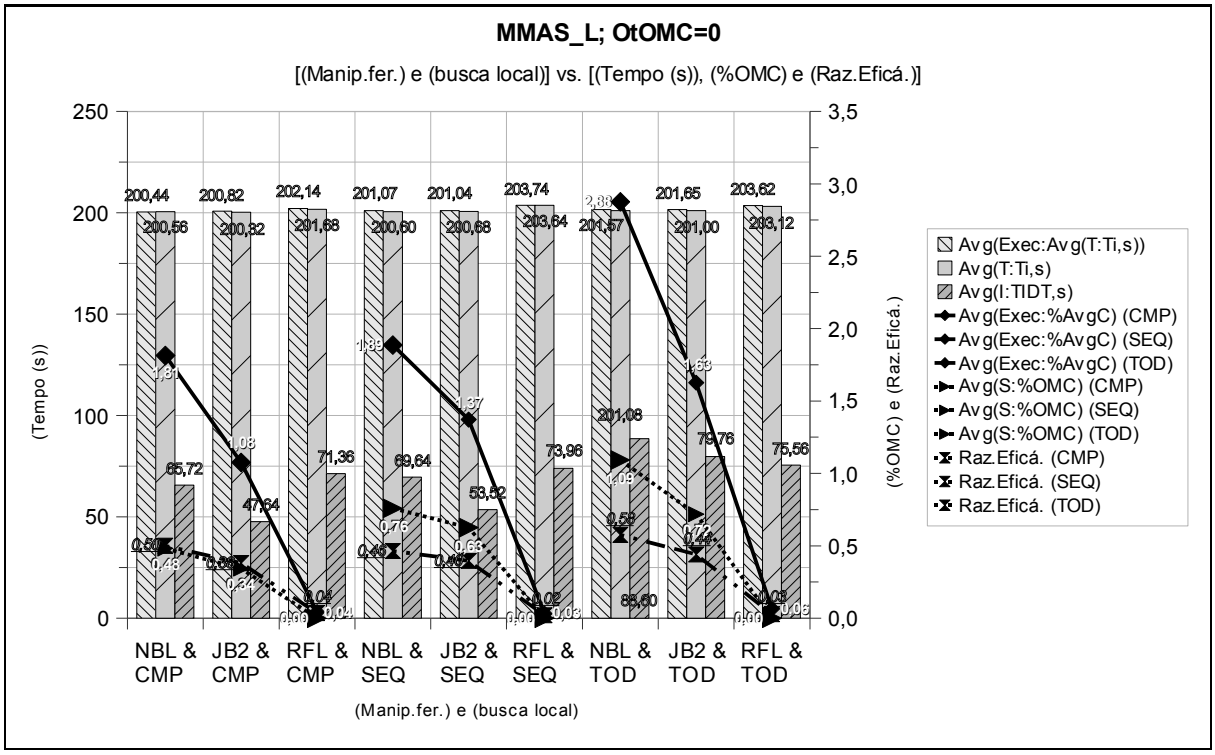


Figura 6.7. Tempo e qualidade da solução em função da manipulação de feromônio e da busca local para problemas que a melhor solução não é conhecida

Nota-se pela referida Figura que o valor de `Avg(T:Ti,s)` é bastante próximo do valor de `Avg(Exec:Avg(T:Ti,s))`, mostrando que a tentativa que encontra a melhor solução do algoritmo tem tempo bastante semelhante ao da média de todas as tentativas, de modo que atingem o valor máximo de 200 segundos de tempo permitido para o MMAS_L no *TOWARD*. O que é interessante de notar é que o tempo de encontrar a melhor solução pela primeira vez – dado por `Avg(I:TIDT,s)` – é significativamente menor que `Avg(T:Ti,s)` indicando que a diferença entre esses dois tempos é o tempo necessário para o algoritmo parar depois de encontrar a melhor solução reportada. Dessa forma, critério de parada bom nesses casos seria aquele que fizesse o tempo `Avg(T:Ti,s)` do algoritmo parar num tempo mais próximo de `Avg(I:TIDT,s)`, e não de `Avg(Exec:Avg(T:Ti,s))` como está ocorrendo.

Pela Figura 6.9 verifica-se o desempenho para problemas que possuem sua solução ótima conhecida.

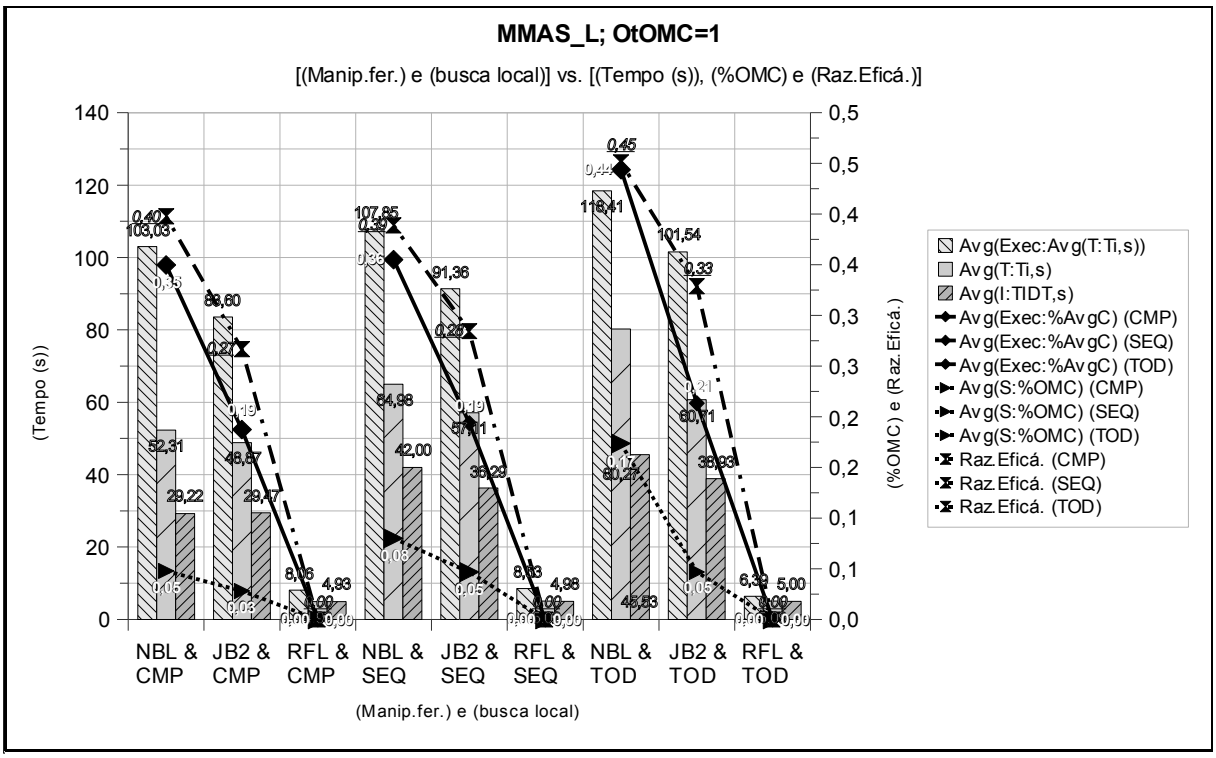


Figura 6.8. Tempo e qualidade da solução em função da manipulação de feromônio e da busca local para problemas que a melhor solução é conhecida

