UNIVERSIDADE SALVADOR – UNIFACS
MESTRADO EM SISTEMAS E COMPUTAÇÃO

JOÃO PAULO SANTANA LAMARTIN MONTES

ANTBEEPATH: UM ALGORITMO HÍBRIDO BIO-INSPIRADO PARA DETERMINAÇÃO DE ROTAS

Salvador
2012

JOÃO PAULO SANTANA LAMARTIN MONTES

ANTBEEPATH: UM ALGORITMO HÍBRIDO BIO-INSPIRADO PARA DETERMINAÇÃO DE ROTAS

Dissertação apresentada ao Mestrado em Sistemas e Computação da Universidade Salvador – UNIFACS, como requisito parcial para obtenção do título de Mestre em Sistemas e Computação.

Orientador: Joberto S. B. Martins.

Salvador
2012

Ficha Catalográfica elaborada pelo Sistema de Bibliotecas da Universidade Salvador – UNIFACS, Laureate Internacional Universities.

Montes, João Paulo Santana Lamartin

AntBeePath**:** um algoritmo híbrido bio-inspirado para determinação de rotas**./** João Paulo Santana Lamartin Montes . – Salvador, 2012

70 f. : il.

Dissertação apresentada ao Curso de Mestrado em Sistemas e Computação UNIFACS Universidade Salvador, Laureate Internacional Universities como requisito parcial para a obtenção do grau de Mestre.

Orientador: Prof. Dr. Joberto S. B. Martins.

1. Algoritmos. 2. Computação Bio-inspirada. I. Martins, Joberto S. B., orient. II. Título.

CDD. 005.133

JOÃO PAULO SANTANA LAMARTIN MONTES

ANTBEEPATH: UM ALGORITMO HÍBRIDO BIO-INSPIRADO PARA DETERMINAÇÃO DE ROTAS

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre em Sistemas e Computação, UNIFACS Universidade Salvador, Laureate Internacional Universities, pela seguinte banca examinadora:

Joberto S. B. Martins – Orientador _____
Docteur en Informatique, Université Pierre et Marie Curie
UNIFACS Universidade Salvador, Laureate Internacional Universities

Carlos Alberto Malcher Bastos _____
Doutor em Informática, Universidade Federal Fluminense
Universidade Federal Fluminense

Jorge Alberto Prado de Campos _____
Ph.D. in Spatial Information Science, University of Maine
UNIFACS Universidade Salvador, Laureate Internacional Universities

Salvador 24 de agosto de 2012

“Tendo em conta as condições de que dispõe e
na medida do possível, é a natureza que faz
sempre as coisas mais belas e melhores.”

Aristóteles

RESUMO

Este trabalho introduz o AntBeePath, um algoritmo híbrido bio-inspirado baseado no comportamento de duas espécies biológicas: as formigas e as abelhas. Ele foi criado como ferramenta de resolução do problema de determinação de menores caminhos em topologias de rede de computadores. O algoritmo combina o mecanismo de comunicação através da liberação de feromônio das formigas, popularizado por algoritmos de colônia de formigas (ACO) existentes, com um novo mecanismo bio-inspirado baseado na estratégia de recrutamento das abelhas. Três versões do algoritmo foram desenvolvidas de forma incremental. Resultados da prova de conceito realizada indicam que a versão Decay Chain Hybrid é mais eficiente do que as outras versões desenvolvidas e, além disso, apresentou um aumento de desempenho em relação a um algoritmo ACO equivalente. Os resultados sugerem que um algoritmo híbrido, combinando a liberação de feromônio das formigas com o novo mecanismo bio-inspirado de recrutamento das abelhas, associados a um mecanismo de controle de estagnação pode resultar em um novo algoritmo bio-inspirado capaz de determinar rotas.

Palavras-chave: Computação Bio-inspirada. Algoritmos de otimização de colônia de formigas – ACO. Algoritmos de otimização de colônia de abelhas – BCO. Determinação de rotas. Inteligência de Enxame.

ABSTRACT

This piece introduces the AntBeePath, a hybrid bio-inspired algorithm based on the behavior of ants and honeybees. It was designed as a tool for the resolution of the problem of finding the shortest paths for a given computer network topology. The algorithm, in brief, combines the pheromone release mechanism of existing Ant Colony Optimization (ACO) algorithms with a new bio-inspired mechanism based on the recruitment strategy of bees. Three versions of the algorithm were developed incrementally. Proof-of-concept results indicate that the AntBeePath Decay Hybrid Chain version is more efficient than the other developed versions and, beyond that, presented an improved performance in relation to an equivalent ACO algorithm. The results suggest that a hybrid algorithm, combining the ant's pheromone release with the new bio-inspired mechanism of bee recruitment along with a stagnation control mechanism can result in a new bio-inspired algorithm for path determination with improved characteristics.

Keywords: Bio-inspired computing. Ant colony optimization algorithms – ACO. Bee colony optimization algorithms – BCO. Path determination. Swarm Intelligence.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ABC	Ant-Based Control
ACK	Acknowledgement
ACO	Ant Colony Optimization
ACS	Ant Colony System
AODV	Ad-Hoc On-Demand Distance Vector Routing
ARA	Ant-Colony Based Routing
AS	Ant System
BCO	Bee Colony Optimization
BS	Bee System
DSDV	Dynamic Destination-Sequenced Distance-Vector
DSR	Dynamic Source Routing
IBM	International Business Machine
MMAS	Max Min Ant System
NTTnet	Nippon Telegraph & Telephone Network
PI	Path Integration
SI	Swarm Intelligence
SNA	Sistema Nervoso Autônomo
TCP	Transport Control Protocol
TSP	Travelling Salesman Problem
UDP	User Datagram Protocol

SUMÁRIO

1 INTRODUÇÃO

1.1 COMPUTAÇÃO BIO-INSPIRADA

A crescente complexidade dos sistemas computacionais tem levado os engenheiros e cientistas da computação a buscar inspiração nas mais diversas áreas. A matemática e a física sempre estiveram intrinsecamente ligadas à computação, porém nos últimos anos uma área diferente vem ganhando cada vez mais influência sobre a informática: a biologia.

Definida como a ciência que estuda a vida e os organismos vivos, sua estrutura, crescimento, funcionamento, reprodução, origem, evolução, distribuição, bem como suas relações com o ambiente e entre si (HOUAISS, 2009); a biologia pode ser vista por alguns como o oposto da tecnologia, porém nada poderia estar mais longe da verdade. De fato, os sistemas biológicos, através de bilhões de anos de evolução, conseguiram solucionar problemas que ainda afligem os atuais sistemas computacionais.

Da junção da computação com a biologia surgiu uma nova área de estudo, apropriadamente chamada de computação bio-inspirada. A proposta da computação bio-inspirada é identificar paralelos entre sistemas naturais e sistemas computacionais, e buscar aplicar na informática as soluções adotadas com sucesso pela natureza.

A noção de buscar inspiração na natureza para resolver problemas atuais não é nova. A história da ciência é rica em exemplos de grandes invenções que surgiram através de observações do mundo biológico.

Na década de quarenta, por exemplo, um engenheiro suíço chamado George de Mestral percebeu que quando levava seu cão para passear pelos Alpes Suíços, uma espécie de carrapicho se prendia ao seu pelo. Após examiná-lo com o microscópio, de Mestral percebeu que o carrapicho era composto de pequenos ganchos que se agarravam a qualquer coisa que tivesse um *loop*, como o pelo de seu cachorro ou até mesmo ao tecido de sua calça. Após anos de pesquisa e desenvolvimento George de Mestral patenteou a sua invenção, a que batizou de velcro, a combinação das primeiras três letras de duas palavras em francês: velours (veludo) e crochet (gancho).

Algumas décadas depois, John H. Holland (a primeira pessoa nos Estados Unidos a receber um Ph.D. em Ciência da Computação), se baseou no fenômeno de evolução tal qual conhecemos na natureza para desenvolver o conceito de algoritmos genéticos (HOLLAND,

1975). O objetivo de Holland era entender o mecanismo de adaptação natural e desenvolver formas nas quais tais mecanismos pudessem ser transpostos para sistemas computacionais (MITCHELL, 1995).

Além de servir como inspiração para o desenvolvimento de soluções tais como o velcro e os algoritmos genéticos, a biologia tem muito mais a oferecer ao avanço da tecnologia, em especial a ciência da computação. De acordo com Dobson (2006), sistemas biológicos tendem a explorar modelos descentralizados com alto nível de desacoplamento, dependo primariamente da transferência de informação através do ambiente. Isso se traduz em propriedades desejáveis em um sistema computacional, tais como: adaptabilidade, sob condições e ambientes em constante alteração; escalabilidade, podendo ser aplicado a poucos ou até mesmo a milhões de nós; além de robustez, a capacidade de superar problemas, falhas e ataques e restabelecer o funcionamento normal do sistema.

A aproximação do relacionamento entre a biologia e a computação é importante porque os modelos computacionais atuais têm impulsionado o desenvolvimento de sistemas distribuídos cada vez mais complexos e heterogêneos.

Em resposta a essa crescente complexidade a IBM divulgou um manifesto em 2001 no qual introduziu a computação autonômica (HORN, 2001). Um dos pontos mais importantes discutido nesse estudo é o desafio, cada vez maior, de se construir, administrar e gerenciar sistemas computacionais cada vez mais complexos, compostos, muitas vezes, de uma miríade de tecnologias de hardware e software diferentes, de forma a fazer tudo funcionar de maneira integrada e eficiente. O problema é que os seres humanos não seriam mais capazes de lidar com a complexidade desses sistemas de maneira eficaz. Para solucionar esta questão, a IBM se baseou na natureza para criar um novo conceito na computação, uma verdadeira quebra de paradigma, a computação autonômica.

A inspiração por trás da computação autonômica veio da biologia, mais especificamente do sistema nervoso autônomo, responsável pelo controle de uma série de processos que ocorrem no corpo humano. Entre as diversas funções realizadas pelo sistema nervoso autônomo (SNA) destacam-se o controle da frequência cardíaca, temperatura corporal, pressão sanguínea, digestão, respiração, além de diversas outras tarefas essenciais à manutenção da vida humana (KHAN, 2010).

1.2 PROBLEMÁTICA

Este estudo se concentra em investigar o problema de determinação de rotas sobre a nova ótica da computação bio-inspirada. As questões que esse estudo se propôs a solucionar foram:

a) É possível modelar o comportamento de duas espécies biológicas e combiná-los em uma única solução para resolver o problema de determinação de rotas?

b) Caso a afirmativa acima seja verdadeira, existiriam maneiras de aumentar a desempenho desta solução?

c) Como essa solução se compararia a uma solução equivalente já existente na literatura?

1.3 OBJETIVO

O propósito deste trabalho é propor um novo algoritmo capaz de combinar as características de duas espécies biológicas, a formiga e a abelha, na resolução do problema de determinação de rotas. Mais do que apenas sugeri-lo, este estudo objetiva analisar o comportamento do algoritmo, em termos de eficiência, e, se possível, implementar mecanismos que possam melhorar o seu desempenho.

1.4 ESTRUTURA DO TRABALHO

O presente trabalho se encontra divido em cinco capítulos:

1. O primeiro capítulo faz uma breve introdução sobre a área da computação bio-inspirada, além de introduzir a problemática e o propósito desta dissertação.

2. O capítulo dois está dividido em três partes e reúne uma compreensiva (mas não exaustiva) revisão da literatura, analisando os algoritmos que precederam e inspiraram de alguma forma o desenvolvimento do algoritmo AntBeePath que é proposto nesta dissertação. A primeira parte deste capítulo trata dos algoritmos inspirados em colônia de formigas, já a segunda aborda os

algoritmos inspirados em colônias de abelhas. A terceira e última parte do capítulo dois descreve brevemente alguns outros tipos de algoritmos bio-inspirados.

3. A terceira seção introduz o algoritmo AntBeePath. A seção se divide em quatro partes:
   a. A primeira traz uma descrição de alto nível do algoritmo.
   b. A segunda caracteriza os estados que compõem os agentes do algoritmo.
   c. A terceira parte detalha a parte operacional do algoritmo.
   d. A quarta expõe a maneira com a qual se deu o desenvolvimento (implementação) do algoritmo.

4. A quarta seção desta dissertação descreve a prova de conceito à qual foi submetida o algoritmo AntBeePath. A seção está divida em duas subseções. A primeira descreve a maneira com qual foram definidos os parâmetros dos experimentos que foram realizados, enquanto a segunda trata dos resultados dos testes propriamente ditos.

5. A quinta e última seção deste trabalho traz as conclusões alcançados neste estudo.

2 REVISÃO DA LITERATURA – IDENTIFICAÇÃO DO ESTADO DA ARTE

Não existem dúvidas sobre a infinidade de sistemas biológicos que compõem o planeta terra. Este estudo tem como foco a busca por paralelos biológicos que possam vir a solucionar problemas de roteamento em redes de computadores comutados por pacotes e se concentra principalmente na área da computação bio-inspirada que investiga a inteligência de enxame.

O conceito de inteligência de enxame, da expressão em inglês *Swarm Intelligence* (*SI*), foi introduzido em um trabalho sobre um sistema robótico distribuído no qual pequenos robôs não-inteligentes cooperam na realização de determinadas tarefas em nome de um objetivo comum e que é alcançado de forma não previsível (BENI; WANG,1989).

A premissa básica da inteligência de enxame, e que pode ser transposta para as mais diversas áreas, consiste na existência de um sistema distribuído composto por elementos de pequena complexidade encarregados de funções simples, mas que se sobrepõem e interagem entre si de forma a agregar valor e produzir sinergia. Da soma de comportamentos individuais de aparente simplicidade, surge um complexo padrão global.

Diversos sistemas biológicos apresentam as características referenciadas pela inteligência de enxame, porém este estudo se limita a analisar o comportamento de algumas espécies de insetos sociais, mais notadamente as formigas e abelhas.

2.1 OTIMIZAÇÃO DE COLÔNIA DE FORMIGAS - ANT COLONY OPTIMIZATION ACO)

2.1.1 Conceito

A primeira vista uma colônia de formigas pode aparentar ser um sistema simples e sem muita sofisticação. No entanto, uma inspeção mais cuidadosa revelará que esses diminutos insetos são mestres na arte da busca. Ao sair da colônia em busca de alimento, a formiga parte em uma direção randômica e deixa por onde passa uma trilha de feromônio. Ao encontrar uma fonte de alimento a formiga então retorna pelo caminho e reforça a trilha liberando ainda mais feromônio. Quando o caminho de uma segunda formiga se cruza com a trilha deixada pela primeira, esta tem uma probabilidade de segui-la e se ela assim o fizer, deixará o caminho ainda mais atraente para novas formigas; vale dizer, quanto mais formigas

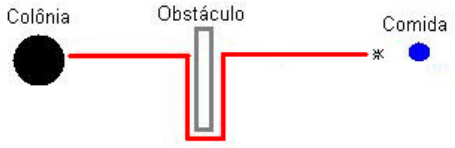
trafegarem pela rota, maior será a probabilidade de uma nova formiga decidir seguir o caminho.

Esse padrão de comportamento, no qual uma formiga introduz uma modificação no seu ambiente que afeta e estimula outras formigas a realizarem uma ação que reforça a anterior e leva ao surgimento de um esforço cooperativo é conhecido como stigmergia e foi introduzido pela primeira vez pelo biólogo francês Grassé em seu tratado sobre cupins (GRASSÉ, 1959).

Além da stigmergia, uma das características mais interessantes acerca da busca realizada pelas formigas é que mesmo após o estabelecimento de uma trilha já seguida por centenas de formigas, quando uma nova formiga se aproxima desta, a probabilidade de segui-la não é 100%, ela ainda pode decidir ir por um novo caminho, ou seja, as formigas continuam a buscar rotas alternativas.

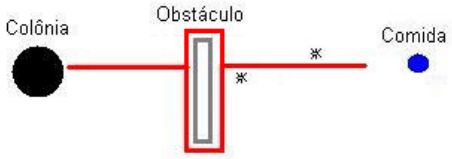
Outro aspecto importante a respeito das formigas é que elas sempre encontram o menor caminho entre a colônia e a fonte de comida. Para entender o porquê desse fenômeno pode-se considerar o seguinte cenário: Uma formiga encontra uma fonte de comida e deixa um trilha de feromônio. O caminho estabelecido pela formiga não é o mais curto, pois ao se deparar com um obstáculo no meio da rota, a formiga, tendo que escolher entre duas possibilidades, aleatoriamente elegeu o caminho mais longo dos dois (Figura 1).