Mesmo separando as simulações referentes apenas à situações com ótimos conhecidos, ocorre a situação em que o valor de `Avg(T:Ti,s)` é distante de `Avg(I:TIDT,s)`, porém já é notória a tendência de que `Avg(T:Ti,s)` seja significativamente menor que `Avg(Exec:Avg(T:Ti,s))`.

Para tentar identificar as situações em que `Avg(T:Ti,s)` se aproxime de `Avg(I:TIDT,s)`, faz-se a separação das simulações que possuam solução ótima conhecida e cujo valor da média da distância da solução do ótimo – dado por `Avg(Exec:%AvgC)` – seja 0, o que significa que todas as tentativas resultaram em encontrar a solução ótima. Dessa forma tem-se tal formulação na Figura 6.10. Em tal gráfico é importante notar a presença do valor `Avg(Exec:Avg(I:TIDT,s))`, que indica para cada execução a média dos `Avg(I:TIDT,s)` de suas tentativas. Dessa forma, esse valor representa o comportamento médio do algoritmo, sendo este o comportamento mais provável em uma tentativa isolada.

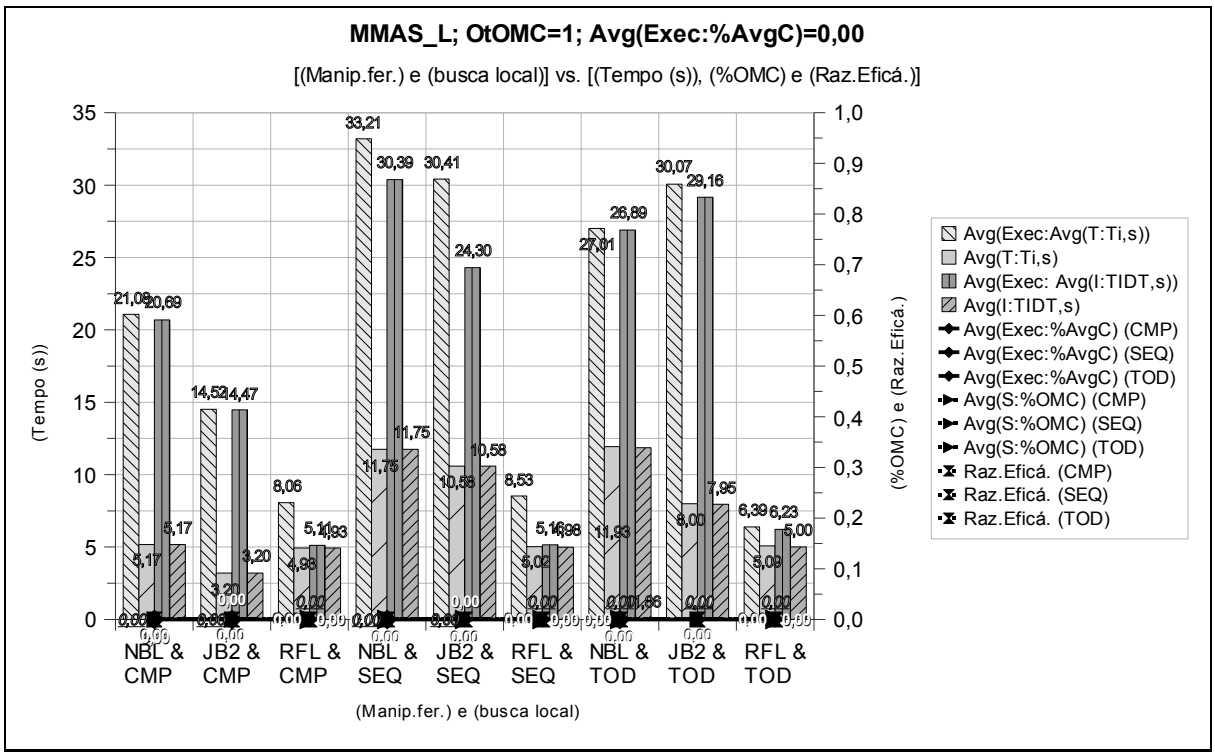


Figura 6.10. Tempo e qualidade da solução em função da manipulação de feromônio e da busca local para problemas que a melhor solução é conhecida e Avg(Exec:%AvgC)=0,00

Dessa forma, pode-se relacionar entre si os valores advindos de médias de tentativas de execuções `Avg(Exec:Avg(T:Ti,s))` e `Avg(Exec:Avg(I:TIDT,s))`, bem como os valores advindos de melhores tentativas de execuções `Avg(T:Ti,s)` e `Avg(I:TIDT,s)`. Assim, verifica-se que vale `[Avg(Exec:Avg(T:Ti,s))]>[Avg(Exec:Avg(I:TIDT,s))]` e `[Avg(T:Ti,s)]>[Avg(I:TIDT,s)]`. Diante dessas informações, tem-se ainda que a o valores de `Avg(I:TIDT,s)` são bem próximos de seus `Avg(T:Ti,s)` correlatos o que já mostra que o tempo para a parada está bem ajustado, porém a distância deste para `Avg(Exec:Avg(T:Ti,s))` – e consequentemente para `Avg(Exec:Avg(I:TIDT,s))` – ainda é distante. Tal comportamento pode ser explicado pelo critério de parada utilizado por algumas tentativas, por conta de que existem situações em que a solução ótima é conhecida, ela é encontrada pelo algoritmo em relação ao aspecto custo, porém o aspecto quantidade de colunas – que não está sendo reportado no gráfico da Figura 6.10 – não é tão bom quanto o informado na solução ótima, fazendo com que o algoritmo continue a execução até que algum critério de parada seja satisfeito, que não o critério que indica que a solução ótima foi encontrada.

Assim, mediante tal observação, a partir da base de dados utilizados na Figura 6.10 confecciona-se gráfico com o cenário:

- MMAS_L;
- `OtOMC` : 1;
- `Avg(Exec:%AvgC)` : 0,00; e

- `T:SITOM` : 11101 (se fosse substituído por `Avg(Exec:%TOMC)` o resultado final seria equivalente).

Tais configurações sobre a referida base de dados culmina no gráfico da Figura 6.11.

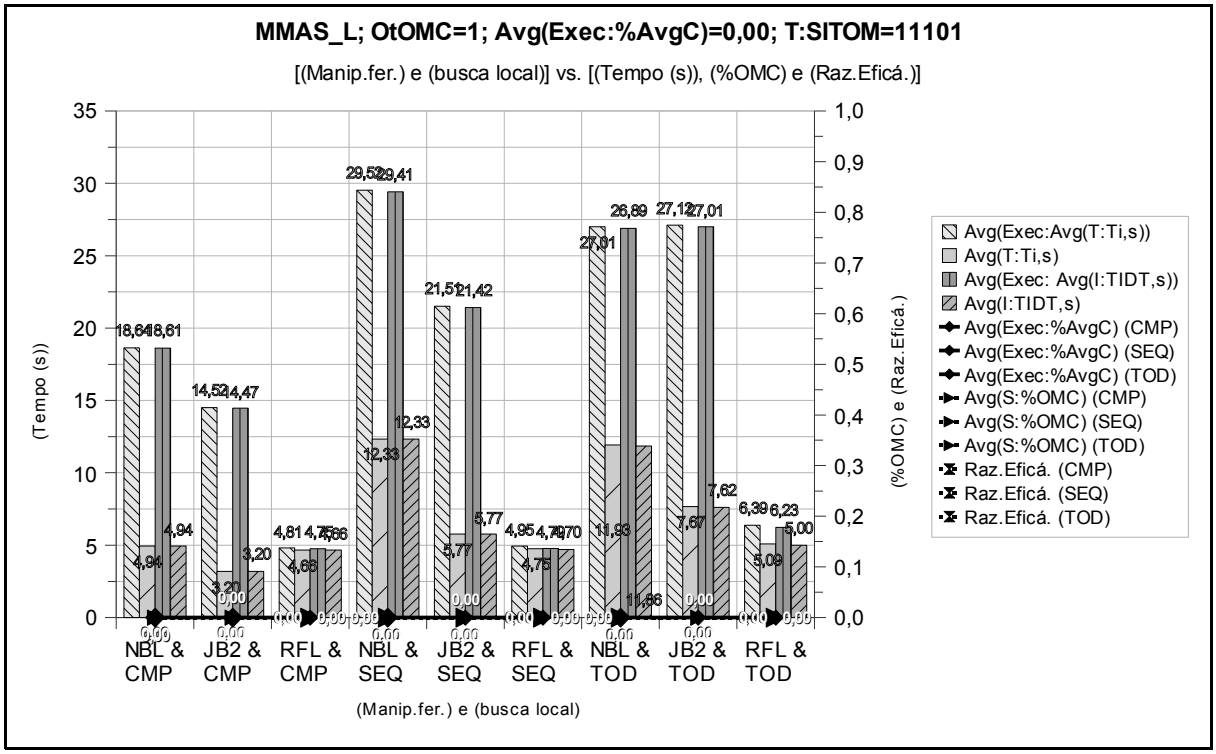


Figura 6.11. Tempo e qualidade da solução em função da manipulação de feromônio e da busca local para problemas que a melhor solução é conhecida e Avg(Exec:%AvgC)=0,00 e T:SITOM=11101

O que indica que no referido cenário tem-se que:

- `Avg(Exec:Avg(I:TIDT,s))` deve muito próximo de `Avg(Exec:Avg(T:Ti,s))` ; e
- `Avg(I:TIDT,s)` deve ser muito próximo de `Avg(T:Ti,s)` .

Isso é bastante útil, porque revela o cenário que os critérios de parada devem buscar mesmo em situações onde não se saiba o ótimo com antecedência (e claro, também em situações onde se saiba o ótimo, mas deve-se testar o algoritmo de modo que os critérios não estejam utilizando o critério de para por ótimo).

Uma situação que pode ser concluída e que serve como um dos guias na calibragem de parâmetros em geral – e não somente dos critérios de parada – é em relação à média dos `Avg(I:TIDT,s)` , dado por `Avg(Exec:Avg(I:TIDT,s))` . Dessa forma, tem-se que `Avg(Exec:Avg(I:TIDT,s))` revela a base de comportamento a ser perseguido em parametrizações de outras configurações nos aspectos que se seguem:

- Critérios de parada: calibrar para que `Avg(Exec:Avg(I:TIDT,s))` fique próximo (será sempre pouca coisa menor) ao valor de `Avg(Exec:Avg(T:Ti,s))` e
- Configurações gerais: calibrar para que `Avg(Exec:Avg(I:TIDT,s))` fique próximo (tenderá a ser maior) ao valor de `Avg(I:TIDT,s)` .

Vale notar, que a diminuição do tempo de execução mantendo a qualidade da solução só pode ser conseguida a partir de alterações nos parâmetros que não influenciam os critérios de parada.

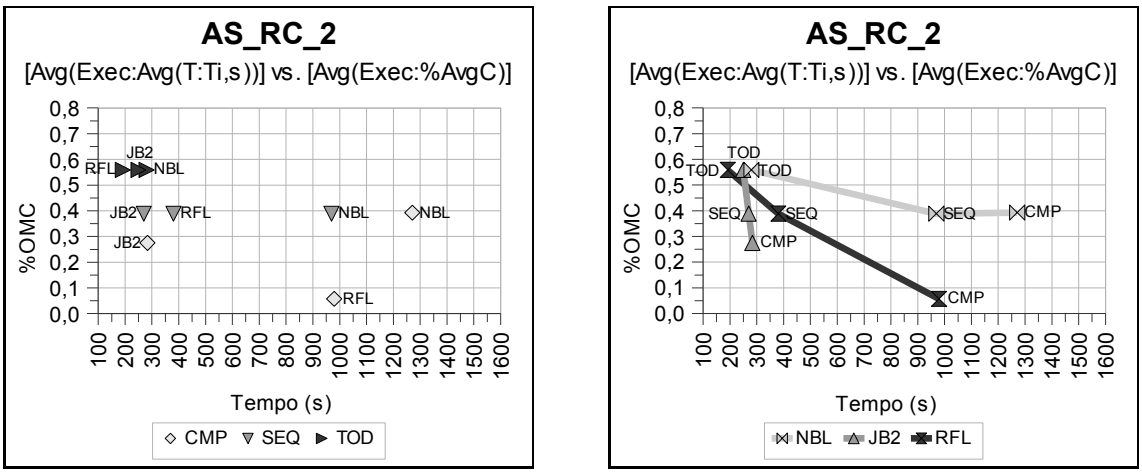
AS_RC_2

Os dados referentes ao AS_RC_2 são ilustrados na Tabela 6.11 com significado dos títulos análogo aos descritos para a Tabela 6.8. Além das várias diferenças na configuração básica em relação ao MMAS_L, tem-se que neste caso o tempo não é um fator considerado limitante, dado que o valor do tempo máximo por tentativa é setado como 2592000 segundos – dado pelo parâmetro `tempoMaxPorTentativa=2592000` – que é equivalente a 30 dias, o que não deve ser atingido dado as proporções dos problemas e dos algoritmos que estão sendo utilizados no presente trabalho, podendo-se interpretar que não há restrição de tempo.