Figura 1 - Formiga Estabelecendo uma Rota



Após retornar a colônia e reforçar a rota, duas novas formigas chegam a colônia e seguem o caminho. Ao chegar ao obstáculo uma das formigas segue a mesma rota da formiga original, mas a segunda decide tentar a outra e ao fazê-lo acaba formando um novo caminho (Figura 2).

Figura 2 - Duas Formigas Estabelecendo Rotas Diferentes



Como o caminho escolhido pela segunda formiga é menor que o da outra, ela retornará antes a colônia (DENNY; WRIGHT; GRIEF, 2001) e a sua trilha terá mais feromônio. Já que essa substância evapora com o tempo, quanto antes a formiga retornar maior a chance de reforçar o caminho e torná-lo mais atraente para outras formigas.

As colônias de formigas apresentam uma série de propriedades desejáveis aos sistemas computacionais. Em primeiro lugar as formigas representam entes distribuídos desprovidos de uma unidade de controle central. Segundo: elas possuem a capacidade de encontrar recursos e estabelecer rotas entre dois pontos determinados. Terceiro: as formigas têm a capacidade não só de encontrar rotas, mas de encontrar as rotas mais curtas, e fazem isso sem utilizar troca de mensagens, mas somente através da modificação do seu ambiente.

2.1.2 Algoritmos ACO

O propósito desta seção é apresentar uma visão geral da literatura sintetizando os algoritmos e soluções computacionais existentes que foram baseados nas características comportamentais das colônias de formigas.

2.1.2.1 AS (ANT-quantity, ANT-quality, ANT-Cycle)

O *Ant System* (DORIGO; COLORNI; MANIEZZO, 1991) foi o primeiro algoritmo *ACO* proposto. Em sua forma mais simples o *Ant System* não compete com outros algoritmos mais modernos (LEMMENS, 2006), porém como serve de base para os outros algoritmos, se faz necessária uma detalhada explicação acerca do seu funcionamento.

No trabalho original os autores propõem o *Ant System* como forma de solucionar o problema do caixeiro viajante e estabelecem as seguintes equações:

$b_i(t)$ (i=1,.., n) representa o número de formigas em uma cidade i no tempo t e m representa o número total de formigas:

$$m = \sum_{i=1}^{n} b_i(t)$$

$T_{ij}(t+1)$ é a intensidade da trilha no caminho $_{ij}$ no tempo t+1:

$T_{ij}(t+1) = \rho T_{ij}(t) + \Delta T_{ij}(t,t+1)$, onde ρ representa o coeficiente de evaporação.

$$\Delta T_{ij}(t,t+1) = \sum_{k=1}^{m} \Delta T^{k}_{ij}(t,t+1),$$

onde $\Delta T^{k}_{ij}(t,t+1)$ representa a quantidade de feromônio depositada pela ^{k}ésima formiga no caminho $_{ij}$ entre o tempo t e t+1.

No algoritmo *Ant System* as formigas não se guiam exclusivamente pela percepção de feromônios, mas também utilizam a visão para determinar distâncias. Assim, η representa a visibilidade e é função da distância entre as duas cidades:

$\eta_{ij} = 1/d_{ij}$

A função da probabilidade da formiga eleger determinado caminho então é:

$$p_{ij}(t) = \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{j=1}^{n} [\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}$$

Onde α e β são parâmetros ajustáveis pelo usuário que representam o peso da proximidade da cidade (visibilidade) e da atratividade da trilha, respectivamente, no cálculo da escolha de qual será a próxima cidade a ser visitada pela formiga.

Para evitar que as formigas escolham um ponto pelo qual elas já passaram, criando assim um *loop*, os autores introduziram uma lista que armazena os pontos já visitados, chamada lista tabu. Assim a fórmula passa a verificar se a escolha está entre as cidades permitidas (*allowed*):

$$p_{ij}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{j \in \textbf{allowed}} [\tau_{ij}(t)]^{\alpha} [\eta_{ij}]^{\beta}} & \text{if } j \in \textbf{allowed} \\ 0 \text{ otherwise} \end{cases}$$

O *Ant System* é na verdade dividido em três algoritmos: o *ANT-density*, *ANT-quantity* e o *ANT-cycle*. O que distingue os três é a escolha sobre como calcular $\Delta T^k_{ij}(t,t+1)$ e quando atualizar $T_{ij}(t)$.

No modelo *ANT-quantity*, a quantidade de feromônio liberada por uma formiga é inversamente proporcional a distância do caminho:

$\Delta T^k_{ij}(t,t+1) = Q1/d_{ij}$

Já no algoritmo *ANT-density* a formiga libera uma quantidade de feromônio Q2 a cada unidade de distância:

$\Delta T^k_{ij}(t,t+1) = Q2$

Tendo sido estabelecidas as diferenças entre os dois, segue o algoritmo comum entre ambos:

1 Inicializa:

Define t:=0 {t é o contador de tempo}

Para cada vértice (i,j) define um valor inicial $\tau_{ij}(t)$ para a intensidade da trilha e $\Delta\tau_{ij}(t,t+1):= 0$

Coloca $b_i(t)$ formigas em cada nó $_i$ {$b_i(t)$ é o número de formigas no nó $_i$ no tempo t}

Define s:=1 {s é a lista tabu}

De i:=1 até n faça

De k:=1 até $b_i(t)$ faça

tabu$_k$(s):=$_i$ {cidade inicial é o primeiro elemento da lista tabu para a formiga k}

2 Repita até a lista tabu estar cheia {esse passo será repetido (n-1) vezes}

2.0 Define s:=s+1

2.1 De i:=1 até n faça {para cada cidade}

De k:=1 até $b_i(t)$ faça {para cada formiga k na cidade I que ainda não se moveu}

Escolha a cidade $_j$ para se mover, com probabilidade $p_{ij}(t)$ dada pela equação (2)

Mova a formiga k para j {essa instrução cria os novos valores $b_j(t+1)$}

Insira nó $_j$ em $tabu_k(s)$

Define $\Delta\tau_{ij}(t,t+1):= \Delta\tau_{ij}(t,t+1)+ Q1$ no caso do modelo Ant-density ou

$\Delta\tau_{ij}(t,t+1):= \Delta\tau_{ij}(t,t+1) +Q2/d_{ij}$ no caso do modelo Ant-quantity

2.2 Para cada vértice(i,j) computa $\tau_{ij}(t+1)$ de acordo com a equação (1)

3 Memoriza o caminho mais curto encontrado até agora

Se (NC<NCMAX) ou (nem todas as formigas escolhem o mesmo caminho) {NC = número de ciclos}

então

Esvazia todas as listas tabus

Define s:=1

De i:=1 até n faça

De k:=1 até bi(t) faça

$tabu_k(s)$:=i {depois de uma volta o formiga k está novamente na posição inicial}

Define t:=t+1

Para cada vértice (i,j) define $\Delta\tau_{ij}(t,t+1):=0$

Vai para o passo 2

Senão

Imprime a menor volta ou Para

O funcionamento do algoritmo é basicamente o seguinte: durante a fase de inicialização (Passo 1, t=0) as "formigas" são posicionadas em diferentes cidades, valores iniciais são atribuídos aos vértices e a lista tabu é inicializada com a cidade na qual a formiga começa.

A segunda etapa (Passo 2) consiste em dois *loops*: o primeiro percorre as cidades, uma a uma, e o segundo seleciona as formigas de cada cidade, novamente, uma por uma, e calcula a cidade para qual a formiga se deslocará. Após o translado da agente, o algoritmo atualiza o valor da intensidade da trilha com a quantidade de feromônio depositado pela formiga. Como estabelecido anteriormente, a quantidade de feromônio liberada pela formiga varia de acordo com o tipo escolhido (*ANT-density* ou *ANT-quantity*).

Após percorrer todas as cidades o algoritmo computa o menor caminho encontrado e verifica se o número de ciclos foi atingido, senão ele esvazia as listas tabus e repete-se o ciclo novamente.

O algoritmo *ANT-Cycle* difere do *ANT-density* e *ANT-quantity* no momento e no cálculo da atualização da intensidade da trilha e da quantidade de feromônio liberada por cada formiga (ΔT^k_{ij}) . Enquanto nos outros dois algoritmos a intensidade é recalculada após cada formiga se deslocar, no *ANT-Cycle* a atualização só é feita após todas haverem se deslocado. Desta forma a quantidade de feromônio é calculada da seguinte forma:

$\Delta T^k_{ij}(t, t+n) = Q3/L^k$, onde L^k é a distância total percorrida pela ^{K}ésima formiga.

E a intensidade da trilha:

$$T_{ij}(t+n) = \rho 1 \cdot T_{ij}(t) \cdot \sum_{k=1}^{m} \Gamma_{ij}(t,t+n). \qquad \{\rho 1 \text{ é diferente de } \rho \text{ já que a equação só é atualizada a}$$

cada n iterações}

Onde $\Delta T_{ij}(t,t+n) = \Delta T^k_{ij}(t,t+n)$.

Os autores obtiveram melhores resultados com o *ANT-Cycle* do que com o *ANT-quantity* e *ANT-density*.

Os autores também utilizaram algumas estratégias diferentes no estudo que inspiraram outros trabalhos, entre elas destacam-se a estratégia elitista, na qual a melhor rota encontrada recebe uma dose extra de feromônio:

eQ3/L*, onde e representa o número de formigas elitistas e L* equivale a distância do menor caminho.

2.1.2.2 ASrank

Entre os estudos inspirados pelo *Ant System*, principalmente no que tange a utilização da estratégia elitista, destacam-se o *ASrank* (BULLNHEIMER; HARTL; STRAUSS, 1999). O conceito do algoritmo é o de classificação (*rank*) das formigas.

Após um ciclo completo, as formigas são ordenadas de acordo com a distância percorrida (*Dp1* <= *Dp2* <= ... <= *DpN*) e a quantidade de feromônio depositada por cada formiga é pesada de acordo com essa classificação. Além disso, somente algumas formigas são consideradas para diminuir a probabilidade de estagnação (descrito na seção 4.1.5).

Nos experimentos realizados pelos autores o *ASrank* apresentou resultados melhores do que seu antecessor.

2.1.2.3 ANT-Q

O *ANT-Q* (GAMBARDELLA; DORIGO, 1995) é mais uma extensão ao *AS*. Proposto por um dos autores do algoritmo original, o diferencial do *ANT-Q* é o fato de implementar aprendizagem por reforço (*Reinforcement Learning*), que utiliza um sistema de recompensa quando um agente percorre uma rota melhor.

2.1.2.4 ACS

O Sistema de Colônia de Formiga (GAMBARDELLA ; DORIGO, 1996) ou *ACS* (do inglês, *Ant Colony System*) foi proposto como melhoria ao AS, mas na verdade difere do *ANT-Q* apenas na forma de atualizar feromônios.

No *ANT-Q* a maneira com a qual era realizado cálculo de atualização das trilhas de feromônio envolvia a utilização de um valor que era uma previsão do valor do próximo estado

(DORIGO; DI CARO; GAMBARDELLA, 1999). Entretanto, os autores descobriram através de experimentos que a substituição deste valor por outro constante resultava no mesmo desempenho, desta maneira, apesar do desempenho ser aproximadamente equivalente, o *ANT-Q* foi abandonado em detrimento do *ACS*, que é mais simples e tão bom quanto.

2.1.2.5 MMAS

Da mesma maneira que os algoritmos anteriores foram todos baseados no *AS*, o *MMAS*, ou *Max-Min Ant System* (STÜTZLE; HOOS, 1997) também foi. A principal diferença entre o *MMAS* e o *AS* é que ele utiliza um intervalo (valor máximo e mínimo) que limita os possíveis valores das trilhas de feromônios.

Para evitar que o algoritmo entre em um estado de estagnação das buscas, que pode acontecer quando uma trilha se aproxima do limite máximo de feromônio enquanto outras estão no mínimo, os autores sugerem a utilização de um mecanismo de suavização de trilhas. O mecanismo se baseia em dois princípios:

1. A intensidade da trilha é aumentada proporcionalmente entre a diferença do limite máximo e a intensidade atual, evitando assim que o valor ultrapasse o limite.

2. O limite mínimo é aumentado durante uma interação e todas as intensidades que forem inferiores ao novo limite serão igualadas a este.

A ideia básica é garantir que apesar de uma trilha ser bastante atraente, as formigas continuarão a buscar novas rotas e o sistema não ficará estagnado.

2.1.2.6 ABC

Ant-Based Control ou *ABC* Schoonderwoerd e Holland (1999) foi o primeiro algoritmo de otimização de colônia de formigas desenvolvido para ser aplicado em roteamento e balanceamento de redes de computadores. De fato, a rede para a qual o ABC foi proposto foi uma rede telefônica na qual uma chamada é estabelecida através da reserva de um circuito virtual.

O algoritmo foi proposto com a intenção de servir como ferramenta de balanceamento de carga, distribuindo a chamada entre diversos *switches*, para minimizar o número de

chamadas que não podem ser servidas devido à existência de congestionamento (FAROOQ; DI CARO, 2008).

Uma das características do *ABC* é que ele presume que os enlaces entre os nós são simétricos, ou seja, um determinado caminho fim-a-fim possui o mesmo custo em ambas as direções.

O *ABC* também possui outras peculiaridades, como o envelhecimento, atraso e barulho.

O envelhecimento garante que formigas que já estão há mais tempo no sistema liberam menos feromônio, garantindo que a influência de um agente que passa sobre um caminho congestionado e/ou um mais longo será menor que um que passe por um caminho desobstruído e/ou menor.

O atraso é imposto sobre formigas de maneira proporcional a capacidade livre do nó, o que associado ao envelhecimento acarreta redução do impacto dos agentes sobre o congestionamento.

Já o barulho é o elemento que garante que as formigas podem, por vezes, ignorar as trilhas de feromônio e seguirem por um caminho aleatório, o que é interessante para redes dinâmicas.

O algoritmo *ABC* pressupõe enlaces com custos simétricos, assim, formigas que partem de uma origem nó S ao chegarem ao nó intermediário I, atualizam as entradas na tabela de roteamento de I que têm como destino S. Dessa maneira fica claro que o algoritmo *ABC* não é adequado a redes com enlaces assimétricos. Para isso foi proposto um novo algoritmo, o *AntNet*.

2.1.2.7 AntNet (AntNet-CL, AntNet-CO)

O algoritmo *AntNet* Di Caro e Dorigo, (1998) foi desenhado para redes assimétricas comutadas por pacotes e o seu objetivo primário é maximizar a desempenho da rede como um todo. O algoritmo implicitamente alcança o balanceamento de carga através da distribuição probabilística de pacotes em múltiplos caminhos (WEDDE; FAROOQ, 2006).

O algoritmo *AntNet* é similar ao *AS*, a principal diferença é que ele define dois tipos diferentes de agentes: a formiga de ida, $F_{s \to D}$, (*Forward Ant*) e a formiga de volta, $B_{D \to S}$, (B*ackward Ant*); que viajam de um nó origem *S* até um nó destino *D*. As formigas de ida possuem uma pilha na qual armazenam a rota pela qual passam e os tempos entre nós. Os tempos são calculados pela diferença entre o momento em que a formiga foi enviada de um nó e o momento em que ela chega no próximo, o que claramente sugere a importância da sincronização de relógios entre os nós.

As $F_{s \to D}$ são liberadas de acordo com o tráfego gerado em *S* com destino a *D*, de acordo com a equação abaixo:

$$p_d = \frac{f_{sd}}{\sum_{d'=1}^{N} f_{sd'}}$$

Onde P_d corresponde à probabilidade de se criar no nó *S* uma formiga com destino *D* e f_{sd} representa a quantidade de bits gerados em *S* com destino a *D*.

Quando a formiga de ida chega ao seu destino final, uma formiga de volta é criada e enviada pelo caminho contrário da formiga original, ou seja, $B_{D \to s}$. Por cada nó pelo qual a formiga de volta passa ela atualiza as tabelas de roteamento (as probabilidades de escolher determinados nós) de acordo com os tempos registrados pela formiga de ida. As entradas da tabela que são atualizados são aquelas correspondentes a todos os nós no caminho feito pela formiga de ida.

Enquanto as formigas de ida utilizam as filas normais que representam o tráfego real da rede, as formigas de volta utilizam filas prioritárias.

Posteriormente, os autores introduziram uma variante do *AntNet* voltada a redes orientadas a conexão (DORIGO; DI CARO, 1998), que chamaram de *AntNet-CO* (*Connection-Oriented*) e o *AntNet* original recebeu a alcunha de *AntNet-CL* (*Connectionless*). São três as diferenças principais entre os dois algoritmos: Em primeiro lugar, no CO as formigas de ida também usam filas de alta prioridade. Segundo, as formigas de ida não carregam os tempos da viagem na pilha. E por último, como as formigas de volta não têm os tempos armazenados pela formigas de ida, para atualizar as tabelas de roteamento elas apenas utilizam estimativas dos tempos.

2.1.2.8 "AntNet 1.1"

Uma versão modificada do *AntNet*, sugestivamente chamada de *AntNet* 1.1, foi proposta por autores diferentes (BARÁN; SOSA, 2000). A intenção dos pesquisadores paraguaios foi de sugerir alterações que pudessem potencialmente melhorar o desempenho do *AntNet*. Abaixo segue um resumo das alterações propostas que, de acordo com os autores, mais contribuiu para o aumento do desempenho :

1. Inicialização Inteligente das Tabelas de Roteamento: Enquanto o algoritmo *AntNet* não especifica a maneira como a tabela de roteamento será inicializada, o *AntNet* 1.1 sugere uma inicialização que faça uso de conhecimento prévio da topologia da rede, atribuindo probabilidades maiores para determinados nós, que podem ser ao mesmo tempo vizinhos e destinos.

2. Método Dual de Escolha do Próximo Nó: Os algoritmos anteriores têm em comum o fato de utilizarem determinadas probabilidades na determinação da escolha do próximo nó. A AntNet 1.1 propõe que duas abordagens sejam utilizadas, com probabilidade de 0,5 entre elas. A primeira envolveria a mesma forma de calcular a probabilidade dos algoritmos anteriores, já a segunda utilizaria uma maneira determinística de dividir os pacotes entre os caminhos disponíveis.

2.1.2.9 AntHocNet

Alguns algoritmos na literatura foram propostos com a intenção de adaptar os conceitos dos algoritmos de otimização de colônia de formigas para o roteamento em redes *MANET* (*mobile ad hoc wireless networks*). O *ARA*, do inglês *Ant-Colony Based Routing Algorithm* (GÜNES; SORGES; BOUAZIZI, 2002), é considerado o primeiro algoritmo MANET realmente inspirado nos conceitos de *ACO* e não somente no nome (MEISEL, PAPPAS, ZHANG, 2010).

Apesar da senioridade do *ARA*, apenas *o AntHocNet* (DI CARO; DUCATELLE; GAMBARDELLA, 2005) representará os algoritmos *MANET* neste estudo, pois este, além de ser mais recente, incorpora algumas características interessantes ao problema ao qual este trabalho investiga.

O *AntHocNet* é considerado um algoritmo hibrido, pois ele possui comportamentos reativos e proativos. Ele é dito reativo pois as informações de roteamento acerca de um determinado destino só são coletadas a partir do momento em que existe a necessidade de estabelecimento de um canal de comunicação entre dois pontos (FAROOQ; DI CARO, 2008). Ele também é proativo porque enquanto dados estão sendo transferidos, os caminhos são monitorados e melhorados proativamente. Isso é possível através da liberação de um agente (*forward_ant*) a cada *N* pacotes de dados. O agente pode ser enviado por *unicast* ou *broadcast*, com uma probabilidade de 0,9 e 0,1 respectivamente em cada nó do caminho. *Unicast* é utilizado para percorrer rotas conhecidas e *broadcast* para descobrir novas rotas, o que significa dizer que caso a formiga chegue até o destino somente utilizando *unicasts* ela terá trafegado por um caminho já conhecido e sua jornada servirá como forma de avaliar a qualidade atual desta rota; agora caso a formiga tenha sofrido *broadcast* em algum nó, isso significará que ela abandonará as rotas conhecidas e explorará uma nova. Os autores impõem um limite à quantidade de saltos (*hops*) que uma formiga pode dar, garantindo que os caminhos descobertos sejam melhorias ou alternativas ao caminho original.

O *AntHocNet* também considera a hipótese de falhas em enlaces da rede. A utilização de mensagens *hello* funcionam na manutenção das conexões entre os nós. A ausência do recebimento de tal mensagem em um determinado espaço de tempo indica a falha do nó vizinho. Quando um nó detecta a falha de outro ele envia um mensagem de *broadcast* que contem uma lista dos destinos para os quais o nó perdeu o seu melhor caminho, além do novo melhor tempo e quantidade de saltos associados ao que seria o caminho alternativo (DI CARO; DUCATELLE; GAMBARDELLA, 2005).

O *AntHocNet* foi avaliado em simulações e considerado como o mais eficiente algoritmo *ACO* para redes wireless (UCHHULA; BHATT, 2010). Os autores atribuem esse resultado ao fato do algoritmo possuir uma probabilidade maior de explorar novos caminhos, apesar de utilizar um número maior de agentes do que outros algoritmos baseados em formigas. Ainda que gere uma quantidade maior de tráfego de controle do que outros algoritmos do gênero o *AntHocNet* é mais eficiente em manter rotas e tem uma probabilidade maior de explorar novos caminhos.

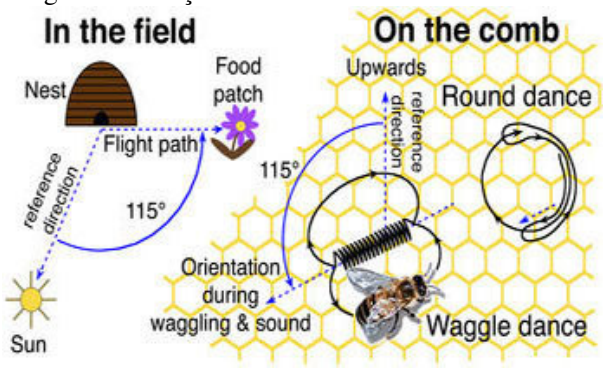
2.2 OTIMIZAÇÃO DE COLÔNIA DE ABELHAS - BEE COLONY OPTIMIZATION

2.2.1 Conceito

Diferentemente das formigas, que se comunicam através da liberação de feromônios, as abelhas não utilizam feromônios. No lugar de liberar tais substâncias elas adotam um mecanismo mais direto e complexo de comunicação, a dança.

Da mesma forma que a formiga sai em uma direção supostamente aleatória em busca de uma fonte de comida, a abelha também assim o faz, porém, ao encontrá-la, ela regressa diretamente a colmeia e comunica as outras a distância e direção da comida. Isso é feito através de uma movimentação que se assemelha a uma dança. A abelha se posiciona de tal forma que o ângulo formado entre a posição do seu corpo na colmeia e o sol informe as outras a direção na qual se encontra a fonte de comida e a duração da dança revela a distância (Figura 3).

Figura 3 - Dança das Abelhas



Fonte: Nagpal (2007)

Graças à rotação da terra em torno do sol e do seu próprio eixo, da perspectiva da abelha o sol se move no céu. Essas incríveis criaturas são capazes de compensar essa movimentação ajustando a angulação da dança de acordo.

Porém uma questão se faz imperativa: Através da observação da dança, uma abelha descobre a localização da fonte de comida, mas como faz para navegar até lá se o caminho não tem feromônios? A resposta é que as abelhas calculam um vetor de integração do caminho (*Path Integration Vector, PI*) que representa a combinação de todas as distâncias e ângulos percorridos ao longo da rota (LAMBRINOS et al, 2000). Para construir um vetor de integração de caminho o inseto utiliza uma computação aproximada, no lugar de realizar a

soma de vetores como um ser humano faria (MÜLLER, M.; WEHNE. 1988). Essa computação aproximada nada mais é do que a média aritmética de todos os ângulos formados pelo trajeto, na qual o peso de cada ângulo é um reflexo da distância que o agente percorreu naquela direção.

Na realidade, o comportamento da colônia de abelhas ainda é pouco compreendido pela ciência. Sabe-se, por exemplo, que as abelhas provam uma amostra do pólen trazido pelas abelhas "dançarinas" e que elas decidem se irão seguir a rota ou não baseado em fatores tais como a qualidade dessa amostra e a distância indicada pela dança, porém ainda não se sabe ao certo que peso tem cada fator no cálculo da decisão final.

2.2.2 Algoritmos BCO

Apesar do mistério que ainda permeia boa parte do sistema biológico que a colônia de abelhas representa, os pesquisadores já descobriram o suficiente para permitir que ele seja modelado e utilizado como algoritmo na resolução de problemas complexos. Esta seção oferece uma visão de alto nível de alguns dos principais algoritmos baseados em colônias de abelhas dispostos na literatura.

2.2.2.1 BS

O *BS* Lučić e Teodorović (2003), do inglês *Bee System*, foi proposto com uma orientação diferente a do *Ant System*, porém com um objetivo similar. Enquanto o *AS* faz uso dos conceitos de liberação de feromônios, pelos quais são conhecidos os algoritmos de otimização de colônia de formigas, o BS modela o comportamento da colônia de abelhas, que se comunicam de forma diferente, i.e., não dependem da liberação de feromônios. O que os dois algoritmos têm em comum é que ambos foram sugeridos como forma de resolução da mesma questão: o problema do caixeiro viajante (*TSP*).

No algoritmo *BS*, a cada iteração, a colônia é posicionada aleatoriamente em algum dos nós do grafo. Uma iteração do algoritmo é divida em estágios. Em cada estágio a abelha visitará N nós e depois retornará a colônia. Ao retornar, a abelha decide (probabilisticamente) se:

a) Abandonará a fonte de comida e se tornará uma seguidora, que ficará na colmeia esperando alguma “dançarina”.

b) Continuará a forragear, i.e., buscar alimento.

c) Dançará para recrutar mais abelhas antes de retornar à fonte de comida.

As abelhas iniciam a exploração de forma progressiva, de acordo com uma variável especificada pelo usuário que determina a probabilidade de cada abelha decidir começar a forragear. Dessa maneira, se for usado um valor alto, praticamente todas as abelhas sairão em busca de alimento logo no início. Em contraste, com um valor reduzido, o aumento na quantidade de abelhas exploradoras será mais lento.

A quantidade de “néctar” que pode ser coletada por uma abelha é inversamente proporcional à distância entre o nó e a colônia, o que faz com que as abelhas deem preferência a caminhos mais curtos. A importância do atributo distância na escolha das abelhas é proporcional à quantidade de iterações realizadas. Logo no início as abelhas têm mais liberdade para buscar novos caminhos e a distância afeta pouco as suas decisões. Com o passar do tempo e das iterações, o peso da distância no cálculo do nó que a abelha escolherá aumenta, o que a leva a dar preferência a caminhos mais curtos.

As agentes abelhas decidem qual nó irão explorar de forma aleatória e também possuem memória e armazenam a quantidade de abelhas que visitaram um determinado nó ao longo de I iterações. Quanto maior for esse valor, maior será a probabilidade de uma nova abelha selecionar aquele caminho no futuro.

Assim, são dois os valores que mais influenciam a escolha de uma abelha: a distância entre os nós (chamada simbolicamente como quantidade de néctar) e a quantidade de abelhas que utilizaram aquele caminho.

Os resultados dos experimentos (particularmente os tempos exigidos para encontrar a melhor solução) demonstram que o BS é um algoritmo com grande potencial. Porém os autores pecam em não realizar comparações com outros algoritmos como, por exemplo, o *Ant System*.

Algumas questões interessantes que os autores levantam na conclusão do estudo e que poderiam motivar pesquisas na área:

a) As abelhas deveriam ser todas iguais ou haveria algum ganho em ter tipos de agentes diferentes? Este ponto é explorado no algoritmo *BeeAdHoc* discutido na seção 3.2.3.
b) Seria possível utilizar combinações hibridas de algoritmos de inteligência de enxame ou de outras áreas?

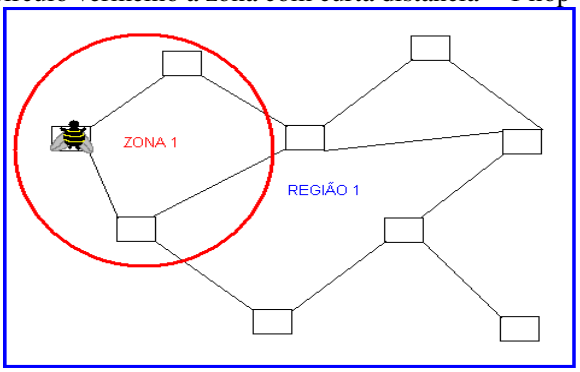
2.2.2.2 BeeHive

Apesar do *BS* ter sido o primeiro algoritmo de colônia de abelhas que propôs uma solução que pudesse ser transposta ao mundo da informática, o primeiro algoritmo de colônia de abelhas verdadeiramente orientado a abelhas e a computação foi o *BeeHive* (WEDDE; FAROOQ; ZHANG, 2004).

O *BeeHive* faz uso de dois tipos de agentes: abelhas de curta e de longa distância. Como o nome sugere, a diferença entre os dois tipo de abelha é a distância que elas podem percorrer. Enquanto as abelhas de curta distância são limitadas a percorrer uma quantidade pequena de *hops* contados a partir do nó de origem, as abelhas de longa distância podem percorrer uma distância maior.

No algoritmo *BeeHive* a rede é dividida de forma hierárquica em dois tipos de área: zonas e regiões (figura 4). Neste contexto, uma zona é a área que abelhas de curta distância podem alcançar a partir de um determinado nó.

Figura 4 - Divisão Hierárquica. O quadrado azul representa a região e o círculo vermelho a zona com curta distância = 1 hop



O mesmo nó pode integrar diversas zonas pertencentes a outros nós. Uma rede também pode ser dividida em regiões, que nada mais são do que *clusters* de zonas. Cada região possui um nó dito representante que é escolhido entre os nós da região, aquele com o menor endereço IP.

No início do algoritmo, quando as regiões ainda não foram estabelecidas, todos os nós se candidatam a posição de nó representante. Para isso eles enviam abelhas de curta distância a todos os seus vizinhos. Se um nó recebe uma abelha de um nó com endereço IP menor que o seu ela desiste e se associa a essa outra região. Os nós continuam a enviar abelhas de curta distancia até que a rede esteja divida em regiões distintas e logo depois comunicam a todos os outros nós a qual região ele pertence.

Cada nó mantém informação de roteamento acerca de todos os outros nós da sua zona a também de todos os nós representantes da rede. Caso precise enviar algo para um nó que esteja fora da sua zona, ou seja, não conste da sua tabela de roteamento, ele enviará no caminho que leve ao nó representante.

De fato, cada nó possui três tabelas de roteamento. A primeira é a tabela que contem os nós pertencentes à mesma zona. Cada entrada na tabela é uma função do atraso de enfileiramento aliado ao atraso propagação ao qual um pacote é submetido trafegando entre esses dois nós. Assim pode-se dizer que a atratividade de uma rota está associada ao tráfego da mesma. A segunda tabela é a que contem as entradas dos nós representantes das regiões e têm os mesmo tipos de entrada (atraso de enfileiramento, etc.) da tabela da zona. A terceira e última tabela é a que contem as correspondências entre destinos conhecidos e regiões. O propósito dessa divisão hierárquica é reduzir o tamanho da tabela de roteamento que em outros algoritmos do gênero possuem uma entrada para cada nó.

Durante a execução do algoritmo os nós de zona constantemente enviam abelhas de curta distância para seus vizinhos através de *broadcast*. Os nós representantes só enviam abelhas de longa distância.

Quando uma abelha chega a um determinado nó, a tabela de roteamento deste nó é atualizada com a informação carregada pela abelha e então, caso o limite de *hops* ainda não tenha sido atingido, a reenvia para os seus vizinhos, com a exceção do nó do qual a abelha chegou.