Tabela 6.11. Resultados do experimento TOWARD para AS_RC_2

Sum(Sim: (CsOMC/ Exec:AvgC) *Qt.Te.)	Sim: Tp. b. Local	Sim: Tp.CF	Avg(Exec: Ti,s)	Avg(Exec:Avg (T:Ti,s)	Avg(Exec: %TOMC)	Avg(Exec: %AvgC)	Sum(Exec: QtOMC)	Avg(T: Ti,s)	Avg(T: QIT)	Avg(I: TIDT,s)	Avg(I: Iter)	Avg(S: %TOMC)	Avg(S: %OMC)
697,3089	NBL		12703,86	1270,39	-1,066	0,39	396	1147,53	7,21	428,09	2,17	-1,71	0,06
698,1273	JB2		2837,20	283,72	-0,912	0,27	585	244,80	6,51	101,00	1,66	-1,57	0,05
699,6081	RFL		9793,83	979,38	-1,429	0,06	684	928,93	2,79	401,86	0,00	-1,75	0,00
2095,0443		CMP	8444,96	844,50	-1,136	0,24	1665	773,75	5,50	310,31	1,28	-1,67	0,04
697,3555	NBL		9692,93	969,29	-0,943	0,39	424	884,33	7,80	323,66	2,80	-1,69	0,09
697,3555	JB2		2698,46	269,85	-0,943	0,39	528	252,79	7,80	88,57	2,80	-1,69	0,09
697,3555	RFL		3802,90	380,29	-0,943	0,39	528	136,06	7,80	64,09	2,80	-1,69	0,09
2092,0664		SEQ	5398,10	539,81	-0,943	0,39	1480	424,39	7,80	158,77	2,80	-1,69	0,09
696,1873	NBL		2798,24	279,82	-0,837	0,56	470	229,40	7,99	88,84	2,93	-1,50	0,10
696,1873	JB2		2499,33	249,93	-0,837	0,56	470	214,64	7,99	76,27	2,93	-1,50	0,10
696,1873	RFL		1919,63	191,96	-0,837	0,56	470	155,10	7,99	62,31	2,93	-1,50	0,10
2088,5618		TOD	2405,73	240,57	-0,837	0,56	1410	199,71	7,99	75,81	2,93	-1,50	0,10
6275,6725		Total	5416,26	541,63	-0,972	0,4	4555	465,95	7,10	181,63	2,33	-1,62	0,08

Como feito para o MMAS_L, faz-se também neste tópico interpretações dos dados apresentados na Tabela 6.11 como qualidade da solução em função do tempo de execução, como pode-se observar na Figura 6.12.



(a) *(b)*

Figura 6.12. Qualidade da solução em função do tempo no AS_RC_2

Na Figura 6.12-a é ilustrado a série da manipulação de feromônio variando-se a busca local. Já na Figura 6.12-b vê-se a série da busca local, variando-se a manipulação de

feromônio. Ambas as Figuras tratam exatamente os mesmo dados, porém a visualização torna-se facilitada de acordo com o ponto de vista adotado. Com base nisso e na Figura 6.12-a confeccionou-se a Figura 6.13 com visualização apenas dos dados referentes à manipulação de feromônio TOD, possibilitando que os valores da função possam ser observados em uma escala mais adequada.

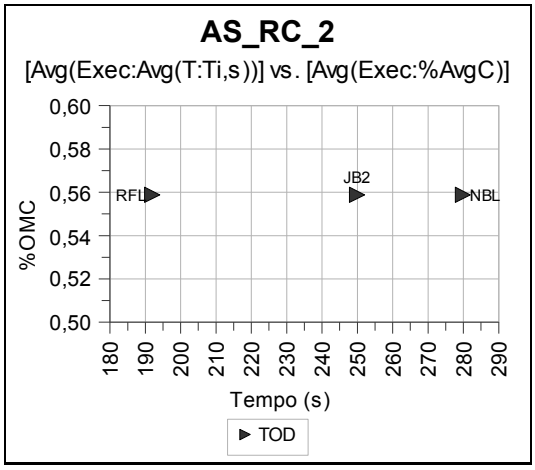


Figura 6.13. Comparação da manipulação de feromônios com R_FLIP no AS_RC_2

A partir dos gráficos das Figuras 6.12 e 6.13, verifica-se comportamentos interessantes:

- É interessante verificar o comportamento das variações com TOD e SEQ, pois em ambas:
    - A qualidade da solução é a mesma entre as variações de busca local de cada uma, variando apenas o tempo retornado pelas diferentes buscas locais;
    - Isso pode indicar que tais tipos de manipulação de feromônio podem estar levando o algoritmo a uma situação de estagnação (levando em conta também o critério de parada, por quantidade de iterações sem melhora) que as buscas locais não conseguem fazer muita coisa, a não ser melhorar o tempo de execução;
    - Uma outra coisa muito importante: mostra que o algoritmo, mesmo sem a busca local pode alcançar os resultados com a busca local se o tempo permitir;
- Muito interessante também o fato de CMP, NBL ter a mesma qualidade de solução, mas com um tempo pior, de todas as variações de SEQ, que nos dá a idéia de que a variação CMP é mais suscetível aos benefícios de uma busca local;
- CMP é a única forma de manipulação de feromônio que permite às busca locais melhorarem a qualidade da solução, estas ficando classificadas de acordo com a qualidade da solução de forma a acompanhar trabalhos já realizados (Lessing *et al.*, 2004) de maneira:

1. RFL;
2. JB2;
3. NBL;

- Assim, o melhor custo (tempo) por benefício (qualidade da solução) parece ser a variação com CMP e JB2, já que o CMP, RFL tem uma qualidade de solução superior, mas com um tempo de execução bastante elevado.

Focando na questão da obtenção de critérios de parada melhores, coisa que pode ser bastante difícil neste caso, dado que já se usa o quantidade de iterações sem melhora, tem-se os gráficos das Figuras 6.14 e 6.15. A tentativa principal é de fazer com que os valores de `Avg(Exec:Avg(T:Ti,s))` se aproxime de `Avg(T:Ti,s)` e este por sua vez se aproxime dos valores de `Avg(I:TIDT,s)`.