A avaliação do algoritmo foi realizada através de experimentos implementados no simulador OMNeT++ (VARGA, 1999) e que compararam o *BeeHive* com alguns outros protocolos de roteamento: o *AntNet, o OSPF* e um algoritmo genético chamado *DGA* (LIANG; ZINCIR-HEYWOOD; HEYWOOD, 2002).

O propósito do experimento foi estudar a desempenho dos algoritmos em um cenário de tráfego crescente. De acordo com os autores, os resultados dos experimentos revelaram índices superiores ao *AntNet* em termos de desempenho, como por exemplo, valores inferiores de atraso por pacote. Além do ganho de desempenho o algoritmo também proporciona uma diminuição do tamanho das tabelas de roteamento em relação ao algoritmo *AntNet*, porém as mesmas ainda são um pouco maiores que as tabelas do *OSPF*.

2.2.2.3 BeeAdHoc

O *BeeAdHoc* Wedde; et al (2005) foi o primeiro algoritmo desenvolvido cujo foco é a aplicação dos conceitos de colônia de abelhas orientados a redes *MANET*. O propósito principal do algoritmo é utilizar a energia, um recurso limitado nas redes *MANET*, de forma eficiente.

Uma das características do algoritmo é utilizar tipos diferentes de agentes. Essa é uma das formas com a qual a literatura distingue entre tipos de algoritmos de roteamento de colônias de insetos. A classificação do *BeeAdHoc*, no que tange esse aspecto em particular, é dita *Multi-Agente*.

São quatro os tipos de agentes presentes no algoritmo: empacotadores, batedores, forrageiros e enxames. Diferentemente do *BS*, no qual as abelhas eram todas iguais e a única coisa que variava eram o estado no qual elas podiam estar (forrageando, aguardando na colônia, etc..) e do *BeeHive*, no qual a única diferença é a distância que elas percorrem; o *BeeAdHoc* define quatro tipos distintos de agentes, com o intuito de realmente modelar o comportamento de uma colônia de abelhas biológica.

As abelhas empacotadoras residem sempre dentre do nó e sua função é receber e armazenar segmentos da camada de transporte e buscar abelhas forrageiras que possam transportar os pacotes. Logo após entregarem os pacotes para as abelhas forrageiras, as empacotadoras morrem.

As abelhas forrageiras são as que efetivamente carregam os pacotes de dados recebidos das empacotadoras. O algoritmo define dois tipos de abelhas forrageiras: de atraso ou de vida. As forrageiras de atraso coletam dados acerca dos atraso fim-a-fim, enquanto as de vida compilam informações sobre a capacidade de energia restante nos nós. Após chegarem ao nó de destino as abelhas forrageiras aguardam tráfego no sentido contrário para voltar ao nó de origem (*piggyback).* Caso seja utilizado o protocolo *TCP*, as forrageiras pegam carona nas confirmações (*ACK's*).

Os pesquisadores criaram um tipo de abelha específica para solucionar o problema que pode ocorrer em uma rede que utiliza um protocolo de transporte não-confiável sem confirmações (*UDP*) nas quais as forrageiras possam pegar carona, o enxame. A função deste tipo de agente é garantir que uma determinada proporção possa ser mantida entre o número de forrageiras esperando retorno e o número de forrageiras chegando. Assim o enxame é uma forma de reunir um número de forrageiras esperando *piggyback* e mandá-las todas de uma só vez, em um único pacote.

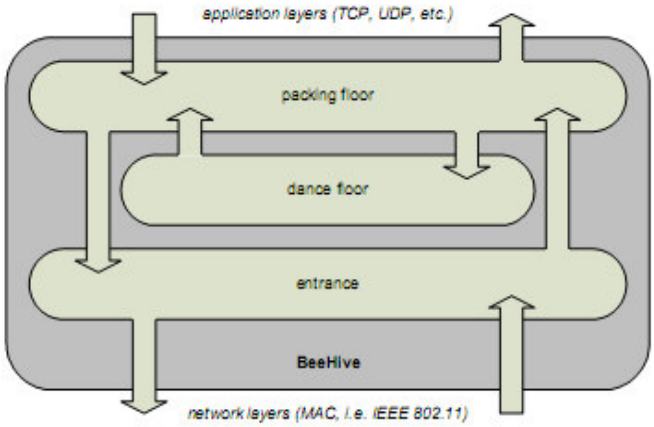
Quando as abelhas forrageiras retornam ao nó de origem elas realizam a "dança" para recrutar outras forrageiras. O recrutamento é baseado na qualidade do caminho, que pode ser o atraso fim-a-fim ou a bateria disponível, a depender do tipo da abelha. Isso significa que o algoritmo tem dois propósitos básicos, procurar caminhos com menor congestionamento e mais bateria disponível.

Além disto, as abelhas forrageiras têm um "prazo de validade", o que faz com que as forrageiras mais novas, correspondentes a rotas mais novas, sejam mais valorizadas.

E por último, as abelhas batedoras têm a função de buscar novos caminhos entre um nó de origem e um nó destino. Assim elas são enviadas através de *broadcast* para todos os vizinhos do nó. Cada agente é identificado através de um número *id* e do endereço de origem. O nó destino envia todas as batedoras de volta pelo mesmo caminho que elas vieram. Após retornarem ao nó de origem os agentes batedores realizam a "dança" para recrutar forrageiras.

No *BeeAdHoc* cada nó representa uma colônia que é divida em três níveis (figura 5).

Figura 5 - Divisão da Colônia em Níveis



Fonte: Wedde et al (2005).

O primeiro nível da colônia é a entrada, responsável por receber os quadros da camada de enlace. A sua função depende do tipo de agente que ela receber. Caso seja uma abelha forrageira e o destino seja a própria colônia, ela é encaminhada para o nível de empacotamento. Caso a forrageira seja destinada a outro nó, ela é enviada diretamente através da camada de enlace para o próximo nó.

Na hipótese de chegar uma abelha do tipo batedora que tiver um destino que não seja a colônia, ela é transmitida através de *broadcast* para os nós vizinhos. A identificação da abelha batedora é mantida em caso de chegar uma réplica, circunstância na qual ela é descartada.

O segundo nível da colônia é o nível de empacotamento, que como estabelecido anteriormente, é a interface entre a camada de transportes e o resto da colônia. É nesse nível que as abelhas empacotadoras realizam a tarefa de receber o segmento e passar o pacote para um agente forrageiro no nível de dança. Caso não haja um forrageiro disponível o pacote é mantido em um *buffer* até que uma retorne. Caso isso não aconteça uma abelha batedora é enviada para encontrar um caminho até o destino do pacote.

O último nível da colônia é o nível de dança, considerado pelos autores como o coração da colônia, pois é onde são mantidas as abelhas forrageiras, que efetivamente mantêm as informações de roteamento. É neste nível que as abelhas recrutam outras. Então é aqui que as abelhas forrageiras são enviadas para entregar os pacotes das abelhas empacotadoras e também onde uma forrageira recruta a outra.

Os experimentos realizados pelos autores objetivaram analisar o desempenho do algoritmo em comparação a outros algoritmos de roteamento (AODV: Ad-Hoc On-Demand Distance Vector Routing, DSR: Dynamic Source Routing, DSDV: Dynamic Destination-Sequenced Distance-Vector). Os resultados das simulações sugere que o *BeeAdHoc* tem um desempenho superior ou equivalente, mas com um gasto de bateria inferior.

2.2.2.4 BeeHave

O *BeeHave* Lemmens; et al (2008) é uma combinação de dois algoritmos e de uma plataforma de simulação inspirada em colônia de abelhas, além disso ela é uma das poucas soluções que utilizam o conceito de vetor de integração de caminho, utilizando um vetor para o destino (fonte de comida) e outro para a origem (a colônia).

O primeiro algoritmo é composto por duas funções básicas: GerenciarAtividadeDaAbelha() e CalcularVetores(). A primeira função lida com a atividade da abelha referente ao seu estado atual. Cada abelha pode estar em qualquer um de seis estados:

1) EmCasa (*AtHome)* – O agente se encontra na colônia em processo de determinação do próximo estado.
2) FiqueEmCasa (*StayAtHome*) – O agente se encontra na colônia, porém só sairá quando poder aproveitar o conhecimento de outro agente.
3) Aproveitamento (*Exploitation*) – Estado no qual o agente se aproveita da experiência de outras abelhas.
4) Exploração (*Exploration*) – Agente está explorando o mundo.
5) IrParaCasa (*HeadHome*) – Agente decide retornar para a colônia.
6) CarregandoComida (*CarryingFood*) – Agente está carregando comida de volta para colônia.

O segundo algoritmo define as condições sob as quais são realizadas as transições entre os seis estados. Abaixo segue o pseudocódigo (autoexplicativo) dos autores:

```
1: if State is StayAtHome then
2:     if Vector exists then
3:         Exploitation
4:     end if
5: else if Agent not AtHome then
6:     If Agent has food then
7:         CarryingFood
8:     else if Depending on chance then
9:         HeadHome, Exploration or Exploitation
10:    end if
11: else if Exploit preference AND state is AtHome then
12:    if Vector exists then
13:        Exploitation
14:    else
15:        Exploration
16:    end if
17: else if StayAtHome preference AND state is AtHome then
18:    if Vector exists then
19:        Exploitation
20:    else
21:        StayAtHome
22:    end if
```

23: else

24: Exploration

25: end if

Na perspectiva desta presente estudo, o *BeeHave* se encaixa na categoria de algoritmos que buscam realizar uma modelagem fiel à biologia, mas que nem sempre se adaptam ao cenário computacional em questão. Na natureza, o vetor de integração de caminho representa uma rota direta entre a colônia e a fonte de comida. Nas redes de computadores esse caminho é por vezes tortuoso e a analogia nem sempre é apropriada. Como os vetores são essenciais ao algoritmo *BeeHave,* isso inviabiliza a sua utilização como modelo de solução ao problema proposto por esse artigo. Assim, sua função aqui é meramente ilustrativa.

2.3 ALGORITMO HÍBRIDO EXISTENTE: ANT-BEE ROUTING

O AntBee Path proposto neste estudo não é o primeiro algoritmo híbrido baseado em colônias de formigas e abelhas a ser sugerido no mundo. Essa honra provavelmente deve ser atribuída ao Ant-Bee Routing (RAHMATIZADEH, 2009).

O Ant-Bee Routing é um algoritmo híbrido ACO/BCA que difere do AntBee Path na forma com a qual o modelo biológico foi concebido e também em relação a algumas funcionalidades implementadas.

Primeiramente, o AntBeePath busca menores caminhos a partir de um nó central (a colônia) para todos os outros nós de uma determinada topologia, semelhante ao mecanismo de descobrimento de rotas em alguns protocolos de roteamento. O objetivo é basicamente determinar um caminho que possa ser usado, por exemplo, para roteamento.

Já o Ant-Bee Routing parte de uma abordagem distribuída e lança agentes de cada nó para nós destino, selecionados de acordo com o tráfego, o que, em essência, quebra a analogia biológica. Enquanto o AntBee Path busca encontrar os menores caminhos, o Ant-Bee Routing objetiva a construção de tabelas de roteamento em cada nó.

Ademais, os agentes do Ant-Bee Routing não são verdadeiramente híbridos. De fato, quando um agente formiga alcança seu destino ele morre e um agente abelha é gerado. No

algoritmo AntBee Path o agente é híbrido desde a sua geração, ela combina o comportamento das duas espécies.

2.4 OUTRAS LINHAS DE PESQUISA BIO-INSPIRADA

A natureza é composta de inúmeros sistemas biológicos, porém, somente uma fração destes foram estudados a partir de uma perspectiva computacional. O propósito desta seção é ilustrar de forma sucinta (breve e não exaustiva) algumas das outras áreas de pesquisa relacionada à computação bio-inspirada:

1) **Algoritmos Evolucionários**:

   a. **Algoritmos Genéticos** mencionado na introdução desta dissertação (HOLLAND, 1975).

   b. **Programação Genética.** Abordagem diferente das áreas similares de Algoritmos genéticos, programação evolucionária e estratégias evolucionárias. A diferença reside no fato de que na programação genética a complexidade da tarefa e a complexidade da estrutura de dados são variáveis através do curso da evolução (TIMMIS, 2005).

   c. **Hardware Evolutivo**. Hardware que pode mudar a sua arquitetura e comportamento de forma dinâmica e autônoma somente através de interações com seu ambiente.

2) **Sistemas Imunes Artificiais**: Modelado no sistema imune dos vertebrados, a ideia é que o sistema seja capaz de desenvolver anticorpos na presença de vírus ou *worms* e que possam reconhecê-los no futuro e ao mesmo tempo minimize a possibilidade de respostas auto-imune (KEPHART, 1994).

3) **Sistema Circulatório Artificial**: Estudo propõe maneira de dispor uma rede de sensores baseado no sistema circulatório mamífero (PAPPAS et al 2009).

4) **Protocolos de Fofoca – *Gossip***: Distribuição de dados replicados em rede. Mensagens são propagadas de nó a nó até que todos em um grupo tenham recebido a mensagem (WOKOMA, 2002). Este estudo em particular,

combinou protocolos de fofoca com o método de sincronização de vagalumes para sincronizar o tempo no qual as “fofocas” são liberadas.

5) **Métodos de Sincronização de Vagalume**: Estudo se propôs a resolver o desafio de sincronização dos vagalumes através da construção autômatos celulares bidimensionais (TEUSHER et al 2003).

3 ALGORITMO ANTBEEPATH

3.1 DESCRIÇÃO

O propósito desta seção é introduzir o *AntBeePath*, um algoritmo capaz de combinar o comportamento das duas espécies biológicas apresentadas anteriormente, a formiga e a abelha, e aplicá-lo na resolução do problema de determinação de rotas.

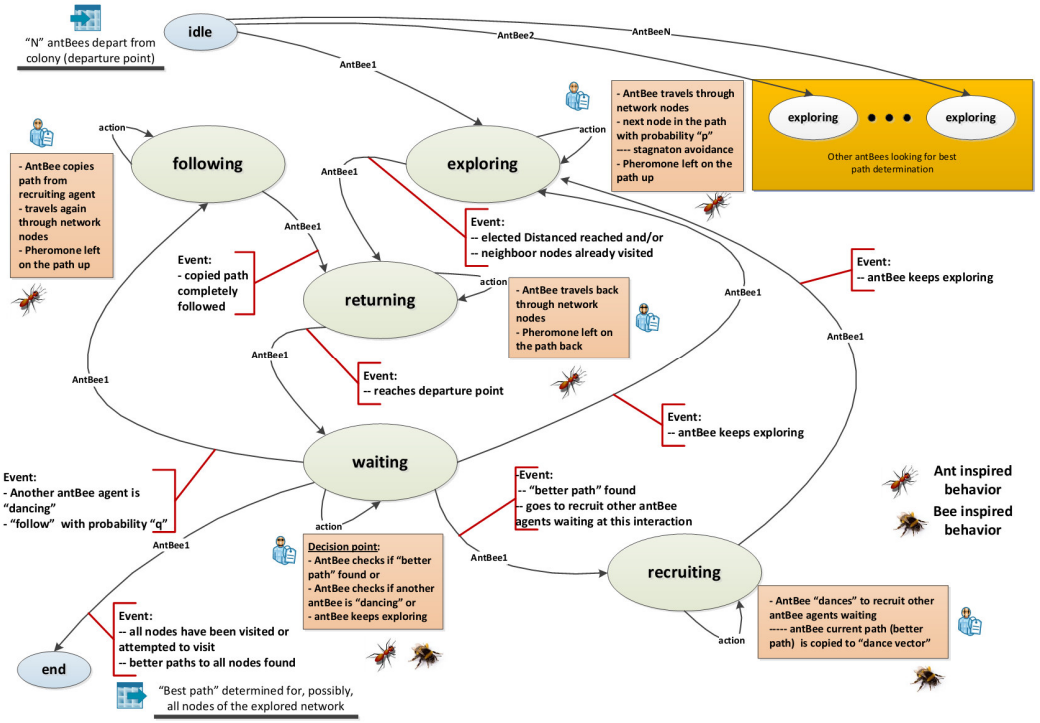
A premissa básica do algoritmo consiste na simulação de uma colônia de agentes híbridos que combinam as características de ambas as espécies: A capacidade das formigas de se comunicarem através da liberação de ferômonio (stigmergia) e a habilidade de dança das abelhas, nas quais elas comunicam a seus pares a rota que acabaram de descobrir.