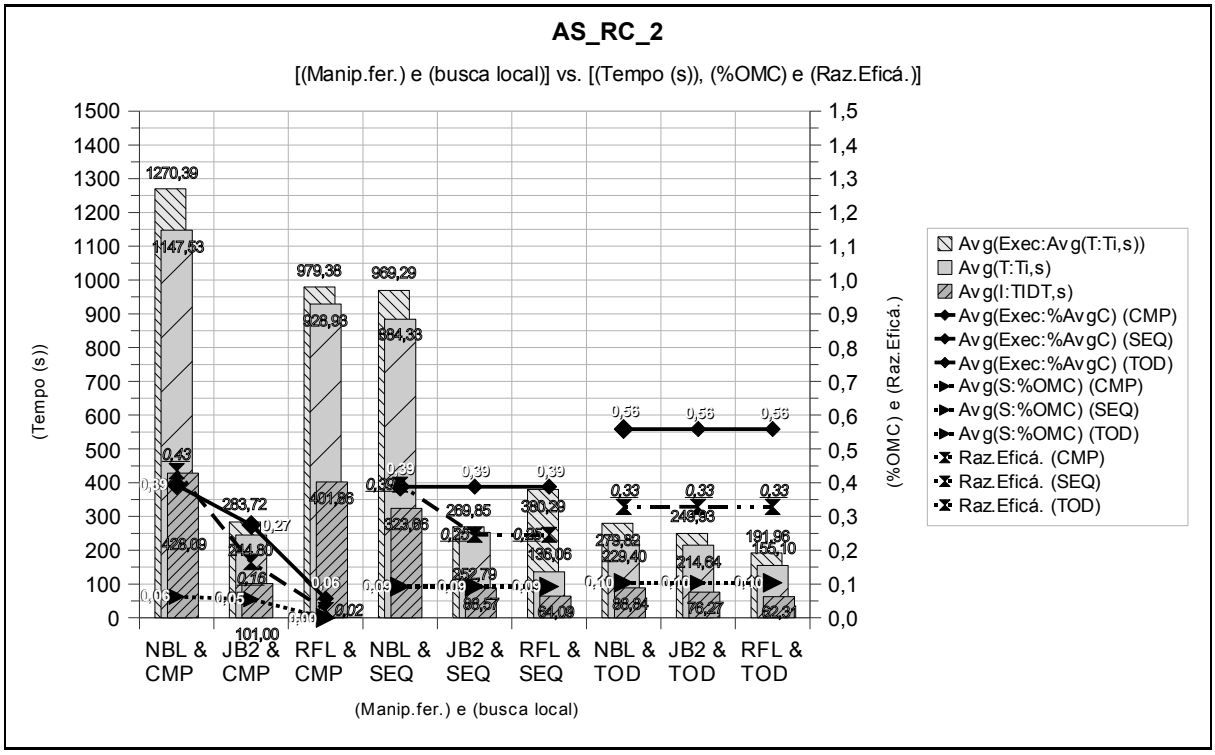


Figura 6.14. Tempo e qualidade da solução em função da manipulação de feromônio e da busca local

Algo que chama atenção de início é o comportamento das séries que indicam a qualidade da solução, verificando-se que para a manipulação de feromônio TOD os valores permanecem os mesmo para as três variações de busca local. Já para SEQ, os valores são os mesmos com exceção de que NBL é menos eficaz que JB2 e RFL, estes iguais no quesito. O CMP demonstrou ser mais suscetível a utilização de busca local, realizando a variação que lembra o comportamento com o MMAS_L, de forma que o RFL fica com a melhor qualidade da solução, porém diferentemente do MMAS_L seus tempos de execução não estão entre os melhores.

Outro ponto que se sobressai é a proporção significativa que existe entre si nos tempos `Avg(Exec:Avg(T:Ti,s))`, `Avg(T:Ti,s)` e `Avg(I:TIDT,s)` de cada variação busca local e manipulação de feromônio. Com base nisto, uma questão importante é que o valor de `Avg(Exec:Avg(T:Ti,s))` não é proporcionalmente muito maior que `Avg(T:Ti,s)` em nenhum dos casos.

Informações a respeito do mesmo contexto mas com foca nas quantidades de iterações executadas pelos algoritmos são demonstradas na Figura 6.15.

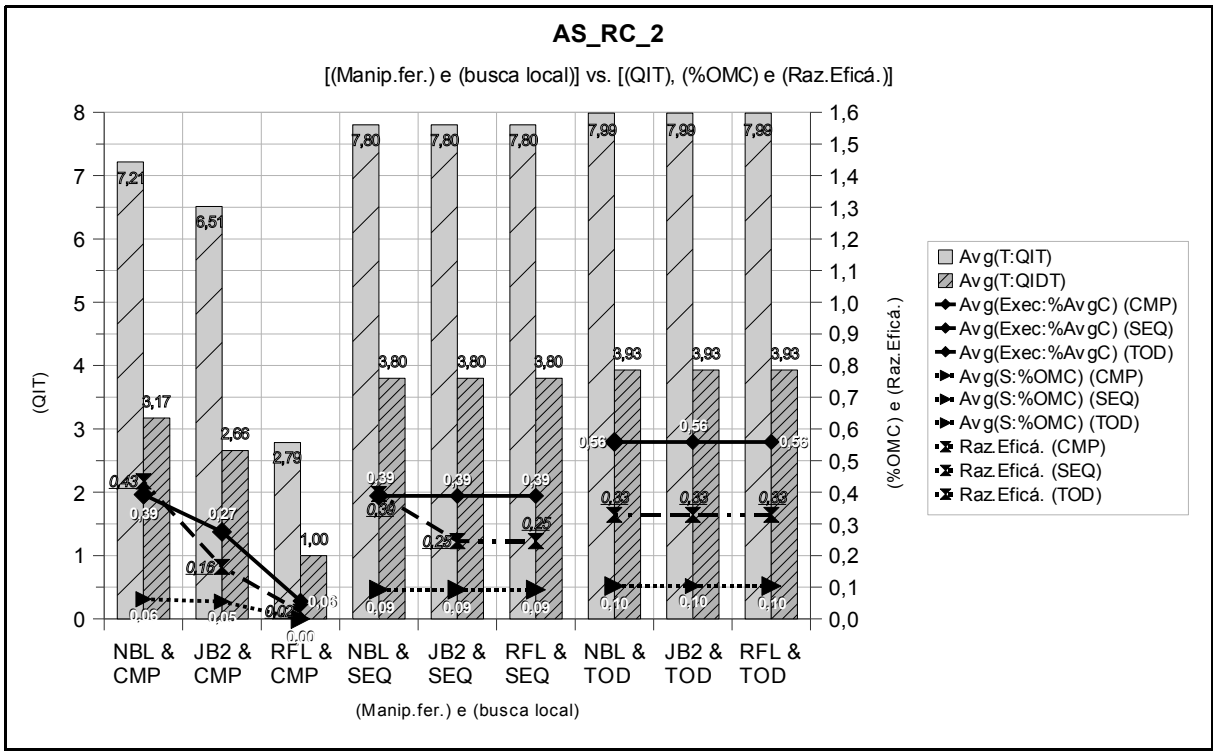


Figura 6.15. Iteração e qualidade da solução em função da manipulação de feromônio e da busca local