Neste estudo, os agentes híbridos foram sugestivamente chamados de *antBees*. Um agente a*ntBee* é composto de um estado, que descreve a situação na qual o agente se encontra em um dado momento e uma lista de nós que ele visita. Um agente também tem outras variáveis associadas a ele que armazenam informações tais como a distância trafegada pelo agente (*distanceCovered* – o número de nós que o agente visitou) e a distância escolhida (e*lectedDistance* – um número pseudorrandômico gerado para cada agente e que especifica a distância máxima que o agente pode percorrer).

3.2 ESTADOS

No algoritmo AntBeePath os agentes navegam através de um conjunto de estados tal como ilustrado na figura 6, abaixo. No momento de inicialização, cada agente antBee deixa a colônia (ponto de partida) e começa a explorar a rede. Ao longo da execução do algoritmo um agente poderá estar em qualquer um dos seguintes estados: Explorando (exploring), Retornando (returning), Esperando (waiting), Recrutando (recruiting) e Seguindo (following), determinado de acordo com a lógica do algoritmo.

Figura 6 - Síntese dos Estados do Algoritmo



Explorando (Exploring): Representa o estado no qual os agentes AntBees exploram a rede. O agente antBee primeiro busca todos os nós que são vizinhos (adjacentes) ao seu nó atual. Neste momento o agente cria um vetor no qual ele armazena todos os nós alcançáveis.

Em seguida é gerado de forma pseudoaleatória um número entre 0 e 100 que será usado para determinar o próximo nó a ser visitado. O agente então varre a lista de nós alcançáveis, um nó de cada vez, aplicando uma equação ponderada (descrita na seção 4.1.5) para determinar a probabilidade de escolher o próximo nó que ele irá visitar. Essa probabilidade é função da quantidade de feromônio contida no enlace entre o nó atual e o respectivo nó testado. Caso o valor gerado de forma pseudoaleatória seja menor ou igual à probabilidade calculada, o nó em questão será eleito como próximo nó a ser visitado e o agente se moverá para ele, liberando feromônio no enlace entre o nó em que estava e este novo nó. Caso o valor aleatório seja superior, um novo valor pseudoaleatório será calculado e o processo será repetido para o próximo nó do vetor de nós alcançáveis. O processo continuará até que um dos nós seja escolhido. Caso o vetor de nós alcançáveis chegue ao fim sem que um deles tenha sido escolhido, o processo se reinicia da primeira posição do vetor e persiste até que um seja escolhido.

A equação que determina a probabilidade de escolher o próximo nó desse conjunto, ($P(n)$, não segue uma relação 1 pra 1 em termos de quantidade de feromônio, para evitar o fenômeno conhecido com estagnação, no qual os agentes deixam de buscar novas rotas na presença de um caminho dominante. A relação entre a probabilidade e feromônio e o fenômeno da estagnação é explicado na seção 4.1.5.

Durante o estado Explorando, cada antBee só pode se mover um nó por iteração e libera uma unidade de ferômonio no link correspondente entre o nó atual e o próximo nó. O agente continua no estado explorando até que pelo menos um dos dois eventos ocorra: ou o agente chega a um ponto no qual todos os nós alcançáveis a partir do seu nó atual já foram visitados ou a distância escolhida é igual à distância percorrida, o que significa que o agente trafegou a distância máxima permitida. Neste ponto o estado do agente muda para Retornando.

Retornando (Returning): Nesse estado o agente antBee retorna para a colônia invertendo o caminho (armazenado no vetor de Nós Visitados) pelo qual ele trafegou durante o estado Explorando. Mais uma vez, se movendo um nó por iteração, o agente libera ferômonio no enlace entre os nós. O agente antBee continua neste estado até que ele finalmente retorne à colônia. Nesse ponto, o agente muda seu estado para Esperando.

Esperando (Waiting): Estado no qual um agente antBee se encontra após retornar à colônia. Nesse ponto existem três possibilidades:

1) O agente compara a rota que fez até o nó final Nf . Caso o agente antBee descubra que este caminho é melhor (mais curta) que a rota armazenada na tabela da colônia ele muda seu estado para “Recrutando”. Caso ele perceba que a sua rota é inferior (mais longa) ou igual à conhecida pela colônia ele vai para o item 2.

2) Caso haja outro agente antBee dançando (comportamento da abelha) ele tem uma probabilidade $P(q)$ de ser recrutado por este agente e proceder para o estado “Seguindo”. Se não houver agente dançando ou caso haja mas ele não seja recrutado ele vai para o item 3.

3) Caso as duas hipóteses acima não se confirmem o agente reinicia a exploração ao retornar para o estado “Explorando.”

Aqui se faz necessária uma observação: O algoritmo não contempla a possibilidade de vários agentes estarem dançando ao mesmo tempo. Caso um agente retorne a colônia, perceba que descobriu uma rota melhor que a já conhecida e resolva recrutar outros agentes, mas já haja algum outro agente recrutando, o primeiro dançarino será substituído pelo segundo (o mais recente). Os agentes que já houverem sido recrutados pelo dançarino anterior seguiram seu caminho normalmente, mas novos agentes que forem recrutados seguirão o caminho do novo dançarino.

Recrutando (Recruiting): O agente antBee assume o estado Recrutando caso ele perceba, ao retornar à colônia, que o caminho que ele percorreu até o nó N é melhor (mais curto) que a rota armazenada na tabela da colônia. O agente então dança para recrutar mais agentes para percorrem sua rota, aumentando a quantidade de feromônio contida no caminho. Os únicos agentes que podem ser recrutados são aqueles que já estiverem na colônia quando o agente chegar e começar a dançar. O recrutamento (dança) é realizado através da cópia do vetor de Nós Visitados do agente recrutador para o Vetor de Dança da Colônia.

Seguindo (Following): Estado no qual um agente antBee é recrutado por outro agente para seguir seu caminho. A antBee que decide seguir o caminho de um outro agente o faz copiando o Vetor de Dança da Colônia para a sua lista de Nós Visitados. A "ida" desse trajeto se distingue da que o agente normalmente faz no estado Explorando porque nesse momento o agente está seguindo um caminho preestabelecido e não toma decisões acerca dos próximos nós que visitará, ou seja, não existe aquele cálculo de probabilidade mencionado na descrição do estado explorando. O agente recrutado segue o caminho, um nó por iteração, liberando feromônio sobre os enlaces até chegar ao nó final. Nesse momento seu estado é alterado para "Retornando" e ele regressa normalmente.

3.3 OPERAÇÃO

3.3.1 Inicialização

1) Define o número total de ciclos ou iterações: Cada agente se move uma vez por ciclo
2) Define o número total de agentes (*numberOfAgents):* Número total de agentes antBees na colônia).

3) Define a tabela de roteamento (*routingTable*[x][y]): Uma matriz de 1's e 0's que representa a topologia da rede, na qual os 1's são usados para representar as conexões entre os nós. Enquanto um agente se move de um nó para o outro a respectiva conexão é incrementada em 01 unidade de feromônio (um agente libera feromônio sempre que trafega pelos nós, seja seu estado Explorando, Retornando ou Seguindo).

4) Define o vetor de agentes *antBee*[*numberOfAgents*]

5) Define p (equação que determina a escolha do próximo nó)

6) Para cada agente *antBee*[k]:

- Inicializa estado como Explorando (Exploring).
- Define a distância percorrida (*distanceCovered*) como 0.
- Pseudorrandomicamente define a distância escolhida (*electedDistance*).
- Define o nó atual como a colônia
- Define o número de Nós Visitados (*numberoOfVisitedNodes*) como 1.
- Coloca a colônia na primeira posição do vetor de Nós Visitados (*visitedNodes*) e todas as outras posições são definida como 0.
- Inicia o timer.

3.3.2 Execução

O propósito desta seção é apresentar uma descrição de alto nível da parte de execução do algoritmo AntBee Path. O código é demonstrado através de uma combinação de pseudocódigo e inglês convencional.

```
for i = 0 to cycles
    for j = 0 to numberOfAgents
        if (antBee[j].state == "Exploring")
            if (distanceCovered > electedDistance)
                if (there is at least a node X in the set of nodes
                    reachableNodes that is not in visitedNodes)
```

```
                    agent chooses the next node to be visited based on a
                    weighted equation.
                else antBee[j].state = “Returning”

            else antBee[j].state = “Returning”
        else
            if antBee[j].state == “Returning”
                if distanceCovered != 0
                    agent goes back one node
                else
                    antBee[j].state = “Waiting”
            else
                if antBee[j].state == “Waiting”
                    if (there is an antBee dancing)
                        agent j has a probability p of: antBee[j].state =
                        “Following”
                    else
                        if (distanceCovered < route known by colony)

                            antBee[j].state = “Recruiting”
                        else
                            antBee[j].state = “Exploring”
                else
                    if antBee[j].state == “Recruiting”
                        agent dances and copies it's visitedNodes vector
                        to dancePath
                        and antBee[j].state = “Exploring”
                    else
                        if  antBee[j].state == “Following”
                            agent follows dancePath

                    if distanceCovered == size of dancePath
                            antBee[j].state = “Exploring”
```

3.3.3 Critério de Parada

Dois critérios de parada são utilizados no algoritmo. O primeiro deles é o momento no qual a colônia, através do trabalho de seus agentes, descobre todos os melhores caminhos. Neste momento uma pergunta se faz necessária: como a colônia sabe que descobriu todos os melhores caminhos? Para efeito de simplificação é informado ao algoritmo uma tabela, calculada previamente, que contem todos os melhores caminhos. Ao fim de cada ciclo (iteração) a colônia compara a sua tabela de melhores rotas conhecidas (que é atualizada pelos agentes) com esta tabela previamente calculada. Quando a colônia chegar ao fim de um ciclo e as duas tabelas forem idênticas, isso significará que todas as melhores rotas foram descobertas. Pode parecer contraditório informar previamente uma tabela com os melhores caminhos para um algoritmo cujo objetivo é exatamente calcular esta informação. O motivo pelo qual isso foi feito foi para que fosse possível medir dois parâmetros, iteração e tempo (seção 4.1.2), nos quais todos os melhores caminhos foram descobertos. Isso foi realizado para simplificar a prova de conceito, que objetiva simplesmente testar a viabilidade da solução.

O segundo critério de parada é o limite de número de ciclos. A razão pela qual esse limite é imposto ao algoritmo diz respeito ao problema de estagnação descrito previamente. Após *C* ciclos de execução chega-se a um ponto no qual pode-se considerar que se o algoritmo ainda não descobriu todas as melhores rotas, isso tende a significar que ele estagnou. Isso não quer dizer necessariamente que ele não tem mais chances de descobrir todas as melhores rotas, mas sim que seu desempenho já fosse comprometido o suficiente para justificar a parada. O número de ciclos utilizados no estudo se baseou no desempenho da primeira versão a ser testada (descrita na seção 4.1.2.1.1). Foi através de diversos experimentos preliminares que se chegou a uma linha de base (*baseline*) na qual as demais versões se basearam.

3.4 DESENVOLVIMENTO/CODIFICAÇÃO DO ALGORITMO

3.4.1 Diferentes Versões

O algoritmo AntBeePath descrito neste trabalho foi desenvolvido de forma incremental. Ao todo três versões dele foram implementadas e submetidas ao processo descrito na seção 1.4, Estrutura do Trabalho. As diferentes versões são de fato,

melhoramentos sequenciais. As subseções abaixo descrevem em maiores detalhes o que compõe cada versão e como elas diferem entre si.

3.4.1.1 Primeira Versão: Hybrid

O algoritmo AntBeePath apresentado neste trabalho foi desenvolvido na linguagem C++. A premissa inicial do trabalho foi de definir um algoritmo que combinasse a capacidade de comunicação através da liberação de ferômonio (stigmergia) das formigas com a estratégia de recrutamento das abelhas. Isso resultou na primeira versão de trabalho do algoritmo, chamada de Hybrid. O "Hybrid" simboliza a característica híbrida do algoritmo.

Sendo fiel a analogia biológica, só o destino final é considerado pelos agentes. Quando o agente retorna à colônia, caso a rota que ele tenha acabado de descobrir for melhor que a conhecida pela colônia, a colônia só atualiza a linha da tabela de rotas que corresponde ao nó destino. Mesmo que o caminho novo tenha uma rota mais interessante para um ou mais dos nós intermediários, essa informação é descartada. A intenção é ser fiel ao modelo biológico e simular o retorno de uma abelha a colônia, quando ela informa às abelhas que recruta a distância e direção da fonte de comida, sem se preocupar com pontos intermediários pelo caminho.

Após iniciar as simulações/prova de conceito/testes e avaliar os resultados, melhorias incrementais foram elaboradas e implementadas, resultando em duas novas versões do algoritmo. O impacto das mudanças sobre o desempenho do algoritmo é descrito na seção 5.

3.4.1.2 Segunda Versão: Chain Hybrid

A segunda versão adicionou um mecanismo de atualização em cadeia. A "Chain" é uma variação no algoritmo Hybrid básico, na qual o conhecimento adquirido na rota é totalmente refletido na tabela de rotas da colônia. Em outras palavras, quando um agente descobre uma rota para um nó destino que é melhor do que o caminho conhecida pela colônia, a colônia compara as informações de todos os nós ao longo da rota e os atualiza sempre que o trajeto for melhor do que o conhecido. O nome "Chain" vem do mecanismo de atualização, que se assemelha a uma corrente, onde cada nó do caminho é relevante para a colônia. Esse mecanismo representa uma melhoria sobre o modelo biológico básico. Já que a abelha quando retorna a colônia não se preocupa com posições intermediárias, mas somente com a fonte de comida, que neste caso, seria equivalente ao nó final.

3.4.1.3 Terceira Versão: Decay Chain Hybrid

Uma terceira versão foi definida para lidar com um problema que foi constatado nos experimentos (maiores detalhes no capítulo 5) e que é descrito na literatura, a estagnação. Estagnação é a situação na qual uma rota passa a ter muito mais ferômonio que as demais e, por consequência, fica tão mais atraente que as demais, que a probabilidade de uma agente se manter nela e não buscar outras tende a se aproximar de 100%. Dessa forma, ou o algoritmo se torna incapaz de convergir e alcançar seus objetivos (descobrir todas as melhores rotas) ou o faz de forma a comprometer o seu desempenho.

Na natureza, o feromônio liberado pelas formigas evapora com o passar do tempo. A forma mais fiel de simular isso seria atribuir a cada unidade de ferômonio um valor de intensidade que fosse decrementado com o passar do tempo. O problema dessa solução seria o overhead/complexidade que adicionaria ao algoritmo, assim, uma solução mais simples foi sugerida: reiniciar a tabela de feromônio em intervalos preestabelecidos. Esse mecanismo de controle de estagnação é o diferencial utilizado na criação dessa variação do algoritmo, chamada de "Decay". Ela foi implementada e avaliada de forma a determinar intervalos adequados para a prevenção da estagnação e, além disso, comparar o comportamento do algoritmo com as duas versões anteriores.

3.4.1.4 Algoritmo Equivalente ACO

Um aspecto interessante do algoritmo AntBeePath é que ele pode ser facilmente revertido para um algoritmo equivalente aos algoritmos ACO descritos na literatura. Isso é feito através da supressão do comportamento híbrido do AntBeePath. O propósito desta supressão é permitir uma comparação entre algo que seria bem semelhante aos algoritmos ACO conhecidos e o AntBeePath.

O algoritmo equivalente ACO usado neste estudo é similar ao algoritmo Decay Chain Hybrid descrito na seção 3.4.1.3, com a exceção de que a estratégia de recrutamento da abelha foi removida. Os mecanismos de controle de estagnação (decay) e atualização em cadeia (chain) foram mantidos, a única diferença é que quando o agente descobre uma rota melhor do que a conhecida ele não recruta outros agentes.