A primeira questão a ser levantada é a questão da quantidade de formigas, sendo que o AS_RC_2 de *TOWARD* está configurado com 1000 formigas – conforme parâmetro `qtFormigas=1000` – .o que faz com que cada iteração tenha um tempo de duração bastante elevado se comparado com o MMAS_L. Dessa forma, qualquer ganho em diminuir a quantidade de iterações teria um impacto significativo. O que faz também com que o algoritmo execute relativamente pouca quantidade de iterações para achar soluções que o MMAS_L utiliza muito mais iterações.

Outro ponto é quantidade de iterações que o algoritmo executa sem que haja melhora na solução, que neste caso foi setado para 12 – dado por `qtIteracoesSemMelhora=12` – exceto os que foram setados como 2 nas sub-classes de problemas scpnrf, scpnrg e scpnrh da variação CMP com RFL, fazendo com que a média de quantidade de iterações sem melhora seja 11,76.

É importante deixar registrado que o fato de `qtIteracoesSemMelhora=2` para algumas execuções de RFL com CMP pode ter inserido um pequeno viés em favor dessa variação, o que não invalida a situação diferenciada como um todo do CMP.

Sabendo que a quantidade máxima de iterações é setada como 20 – por meio de `qtIteracoes=20` – e focando atenção à Figura 6.15 vê-se que as melhores tentativas não chegam à metade disso. Somando-se a isso o fato de que os critérios majoritariamente prováveis de provocar a parada da execução do algoritmo são chegar a solução ótima ou a quantidade de iterações sem melhora. Dado que o primeiro faz com que nenhuma iteração extra seja executada, tem-se que o que faz o valor de `Avg(T:QIT)` ser maior que o valor de `Avg(T:QIDT)` são as execuções em que a solução ótima não é encontrada (mesmo as que possuem ótimos conhecidos). A média das somas de `[Avg(T:QIT)]−[Avg(T:QIDT)]` para todas as variações não é 3,76, sendo bem menor que 11,76 pela influência que as execuções que encontram soluções ótimas produzem nas médias.

Isto sugere que seria elucidante preparar gráficos e fazer a mesma análise separando as execuções pelos critérios de parada utilizados, assim como feito para MMAS_L. Porém tal reflexão é colocada como parte de trabalhos futuros.

6.6.3 Considerações de *TOWARD*

A partir das análises do experimento *TOWARD* com os algoritmos ACO setados como MMAS_L e AS_RC_2, cada qual com sua respectiva configuração base para as simulações pode-se definir aspectos relevantes que possam levar a uma melhor calibragem de parâmetros.

Um aspecto comum aos dois algoritmos é inerente à análise da quantidade de iterações, como pode ser verificado pelas Figuras 6.5 6.6 e 6.15, onde a quantidade de iterações executadas sem que tenha havido melhora na solução é dada por `[Avg(T:QIT)]−[Avg(T:QIDT)]`, sendo esse valor um guia de referência à escolha do valor de `qtIteracoesSemMelhora`. No caso do MMAS_L tem-se que `qtIteracoesSemMelhora` foi setado como 1000, o que torna tal guia importante para novos experimentos. Uma situação aplicável de imediato seria setar o valor de `qtIteracoesSemMelhora` para variação de manipulação de feromônio e busca local como o dobro da referida diferença nos respectivos casos, que já se teria um caminho – que deve ser comprovado empiricamente – que poderia levar a um melhor desempenho do algoritmo. Uma segunda situação candidata a novos experimentos seria setar `qtIteracoesSemMelhora` como a média do dobro das diferenças e verificar como o algoritmo se sai.

Embasado na questão da quantidade de iterações citada, no caso do AS_RC_2 tal análise mostra que o comportamento do algoritmo nesse aspecto está dentro de um valor intuitivamente pequeno e aceitável, porém recai sobre a quantidade de formigas a noção de

que uma possível redução no seu valor inicial de 1000 para algum valor menor, porém testes empíricos são necessário para verificar algo adequado.

6.7 Considerações

O presente Capítulo fornece uma idéia interessante do que é possível de ser feito na utilização de algoritmos heurísticos baseados em ACO aplicados ao PCC, de forma que analisa questões específicas dos algoritmos heurísticos bem como trabalha com o assunto de critério de parada, este que possui importância destacada em instâncias em que não se sabe a solução ótima com antecedência, sendo justamente tais problemas os interessantes de se aplicar o tipo de algoritmo proposto no presente trabalho.

Vale lembrar que para se obter os dados apresentados no experimento *TOWARD* foram necessárias 1152,56 horas-máquina – equivalente à 48,02 dias – de processamento para que execuções com diferentes configurações e quantidade significativa de tentativas fossem suscetíveis à análise e discussão.

7 Considerações Finais

O presente Capítulo conclui o presente estudo apresentando as conclusões analisando aspectos teóricos provenientes da proposta com embasamento na fundamentação teórica proveniente da revisão bibliográfica, bem como faz considerações a respeito dos resultados obtidos pelos experimentos realizados. Em seguida são apresentados os desdobramentos futuros sugeridos.

7.1 Conclusão

O propósito principal do presente trabalho é apresentado como estudo, melhoria e extensão de algoritmos heurísticos baseados na meta-heurística ACO aplicados ao PCC. O estudo é contemplado com uma revisão bibliográfica organizada de forma sistemática, da proposição de forma inovadora na maneira de representar, consultar e atualizar o feromônio nos algoritmos ACO para PCC estudados, caracterizando-se uma nova abordagem na denominada manipulação de feromônio. Também é proposto o algoritmo AAS_MC. Em experimentação prática é realizada a variação de tal manipulação de feromônio em conjunto com implementação própria dos algoritmos heurísticos ACO já existentes MMAS_L e AS_RC_2. Também foram utilizadas buscas locais configuradas como NBL, JB2 e RFL, além de ser possível utilizar as informações heurísticas CCL e CCB.