4 PROVA DE CONCEITO

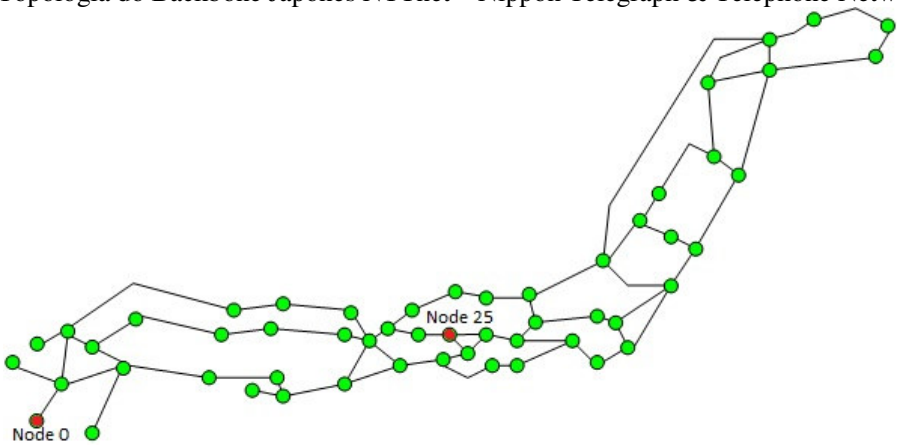
4.1 PARÂMETROS DE TESTE

4.1.1 Topologia Utilizada

A topologia selecionada para testar o algoritmo AntBeePath foi o backbone japonês NTTnet. A razão pela qual ela foi escolhida foi pelo fato de ser tão prevalente na literatura de algoritmos bio-inspirados.

O fato de os nós “0” e “25” da topologia NTTnet se encontrarem destacados na imagem abaixo se deve ao fato de haverem sido utilizados, em diferentes momentos, como colônia, o ponto de partida dos agentes AntBees. Este ponto é explicado em maiores detalhes na seção 4.1.4.

Figura 7 - Topologia do Backbone Japonês NTTnet – Nippon Telegraph & Telephone Network



4.1.2 Número de Iterações e Tempo

O desempenho do algoritmo foi avaliado em termos de dois parâmetros: o número de iterações e o tempo necessários para descobrir todas as melhores rotas. O número de iterações é o mais preciso em termos de simulações porque ele provê uma medida ou indicação do tempo de convergência exigido pelo algoritmo. O tempo de execução é altamente dependente do hardware disponível, além de ser um indicador da viabilidade da implementação do

algoritmo em máquinas comuns e atuais (PCs). Pragmaticamente, ele provê uma indicação sobre quão executável o algoritmo seria em termos de aplicabilidade em diferentes áreas.

4.1.3 Número de Agentes

Os experimentos foram conduzidos utilizando uma grande faixa de agentes (1.000 – 16.000) explorando a rede. O propósito por trás dessa variação foi investigar como o AntBeePath se comportaria com quantidades de agentes diferentes, ou seja, como o número de agentes poderia afetar o desempenho do algoritmo.

O valor inicial de agentes (1000) deriva de pesquisas biológicas reais e corresponde à quantidade de formigas que são necessárias para estabelecer um caminho em 20 min (RATNIEK, 2007).

Cada teste foi realizado uma vez para a quantidade Q de agentes. Em seguida, Q foi incrementado em 100, os demais parâmetros foram reiniciados para as condições iniciais do algoritmo e os testes repetidos, até que Q fosse igual a 16000.

4.1.4 Definição da Colônia

Nos primeiros experimentos realizados, o nó “0” serviu como colônia, o ponto de partida do qual os agentes AntBee partiam para explorar a topologia da rede. Após o término dos testes foi constatado que haveria a possibilidade de que os desempenhos das diferentes versões do algoritmo (descritas na seção 3.4.1) poderiam ser influenciados pelo posicionamento da colônia. Desta forma, outro nó foi escolhido para fazer a função de colônia e os testes foram repetidos para todas as versões. O nó selecionado foi o nó “25” devido a sua posição no centro da topologia.

Os testes não incluíram a utilização de outros nós como colônia por uma questão de limitação de escopo. O motivo de testar um segundo nó como colônia não foi de descobrir qual dos dois seria melhor, mas sim investigar a possibilidade de que o desempenho do algoritmo fosse dependente do posicionamento da colônia.

4.1.5 Definição das Probabilidades

A forma com a qual um agente calcula a probabilidade p de escolher o próximo nó que ele deve visitar durante o estado Explorando é definida pela fórmula descrita abaixo:

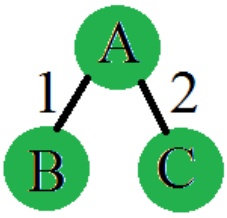
$f_{ij} =$ Quantidade de feromônio no enlace entre os nós i e j, onde i é o nó atual no qual o agente se encontra e j é o "nó candidato", vale dizer, é o nó do vetor de nós alcançáveis que o agente está testando para determinar qual será seu próximo destino, o próximo nó a ser visitado.

$\sum_{k=1}^{n} f_{ik} =$ Soma da quantidade total de feromônio entre o nó i e os n nós que o são adjacentes.

$$p = \frac{f_{ij} * 100}{\sum_{k=1}^{n} f_{ik}}$$

A razão pela qual f_{ij} é multiplicado por 100 é que, inicialmente, foi utilizada uma razão probabilidade/feromônio de 1 para 1. Um exemplo tornará a explicação mais intuitiva: Suponha que o agente esteja atualmente no nó imaginário A que é adjacente aos nós B e C. Suponha que a quantidade de feromônio entre o enlace de A e B seja 1 e de A e C seja 2, conforme a figura 8 abaixo:

Figura 8 - Exemplo da Liberação de Feromônio



Isso significa que a quantidade total de feromônio nos enlaces de A (somando B e C) será 3. Da forma com que a equação foi originalmente construída, a probabilidade do agente escolher o nó B seria 1 * 100 / 3 ≈ 33% e a probabilidade de escolher o nó C seria 2 * 100 / 3 ≈ 66%. Isso mostrou ser um problema nos testes preliminares devido ao fenômeno conhecido como estagnação descrito nas seções anteriores.

$$\frac{f_{ij} * 100}{\frac{\sum_{k=1}^{n} f_{ik}}{2}}$$

Para “diluir” um pouco a importância do feromônio foi introduzido a divisão por 2 no final da fórmula. A divisão diminui em 50% o impacto do feromônio sobre a determinação do próximo nó que o agente escolherá e foi introduzida para reduzir o problema da estagnação, que foi observada na fase inicial de implementação/desenvolvimento do algoritmo. Ainda que a proporção se mantenha a mesma, os resultados da prova de conceito mostraram que a divisão de fato diminui a estagnação.

4.1.6 Definição do Intervalo de Controle de Estagnação

O intervalo no qual o mecanismo de controle de estagnação é utilizado foi definido de forma experimental. Diferentes valores foram utilizados e o que mais se aproximou do “ideal” foi o valor 100. O termo “ideal” é utilizado neste caso para representar o ponto ótimo no qual o algoritmo apresenta o melhor desempenho, ou seja, descobre todas as melhores rotas mais cedo. Por razões de simplificação e diminuição do escopo do trabalho, os resultados deste estudo omitem as comparações desse parâmetro e assumem o valor único de 100, nos casos em que o mecanismo de controle de estagnação foi utilizado. Deve-se observar que o valor do intervalo de controle de estagnação utilizado nesta prova de conceito pode não se aplicar a um outro experimento que utilize condições diferentes. É possível, por exemplo, que a implementação do algoritmo AntBeePath em uma topologia distinta da NTTNet exija um valor diferente para esse intervalo. Esse conceito pode ser expandido para os demais parâmetros, vale dizer: Qualquer alteração em um dos parâmetros usados pode exigir que sejam feitos ajustes nos demais.

4.2 RESULTADOS

4.2.1 Conceitos

O propósito dessa seção é esclarecer alguns conceitos que são utilizados na avaliação de resultados. Dois termos são utilizados com frequência para descrever os resultados, são

eles: convergência e estabilização. Abaixo segue uma breve explanação do sentido no qual cada termo foi empregado.

4.2.1.1 Convergência

No que diz respeito a este trabalho, a expressão “convergência” é utilizada para representar o esforço realizado pela colônia de agentes para chegar ao conjunto de melhores caminhos. Neste sentido, o termo convergência é utilizado para significar a resolução final do problema, ou seja, o momento em que os agentes descobrem todos os melhores caminhos entre a colônia e todos os demais nós da topologia.

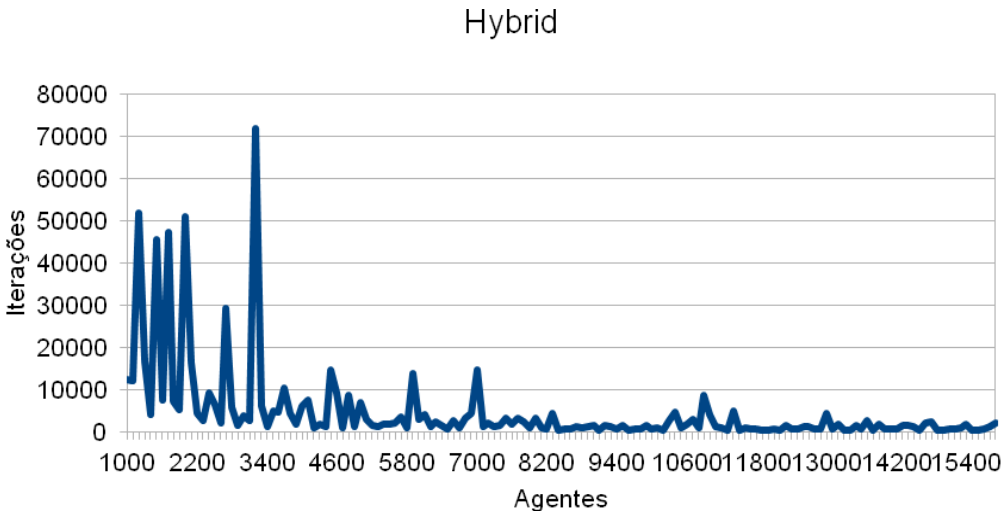
4.2.1.2 Estabilização

O termo “estabilização” é utilizado neste estudo para indicar o momento no qual o algoritmo se aproxima de um estado de equilíbrio em relação aos parâmetros estudados, número de iterações e tempo versus o número de agentes. Vale dizer, os resultados indicam a estabilização quando ele chega a um ponto no qual um aumento sobre o número de agentes não mais influi de forma significante no desempenho do algoritmo.

4.2.2 Versão 1: Hybrid

A primeira versão implementada e testada do algoritmo foi a Hybrid (descrita na seção 3.4.1.1). Nos experimentos, a versão Hybrid se aproximou da estabilização em torno de 7.500 agentes. Inicialmente o algoritmo precisou de uma quantidade de iterações grande (em torno de 50000) para convergir, mas com quantidades de agentes superiores a 3200 o algoritmo se aproximou das 10000 iterações.

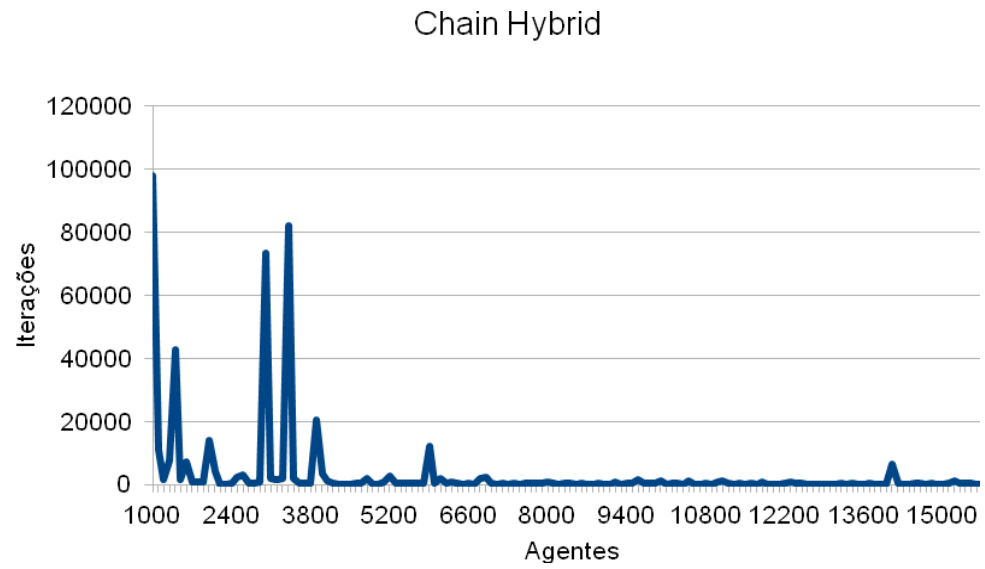
Figura 9 - Desempenho da Versão Hybrid



4.2.3 Versão 2: Chain Hybrid

Inicialmente a versão Chain Hybrid teve resultados piores que a versão Hybrid – 0. Entretanto assim que o número de agentes utilizados passou dos 4000 o desempenho do algoritmo melhorou em relação à versão anterior. Apesar de demorar mais inicialmente para convergir, a versão Chain chegou à estabilização mais cedo e apresentou resultados melhores após esse ponto (em torno de 4000 agentes).

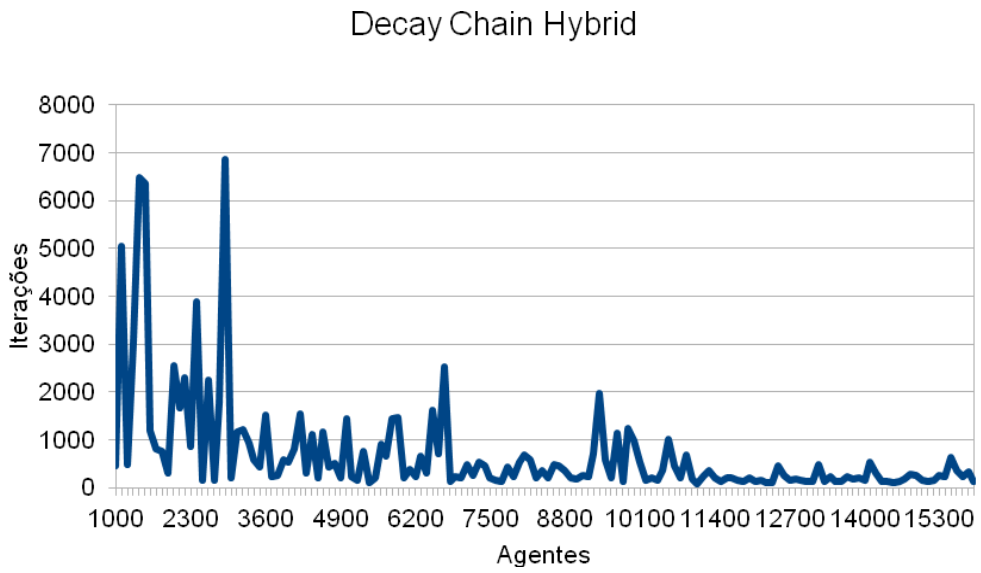
Figura 10 - Desempenho da Versão Chain Hybrid



4.2.4 Versão 3: Decay Chain Hybrid

A terceira versão (Decay Chain Hybrid) foi a que convergiu mais rapidamente e também chegou à estabilização antes. O leitor deve observar que o gráfico abaixo utiliza uma escala muito menor, o que é um indicativo da maior eficiência da versão Decay em relação às anteriores.

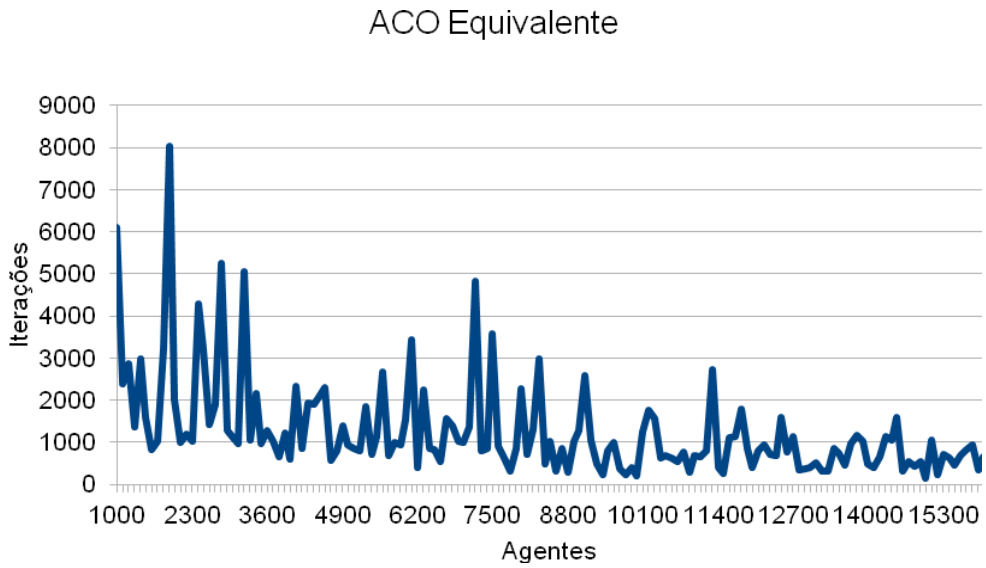
Figura 11 - Desempenho da Versão Decay Chain Hybrid



4.2.5 Versão Equivalente ACO

A versão ACO equivalente (descrita na seção 4.1.2.1.4), que é basicamente a versão 3 (Decay) sem o comportamento de recrutamento das abelhas, foi desenvolvida para poder ser comparada somente com a versão 3, já que ela possui o controle de estagnação que está ausente nas versões 1 (Hybrid) e 2 (Chain) e um mecanismo de atualização em corrente que está ausente na versão 2 (Chain).

Figura 12 - Desempenho da Versão Equivalente ACO

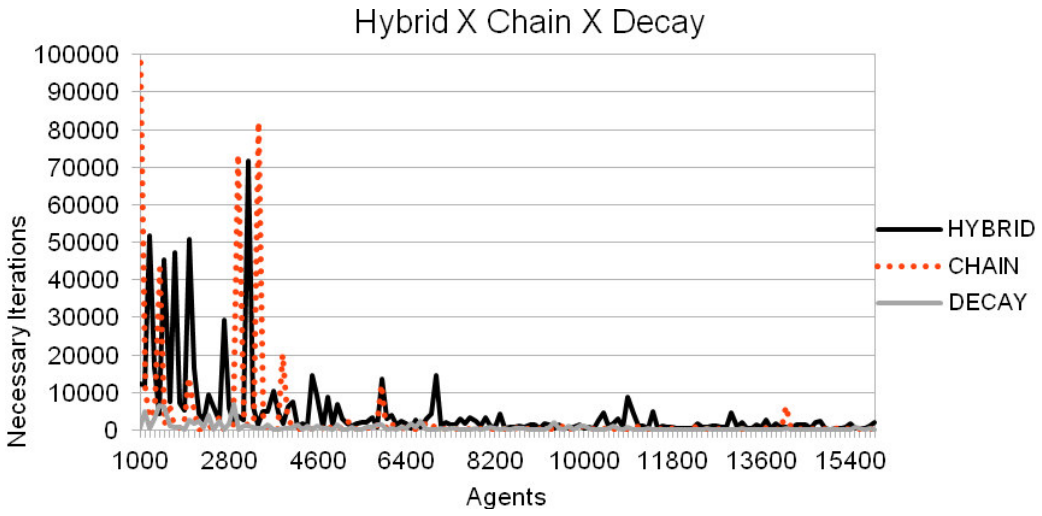


4.2.6 Comparações

4.2.6.1 Comparativo entre as três versões

O gráfico abaixo sintetiza as diferenças entre os resultados das três versões do algoritmo AntBeePath. Como pode ser visto, a versão Decay é a que apresentou os melhores resultados.

Figura 13 - Comparação do Desempenho das Três Versões



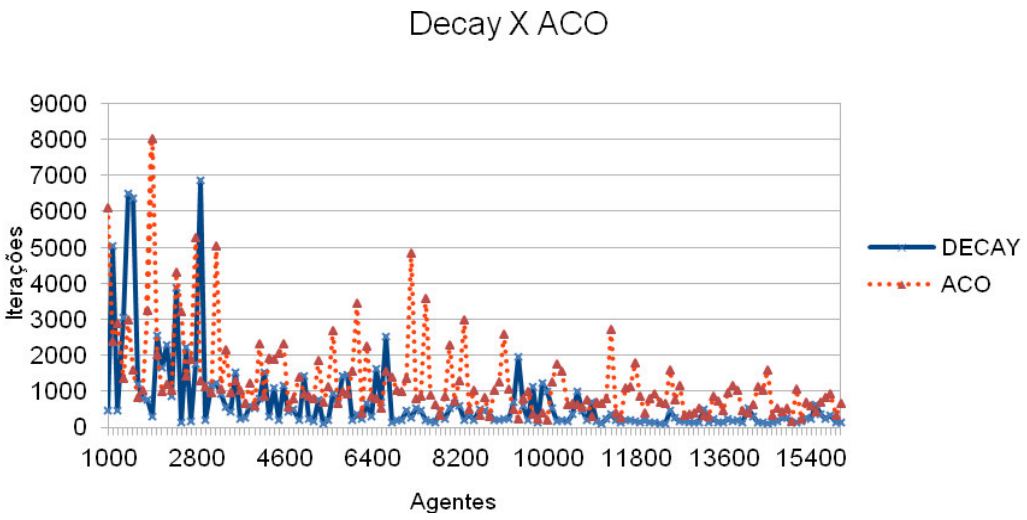
4.2.6.2 Comparativo entre a versão Decay Hybrid Chain e a ACO Equivalente

Como ficou claro pela demonstração no gráfico anterior, o desempenho da versão Decay Hybrid Chain foi superior às duas outras versões (Hybrid e Chain Hybrid) do algoritmo AntBeePath. O acréscimo do mecanismo de controle de estagnação ao algoritmo teve um impacto positivo no seu desempenho, diminuindo bastante a quantidade de iterações necessárias para que o algoritmo convergisse.

O algoritmo ACO equivalente (descrito na seção 3.4.1.4) foi utilizado neste estudo para servir como parâmetro de comparação do desempenho do algoritmo AntBeePath com um algoritmo equivalente aos descritos na literatura e referenciados na seção 2.1.2. Como explicado anteriormente, o algoritmo chamado de ACO equivalente é, basicamente, idêntico ao algoritmo Decay Chain Hybrid. A única diferença é que, com a ausência da estratégia de recrutamento das abelhas, o algoritmo perde seu caráter híbrido.

Como pode ser observado no gráfico abaixo, a versão Decay Chain Hybrid foi capaz de convergir mais cedo do que a versão ACO equivalente.

Figura 14 - Comparativo da Versão Decay Chain Hybrid e Equivalente ACO



4.2.6.3 Posicionamento da Colônia

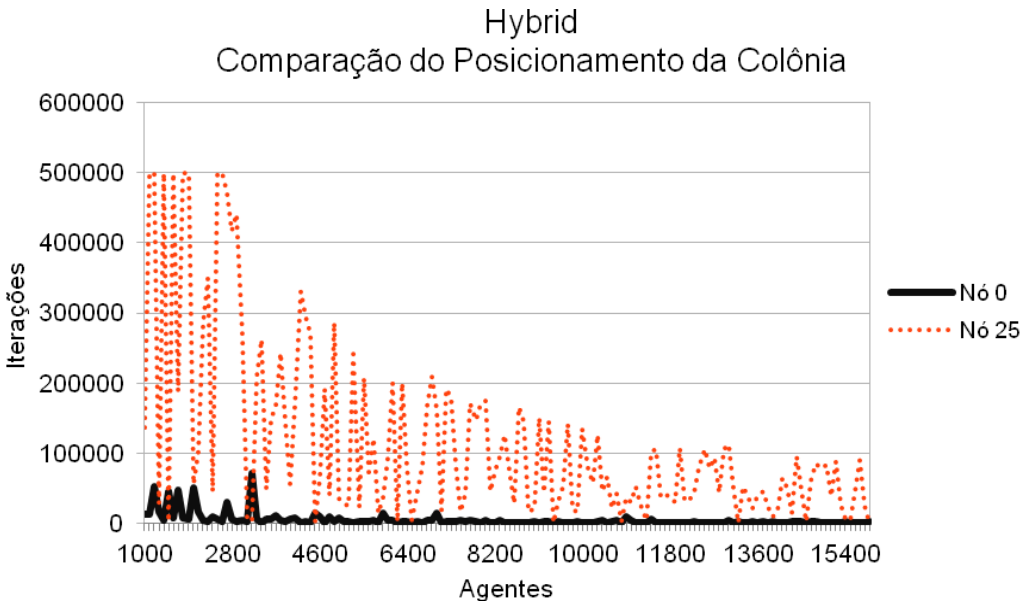
Como descrito na seção 4.1.2.4, o posicionamento da colônia foi testado para verificar que impacto teria sobre o desempenho do algoritmo. Os algoritmos foram testados utilizando primeiramente o nó 0 como colônia (como demonstram os resultados da seção 5.2 - 5.6) e em

um segundo momento o nó 25. Nas seções abaixo, são descritas e analisadas as comparações dos resultados para cada versão do algoritmo. Deve-se observar que, para efeito destas comparações, todos os parâmetros descritos na seção 4.1.2.2 foram mantidos, a única alteração foi à mudança da colônia do nó 0 para o nó 25.

4.2.6.3.1 HYBRID

O primeiro teste que foi realizado para verificar se o posicionamento da colônia afetaria o desempenho do algoritmo foi feito com a versão básica Hybrid. Os resultados demonstram uma queda no desempenho do algoritmo quando a colônia é alterada para o nó 25. O algoritmo inclusive chegou a estagnar em oito oportunidades.

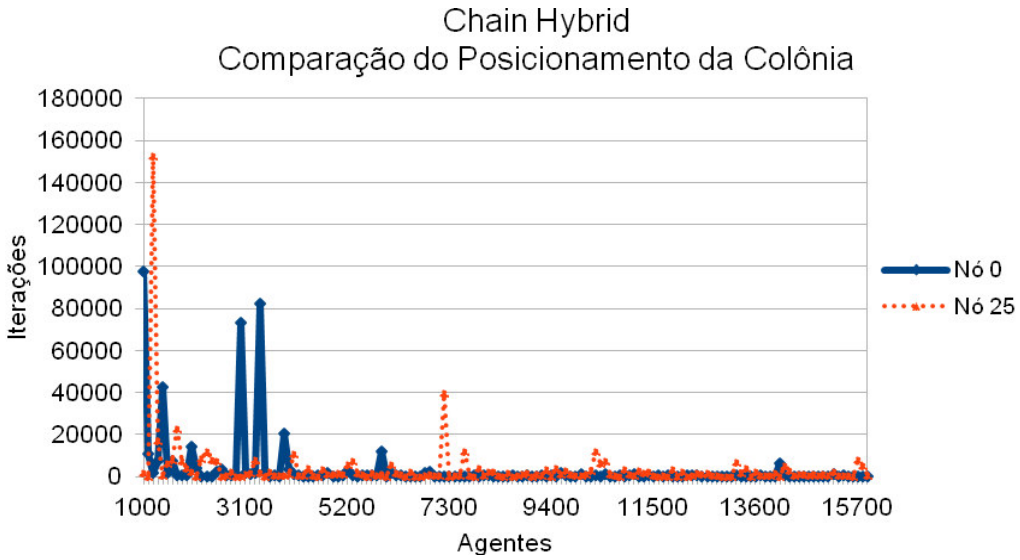
Figura 15 - Comparação do Posicionamento da Colônia no nó 0 e nó 25 na versão Hybrid



4.2.6.3.2 Chain Hybrid

Os testes realizados na versão Chain não apresentaram grandes diferenças em relação ao posicionamento da colônia. Apesar de uma diferença inicial e de alguns picos ao longo dos testes, os resultados demonstram uma similaridade no que diz respeito ao número de iterações.

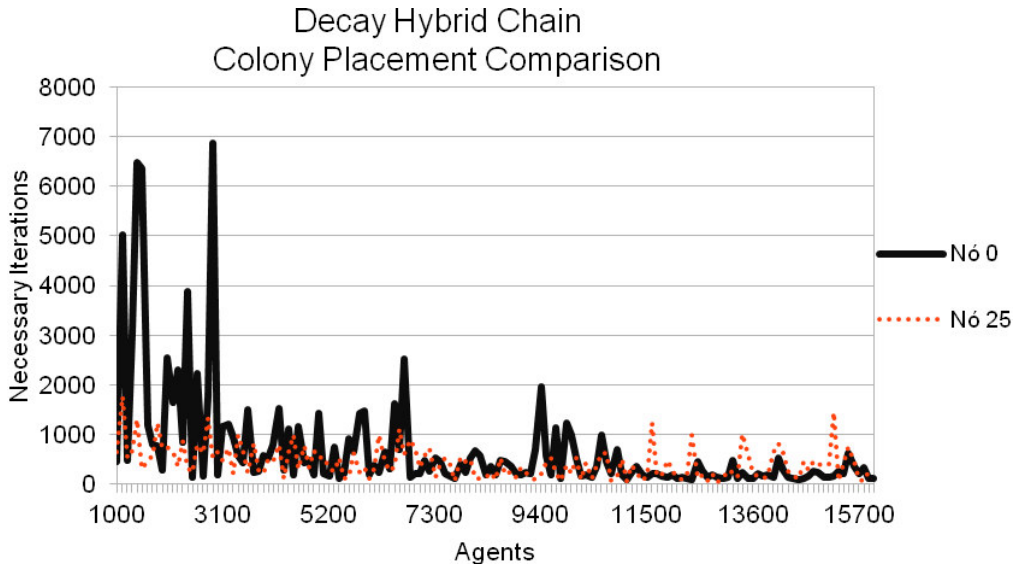
Figura 16 - Comparação do Posicionamento da Colônia no nó 0 e nó 25 na versão Chain Hybrid



4.2.6.3.3 Decay Chain Hybrid

A substituição da colônia para o nó 25 teve um efeito positivo sobre a versão Decay. O algoritmo convergiu mais rápido e também alcançou a estabilização mais cedo.

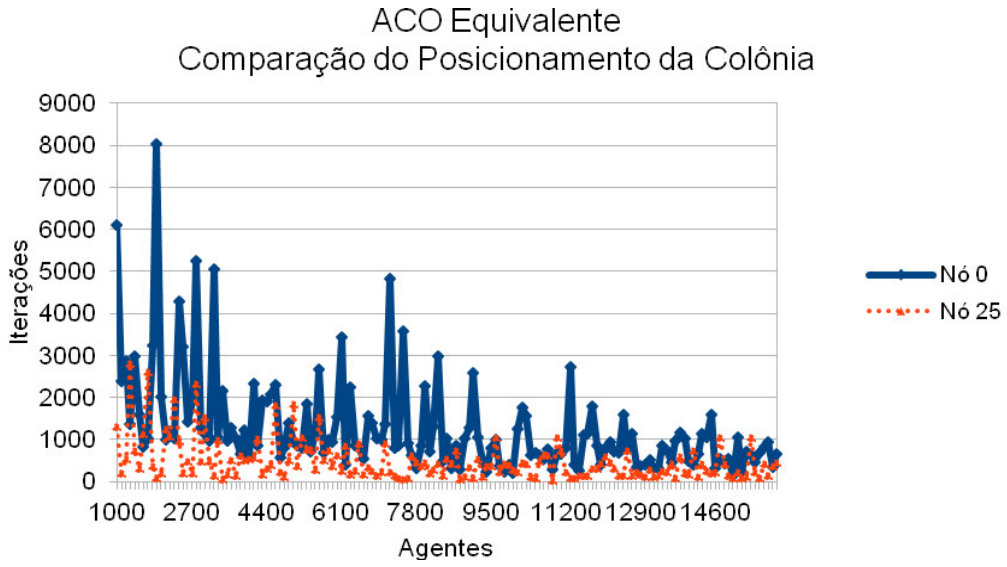
Figura 17 - Comparação do Posicionamento da Colônia no nó 0 e nó 25 na versão Decay Hybrid Chain



4.2.6.3.4 ACO Equivalente

A versão ACO equivalente também foi submetida aos testes com a colônia no nó 25 e demonstrou um desempenho superior aos experimentos realizados com o nó 0 como colônia, como se pode observar pelo gráfico abaixo:

Figura 18 - Comparação do Posicionamento da Colônia no nó 0 e nó 25 na versão Equivalente ACO



5 CONCLUSÃO

Um artigo que sintetiza o trabalho realizado nesta dissertação foi aceito para publicação no *2012 AAAI Fall Symposium on Human Control of Bio-Inspired Swarms* a ser realizado em Arlington, Virginia.