Para verificar o comportamento de tais algoritmos na prática, executou-se o experimento *TOWARD*, que consumiu 1152,56 horas-máquina de processamento, tempo equivalente à 48,02 dias. Os resultados provenientes dos testes realizados no experimento *TOWARD* mostram que o MMAS_L é bastante suscetível a melhorias na quantidade de iterações executadas quando utilizado com as técnicas de manipulação de feromônio SEQ e TOD, porém o tempo de execução e a qualidade da solução pioram. É notório também que o MMAS_L responde bem a utilização de buscas locais.

Os resultados também mostraram que, no geral, o AS_RC_2 piora sua solução e melhora o tempo de execução quando utilizado com as técnicas de manipulação de feromônio SEQ e TOD propostas pelo presente trabalho para o PCC. Porém, a aplicação de busca local a tal algoritmo faz com que SEQ e TOD mantenham a qualidade da solução com diminuição do

tempo de execução, ao passo que em CMP o tempo piora, mas a qualidade da solução melhora. Como principal conclusão empírica tem-se a visão de que o AS_RC_2 provém uma solução de boa qualidade com a quantidade de formigas relativamente elevada. Resta saber se é possível refinar tais resultados de modo a melhorar o alto tempo de computação em relação à outros algoritmos ACO, o que é de escopo do experimento *EVOLUTION*, a ser realizado futuramente.

Tal situação, juntamente com a interessante característica do AS_RC_2 de delimitar uma vizinhança relativamente pequena de componentes candidatos no momento de se escolher um novo componente no passo de construção da formiga, cria um horizonte de possibilidades de tentativas de mudanças e melhorias no AS_RC_2, pois se trata de uma abordagem inovadora de aplicação de ACO ao PCC a qual não é amplamente divulgada na literatura da área.

Dessa forma, o objetivo de se obter algoritmos heurísticos ACO com melhores resultados que algoritmos ACO similares existentes – ou que outros tipos de algoritmos heurísticos – para o PCC não foi atingido completamente na presente investigação, porém se resultados de desdobramentos futuros do presente trabalho obtiverem sucesso em tal tarefa, o presente estudo poderá vir a ter sua parcela de mérito.

Sobre o objetivo da calibragem dos parâmetros é necessário que se realize o experimento denominado *EVOLUTION*, que consiste em aplicar as tendências de parâmetros obtidos com a análise do experimento *TOWARD* exposto no presente trabalho, de modo a tentar melhorar os resultados para MMAS_L e AS_RC_2.

Não menos importante é a proposição do algoritmo AAS_RC, que trabalha com a idéia de adaptação da importância da influência exercida pelo valor de feromônio e pelo valor da informação heurística durante a execução do algoritmo, de modo que tal importância é modificada entre uma iteração e outra a fim de evitar que haja estagnação das soluções. Embora o AAS_MC ainda não possua uma implementação completa e funcional para que execute experimentos, a idéia pode ser testada futuramente para verificar seu comportamento.

7.2 Trabalhos Futuros

Diante das variadas nuances do presente trabalho, as Subseções seguintes apresentam os possíveis trabalhos futuros que podem se beneficiar do que é feito no presente trabalho, de forma a estender e modificar as fundamentações e proposições investigadas bem como se basear nos resultados reportados.

7.2.1 Experimento *EVOLUTION*

Tal experimento visa a realização de testes com o uso de parâmetros com base nas conclusões obtidas a partir do experimento *TOWARD*, que conforme colocado na Conclusão, realizar o experimento *EVOLUTION* é um trabalho complementar à presente investigação, de modo que ele deve permitir analisar os algoritmos estudados e implementados a partir de uma perspectiva mais ajustada à um tempo computacional aceitável.

7.2.2 Implementação e Estudo do AAS_MC

Outro desdobramento futuro intrínseco ao presente estudo é o término da implementação bem como a realização de experimentos e de análise de resultados do algoritmo AAS_MC, isso com base no que é especificado no presente trabalho a respeito de tal algoritmo.

7.2.3 Paralelização de Algoritmos ACO

A fim de melhorar o tempo de computação necessário para se obter soluções satisfatórias pode-se realizar investigações e avaliações no aspecto de paralelização da meta-heurística ACO, especialmente o AS_RC_2, que se mostrou bom quando utilizado com uma quantidade elevada de formigas, situação a qual a paralelização se mostra intuitivamente bastante acertada. A paralelização do AS_RC_2 também é interessante pelo fato de tal algoritmo não ser amplamente conhecido nem possuir proposição paralela na literatura especializada. Tal paralelização é possível de ter suas implementações elaboradas executadas em um *cluster* de computadores – equipamento que oferece um custo-benefício bom em relação a outros equipamentos que fornecem funcionalidades parecidas – utilizando a plataforma de programação paralela *Message Passing Interface* (MPI) (Pacheco, 1997), sendo tal *cluster* o hardware de processamento paralelo disponível na instituição onde o presente trabalho está sendo realizado. Uma das características da plataforma MPI é o suporte fornecido para que programas possam ser executados de forma distribuída em vários computadores simultaneamente, com suporte a primitivas de comunicação entre seus componentes. Logo, a arquitetura composta pelo *cluster* de computadores e a plataforma MPI provém processamento distribuído que podem ser usados para experimentos com diversos modelos paralelos de ACO (Bullnheimer *et al.*, 1997).

7.2.4 Manipulação de Feromônio Com Três ou Mais Dimensões

Neste trabalho apresentou-se a representação, a consulta e a atualização de feromônio com uma e duas dimensões. Também é possível verificar como ficariam os algoritmos e suas

definições, bem como o comportamento de cada um deles utilizando as referidas características em três ou mais dimensões.

7.2.5 Implementação, Experimentação e Análise da Consulta de Feromônio TDN

O tipo de consulta de feromônio TDN foi definido mas não foi implementado no presente trabalho, sendo que tal estudo pode ser extremamente enriquecedor no sentido de inovações no que diz respeito à manipulação de feromônio.

7.2.6 Informação Heurística

Os tipos de informação heurística para o PCC apontados por Lessing *et al.* (2004), em especial o custo de cobertura de Marchiori e Steenbeek, poderiam ser implementados a fim de serem testados com as inovações a respeito da manipulação de feromônio realizadas no presente trabalho.

7.2.7 Algoritmos Heurísticos ACO

Outros algoritmos ACO bastante conhecidos como ACS, ANTS, e MMAS-*Hybrid* (Lessing *et al.*, 2004) também poderiam ser testados com as variações de feromônio propostas no presente trabalho, incluindo aí a consulta de feromônio TDN.

7.2.8 Aplicação de Eliminação de Colunas Redundantes de Maneira Diferente

Aplicar a rotina de eliminar colunas redundantes apenas depois de aplicar busca local, diferentemente do que está sendo feito, onde colunas redundantes estão sendo eliminadas antes de se aplicar a busca local. Tal medida poderia ser benéfica no sentido de fornecer uma solução mais diversificada para a busca local trabalhar, além de causar uma tendência na diminuição do tempo computacional de uma iteração.

Referências Bibliográficas

ALAYA, I.; SOLNON, C. & GHEDIRA, K. *Ant algorithm for the multidimensional knapsack problem*, Ljubljana, Slovenie : 63-72, 2004.

BALAS, E. & HO, A. *Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study*, Math. Program. Study 12 : 37-60, 1980.

BEASLEY, J. *OR-Library: distributing test problems by electronic mail*, Journal of the Operational Research Society 41 : 1069-1072, 1990.

BLUM, C. *Beam-ACO - hybridizing ant colony optimization with beam search: an application to open shop scheduling.*, Computers \& OR 32 : 1565-1591, 2005.

BLUM, C.; ROLI, A. & DORIGO, M. *HC-ACO: The hyper-cube framework for Ant Colony Optimization*, Proceedings of MIC'CiteSeer. IST-Copyright Penn State and NEC , 2001.

BRUSCO, M.; JACOBS, L. & THOMPSON, G. *A morphing procedure to supplement a simulated annealing heuristic for cost-andcoverage-correlated set-covering problems*, Annals of Operations Research 86 : 611-627, 1999.

BULLNHEIMER, B.; KOTSIS, G. & STRAUSS, C. *Parallelization strategies for the Ant System*. : , 1997.

CAPRARA, A.; FISCHETTI, M. & TOTH, P. *A Heuristic Method for the Set Covering Problem*, Operations Research 47 : 730-743, 1999.

COLORNI, A.; DORIGO, M.; MANIEZZO, V. & OTHERS *Distributed optimization by ant colonies*. In: *Proceedings of the First European Conference on Artificial Life*, , : 134-142, 1991.

DESROCHERS, M. & SOUMIS, F. *A column generation approach to the urban transit crew scheduling problem*, Transportation science 23 : 1-13, 1989.

DORIGO, M. *Optimization, Learning and Natural Algorithms [em italiano]*. , : Ph. D. thesis, Politecnico di Milano IT, Dipartimento di Elettronica ed Informatica, 1992. p.

DORIGO, M. & GAMBARDELLA, L. M. *Ant colony system: a cooperative learning approach to the traveling salesman problem.*, IEEE Trans. Evolutionary Computation 1 : 53-66, 1997.

DORIGO, M.; MANIEZZO, V. & COLORNI, A. *Ant system: optimization by a colony of cooperating agents*, Systems, Man, and Cybernetics, Part B, IEEE Transactions on 26 : 29-41, 1996.

DORIGO, M. & STÜTZLE, T. *Ant Colony Optimization*. : Mit Press, 2004.

FIDANOVA, S. *Evolutionary algorithm for multidimensional knapsack problem*, Proceedings of PPSNVII , 2002.

FISHER, M. *The Lagrangian Relaxation Method for Solving Integer Programming Problems*, Management Science 27 : 1-18, 1981.

GAREY, M. & JOHNSON, D. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, , 1979.

HADJI, R.; RAHOUAL, M.; TALBI, E. & BACHELET, V. *Ant colonies for the set covering problem*. In: *Abstract proceedings of ANTS2000--From Ant Colonies to Artificial Ants: A Series of International Workshops on Ant Algorithms*, , : 63-66, 2000.

JACOBS, L. & BRUSCO, M. *A local-search heuristic for large set-covering problems*, Naval research logistics 42 : 1129-1140, 1995.

LEGUIZAMON, G. & MICHALEWICZ, Z. *A new version of ant system for subset problems*. In: *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, , 2, 1999.

LEGUIZAMON, G. & MICHALEWICZ, Z. *Ant Systems for Subset Problems*, Unpublished manuscript , 2000.

LESSING, L. *Ant colony optimization for the set covering problem [em alemão]*. , : Master's thesis, Fachgebiet Intellektik, Fachbereich Informatik, TU Darmstadt, Germany, 2004, 2004. p.

LESSING, L.; DUMITRESCU, I. & STUTZLE, T. *A Comparison Between ACO Algorithms for the Set Covering Problem*, Lecture Notes in Computer Science : 1-12, 2004.

MANIEZZO, V. *Exact and Approximate Nondeterministic Tree-Search Procedures for the Quadratic Assignment Problem*, INFORMS Journal on Computing 11 : 358-369, 1999.

MANIEZZO, V. & COLORNI, A. *The ant system applied to the quadratic assignment problem*, Knowledge and Data Engineering, IEEE Transactions on 11 : 769-778, 1999.

MARCHIORI, E. & STEENBEEK, A. *An Evolutionary Algorithm for Large Scale Set Covering Problems with Application to Airline Crew Scheduling*. In: *Real-World Applications of Evolutionary Computing: EvoWorkshops 2000: EvoIASP, EvoSCONDI, EvoTel, EvoSTIM, EvoRob, and EvoFlight, Edinburgh, Scotland, UK, April 17, 2000: Proceedings*, , : 367-381, 2000.

OLIVEIRA, N. V. D. *Problema de Cobertura de Conjuntos – Uma Comparação Numérica de Algoritmos Heurísticos*. , : Universidade Federal de Santa Catarina, Programa de Pós-Graduação em Engenharia de Produção, 1999. .

PACHECO, P. *Parallel Programming with MPI*. : Morgan Kaufmann, 1997.

RAMALHINHO-LOURENÇO, H. & SERRA, D. *Adaptive Approach Heuristics for the Generalized Assignment Problem*, , 1998.

REIS, P. A. & CONSTANTINO, A. A. *Algoritmos Genéticos e Ant System: Um estudo comparativo para resolução do Problema de Cobertura de Conjuntos*. Graduation Work, : Departamento de Informática, Universidade Estadual de Maringá, Maringá - PR - Brasil, 2003. .

SOLNON, C. & BRIDGE, D. *An ant colony optimization meta-heuristic for subset selection problems*, System Engineering using Particle Swarm Optimization, Nova Science : 7-29, 2006.

STUTZLE, T. & HOOS, H. *Max-Min Ant System and Local Search for Combinatorial Optimization*. In: *2 ndInternational Conference on Metaheuristics, Sophie-Antipolis, France*, , , 1997.

STÜTZLE, T. & HOOS, H. *Improvements on the Ant System: Introducing the MAX-MIN Ant System [C]*, Artificial Neural Networks and Genetic Algorithms, Wien New York: Springer Verlag : 245-249, 1995.

STÜTZLE, T. & HOOS, H. *MAX-MIN Ant system*, Future Generation Computer Systems 16 : 889-914, 2000.

YAGIURA, M.; KISHIDA, M. & IBARAKI, T. *A 3-flip neighborhood local search for the set covering problem*, European Journal of Operational Research 172 : 472-499, 2006.