A abordagem utilizada no desenvolvimento do algoritmo AntBeePath foi incremental. De uma versão básica de trabalho (Hybrid), o algoritmo evoluiu para incluir um mecanismo de atualização em cadeia na segunda versão (Chain Hybrid) e em seguida evoluiu mais uma vez para incorporar um mecanismo de controle de estagnação na terceira versão (Decay Chain Hybrid). Os resultados dos experimentos realizados (descritos na seção 5) demonstraram que os incrementos foram responsáveis por um aumento do desempenho do algoritmo. De fato, a versão Decay Chain Hybrid do algoritmo AntBeePath foi a que apresentou o melhor desempenho neste estudo. Sua superioridade se deve ao fato de ser a versão que encontra, de forma consistente, os melhores caminhos (menores) entre a colônia e todos os demais nós da topologia, utilizando uma quantidade de iterações menor do que as demais versões.

Na maioria dos casos estudados neste trabalho, as diferentes versões do algoritmo AntBeePath, foram capazes de convergir e encontrar todas as menores rotas dentro de um intervalo de tempo definido (com a exceção do algoritmo Hybrid usando o nó 25 como colônia que chegou a estagnar em oito ocasiões). Isso não quer dizer que o algoritmo sempre encontrará todas as menores rotas em uma determinada topologia. O que isso realmente significa é que o algoritmo tende a convergir, em outras palavras: a probabilidade de sucesso tende a se aproximar de 100% a medida que o tempo e/ou a quantidade de agentes aumentam.

Um aspecto relevante deste algoritmo é que ele pôde ser facilmente revertido para um algoritmo equivalente aos algoritmos ACO (descritos na seção 2.1.2). Isso foi realizado através da supressão do comportamento híbrido do algoritmo. A estratégia de recrutamento aplicada pelas abelhas foi retirada da terceira versão do AntBeePath, a Decay Hybrid Chain. Isso permitiu que fosse feita uma comparação de algo que seria equivalente a um algoritmo ACO com o AntBeePath. Os resultados dos experimentos indicaram que o Decay Chain Hybrid , a terceira versão do AntBeePath foi mais eficiente do que a alternativa ACO mais simples. O número de iterações e o tempo necessário para que todos os melhores caminhos fossem determinados aumentou quando o componente de recrutamento das abelhas foi removido do código.

A partir dos resultados demonstrados neste estudo pode-se concluir que é possível combinar o comportamento de duas espécies biológicas em um único algoritmo e aplica-lo na resolução do problema de determinação de rotas de forma eficiente.

5.1 CONTRIBUIÇÃO

5.1.1 Visão geral computação bio-inspirada

A primeira contribuição deste estudo foi oferecer uma visão geral da computação bio-inspirada, incluindo uma explanação detalhada acerca da extensa literatura concernente aos algoritmos de otimização de colônia de formigas (ACO) e abelhas (BCO). Apesar da orientação deste trabalho ser a área de Inteligência de Enxame, ele não se limitou a ela, e também se propôs a oferecer um *overview* das demais áreas da computação bio-inspirada, talvez servindo como motivação para trabalhos futuros.

5.1.2 Algoritmo AntBee Path

A segunda contribuição desta dissertação é propor o AntBeePath, um algoritmo híbrido bio-inspirado baseado no comportamento das formigas e abelhas, aplicá-lo no problema de determinação de rotas, evoluí-lo, adicionando mecanismos que possibilitem um aumento de desempenho e por fim, compará-lo a um algoritmo equivalente ACO.

5.2 SUGESTÕES PARA TRABALHOS FUTUROS

5.2.1 Aplicação do AntBeePath em Outras Áreas de Estudo

Um desdobramento do presente estudo seria investigar maneiras de adaptar o algoritmo AntBeePath para aplicação em problemas computacionais diferentes, tais como o problema do caixeiro-viajante (*Travelling Salesman Problem - TSP*) ou até mesmo como algoritmo de roteamento, para analisar o comportamento de uma rede com tráfego simulado, medindo parâmetros tais como *throughput, delay, jitter*, pacotes entregues, chamadas completadas, etc.

5.2.2 Buscar Novos Modelos Biológicos

O algoritmo AntBee Path foi modelado em características específicas do comportamento de duas espécies biológicas: as formigas e as abelhas. Por outro lado, existem muitos outros sistemas biológicos que podem servir como inspiração para futuros trabalhos. Esta dissertação ofereceu um *overview* de outras áreas da computação bio-inspirada que podem servir como norte para novos estudos, porém deve-se ter em mente que a natureza oferece uma infinidade de maneiras nas quais novos modelos podem ser construídos e até combinados entre si com o propósito de solucionar problemas computacionais.

5.2.3 Investigar outros Parâmetros

Um dos parâmetros envolvidos neste estudo que poderiam ser investigados de uma maneira alternativa seria o posicionamento da colônia. Ficou evidente pelos resultados da prova de conceito realizada neste estudo que existe uma relação entre o desempenho do algoritmo e a determinação do nó que servirá como colônia. Um possível desdobramento dessa linha de raciocínio seria investigar um mecanismo que automatizasse a determinação do posicionamento da colônia de modo a otimizar o desempenho do algoritmo. Tal mecanismo exigiria experimentos com topologias de tamanhos e complexidades diferentes.

REFERÊNCIAS

BARÁN, Benjamín; SOSA, Rubéns. A New Approach for AntNet Routing. In: INTERNATIONAL CONFERENCE ON COMPUTER COMMUNICATION AND NETWORKS IEEE ICCCN-2000. Las Vegas, Estados Unidos, 2000. **Proceedings…** 2000. p. 303-308.

BENI, G.; WANG, J. Swarm Intelligence in Cellular Robotic Systems. In: NATO ADVANCED WORKSHOP ON ROBOTS AND BIOLOGICAL SYSTEMS. Tuscania, Itália, 1989. **Proceedings…** v. 102, jun. 1989.

BULLNHEIMER, B.; HARTL, R.F.; STRAUSS , C. A new rank-based version of the Ant System: A computational study. **Central European Journal for Operations Research and Economics**, Austria, v. 7, n.11, p. 25–38, 1999.

DENNY, Adrian Jay; WRIGHT, Jonathan; GRIEF, Ben. Foraging efficiency in the wood ant, Formica rufa: is time of the essence in trail following**? The Association for the Study of Animal Behaviour,** v**.** 62, n.1, p. 139–146, 2001.

DOBSON, Simon et al. A survey of autonomic communications. **ACM Transactions on Autonomous and Adaptive Systems**, Nova Iorque, EUA, v. 1, n.2, p.223-259, dez. 2006.

DI CARO, Gianni; DORIGO, Marco. AntNet: Distributed Stigmergetic Control for Communications Networks. **Journal of Artificial Intelligence Research**, Ithaca, Nova Iorque, v. 9, p. 317-365, 1998.

DI CARO, Gianni; DUCATELLE, Frederick; GAMBARDELLA, Luca Maria. AntHocNet: An Adaptive Nature-Inspired Algorithm for Routing in Mobile Ad Hoc Networks. **European Transactions on Telecommunications,** v.16, n.2, p. 443-455, 2005.

DORIGO, Marco; DI CARO, Gianni; GAMBARDELLA, Luca Maria. **Ant Algorithms for Discrete Optimization**. Artificial Life 5(2), p. 137–172. Bruxelas, Bélgica, 1999.

DORIGO, Marco; GAMBARDELLA, Luca Maria. Ant colonies for the traveling salesman problem. **Biosystems,** v.43, n.21, p 73-81, 1997.

DORIGO, Marco; COLORNI, Alberto; MANIEZZO, Vittorio. **Ant System:** An Autocatalytic Optimizing Process. Technical report, Diparimento di Elettronica, Politecnico di Milano. 1991.

DORIGO, Marco; COLORNI, Alberto; MANIEZZO, Vittorio. Distributed Optimization by Ant Colonies. In: EUROPEAN CONFERENCE ON ARTIFICIAL LIFE, Paris, França. 1992. **Proceedings…** Elsevier, 1992. p. 134–142.

DORIGO, Marco ; DI CARO, Gianni. Two Ant Colony Algorithms for Best-Effort Routing in Datagram Networks. In: IASTED INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED COMPUTING AND SYSTEMS, 10., 1998. **Proceedings…** 1998. p.541-546.

FAROOQ, Muddassar; DI CARO, Gianni A. **Routing Protocols for Next Generation Networks Inspired by Collective Behaviors of Insect Societies**: An Overview. Swarm Intelligence Introduction and Applications. Berlin Heidelberg: Ed. Springer, 2008.

GAMBARDELLA, Luca Maria ; DORIGO, Marco. Ant-Q: A Reinforcement Learning approach to the traveling salesman problem. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 1995. **Proceedings…**1995. p. 252-260.

GAMBARDELLA, Luca Maria ; DORIGO, Marco. Solving Symmetric and Asymmetric TSPs by Ant Colonies. In: IEEE INTERNATIONAL CONFERENCE ON EVOLUTIONARY COMPUTATION (ICEC'96), 1996. **Proceedings…** 1996. p. 622–627.

GRASSÉ, Pierre-Paul. La reconstruction du nid et les coordinations inter-individuelles chez Bellicoitermes natalenis et Cubitermes sp. La theorie de la stigmergie: Essai d'interpretation des termites constructeurs. **Insect Societies**, v. 6, p. 41-83, 1959.

GÜNES, Mesut; SORGES, Udo; BOUAZIZI, Imed. ARA – The Ant-Colony Based Routing Algorithm for MANETs. In: INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING WORKSHOPS (ICPPW). 2002. **Proceedings…** 2002. p. 79 – 85.

HOLLAND, John Henry. **Adaptation in Natural and Artificial Systems:** an Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. Michigan: University of Michigan Press, 1975.

HORN, Paul. **Autonomic Computing:** IBM's Perspective on the State of Information Technology. IBM Research. 2001. Disponível em: <http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf>. Acesso em: 22 out. 2009.

HOUAISS, Antônio. **Novo Dicionário Houaiss da Lingua Portuguesa**. 1 ed. São Paulo: Objetiva,. 2009.

KEPHART, Jeffrey O. A Biologically Inspired Immune System for Computers. Artificial Life IV. In: INTERNATIONAL WORKSHOP ON SYNTHESIS AND SIMULATION OF LIVING SYSTEMS, 4., 1994. **Proceedings…** [Sl.]: MIT Press, 1994.

KHAN, Shakeeb. **Autonomic Nervous System**. Disponível em: <http://www.surgical-tutor.org.uk/default-home.htm?sciences/physiology/autonomic.htm~right>. Acesso em: 3 mar. 2010.

MITCHELL, Melanie. **An Introduction to Genetic Algorithms**. [Sl.]: MIT Press, 1996.

LAMBRINOS, D. et al. A mobile robot employing insect strategies for navigation. **Robotics and Autonomous Systems,** n.30, p. 39–64, 2000.

LEMMENS, N.P.P.M. **To bee or not to bee: a** comparative study in swarm intelligence.. Maastricht, Holanda: Maastricht University, 2006.

LEMMENS, N.P.P.M. et al. Bee behaviour in multi-agent systems: A bee foraging algorithm. Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning. **Lecture Notes in Computer Science**, v. 4865, 2008.

LIANG, S.; ZINCIR-HEYWOOD, A.; HEYWOOD, M. Intelligent packets for dynamic network routing using distributed genetic algorithm. In: GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE. GECCO, 2002. **Proceedings…**2002.

LUČIĆ, Panta; TEODOROVIĆ, Dušan. Computing with Bees: Attacking Complex Transportation Engineering Problems. **Internation Journal on Artificial Intelligence Tools,** v. 12, n. 3. p. 375-394, 2003.

MEISEL, Michael; PAPPAS, Vasileios; ZHANG, Lixia. A Taxonomy of Biologically Inspired Research in Computer Networking. **Computer Networks**, v. 54, n. 6. p. 901-916, 2010.

MÜLLER, M.; WEHNER, R. Path integration in desert ants, Cataglyphis Fortis. In: NATIONAL ACADEMY OF SCIENCES 85. 1988. **Proceedings…** 1988. p. 5287–5290.

NAGPAL, Sahil. Honeybees' dance inspires more efficient Internet servers. **TopNews**. 2007. Disponível em: <http://www.topnews.in/honeybees-dance-inspires-more-efficient-internet-servers-26403>. Acesso em: 28 ago. 2010.

PAPPAS, Vasileios; VERMA, Dinesh; KO, Bong-Jun; SWAMI, Ananthram. A Circulatory system approach for wireless sensor networks. **Ad Hoc Networks**, v. 7. jun 2009.

RAHMATIZADEH, S.; SHAH-HOSSEINI, H.; TORKAMAN, H.: The Ant-Bee Routing Algorithm: A New Agent Based Nature-Inspired Routing Algorithm. **Journal of Applied Sciences** v. 9, p.983-987, 2009.

RATNIEKS, F.L.W. Biomimicry: Further Insights from Ant Colonies? Bio-Inspired Computing and Communication. 58-66, 2008. In: WORKSHOP ON BIO-INSPIRED DESIGN OF NETWORKS, BIOWIRE, 5., 2007. **Proceedings…**2007.

SCHOONDERWOERD, R.; HOLLAND, O. **Minimal agents for communications network routing:** The social insect paradigm. Software Agents for Future Communication Systems. [S.l]: [s.n.],1999.

STÜTZLE, T.; HOOS, H. Improvements on the Ant System: Introducing MAX ¡MIN ant system. In: INTERNATIONAL CONFERENCE ON ARTIfiCIAL NEURAL NETWORKS AND GENETIC ALGORITHMS, 1997. **Proceedings**… 1997. p. 245–249.

TEUSHER, Christof; CAPCARRERE, Mathieu S. On Fireflies, Cellular Systems and Evolware. In: INTERNATIONAL CONFERENCE ON EVOLVABLE SYSTEMS: FROM BIOLOGY TO HARDWARE, 5., 2003. **Proceedings…** 2003.

TIMMIS, Jon et al. Going back to our roots: Second Generation Biocomputing. **International Journal of Unconventional Computing**. 2005.

UCHHULA, Vasundhara; BHATT, Brijesh. Comparison of different Ant Colony Based Routing Algorithms. **IJCA Special Issue on MANETs,** Foundation of Computer Science. p.97–101, 2010

VARGA, A. **Using the OMNeT++ discrete event simulation system in education**. 1999. Disponível em: <http://www.omnetpp.org>. Acesso em: 5 set. 20010.

WATKINS, C.J.C.H. **Learning with delayed rewards**. England: Psychology Department, University of Cambridge, 1989.

WEDDE, Horst F.; FAROOQ, Muddassar. A comprehensive review of nature inspired routing algorithms for fixed telecommunication networks. **Journal of Systems Architecture.** v. 52, p.461-484, 2006.

WEDDE, Horst. F. et al. BeeAdHoc: An Energy Efficient Routing Algorithm for Mobile Ad Hoc Networks Inspired by Bee Behavior. In: CONFERENCE ON GENETIC AND EVOLUTIONARY COMPUTATION. 2005. Washington DC, Estados Unidos. **Proceedings…**2005

WEDDE, Horst F.; FAROOQ, Muddassar; ZHANG, Yue. BeeHive: An Efficient Fault-Tolerant Routing Algorithm Inspired by Honey Bee Behavior. Ant Colony Optimization and Swarm Intelligence. **LNCS,** n.3172, p. 83-94. 2004.

WOKOMA, Ibiso et al. **A Weakly Coupled Adaptive Gossip Protocol for Application Level Active Networks**. In: INTERNATIONAL WORKSHOP ON POLICIES FOR DISTRIBUTED SYSTEMS AND NETWORKS, 3., 2002. Monterrey, California. **Proceedings…** 2002.