versão impressa ISSN 0101-7438 / versão online ISSN 1678-5142

PROPOSTA DE UM *FRAMEWORK* PARA PROTOTIPAGEM DE SISTEMAS HEURÍSTICOS MULTIAGENTES BASEADOS EM ALGORITMOS DE COLÔNIA DE FORMIGAS

Roberto Fernandes Tavares Neto*
Departamento de Engenharia de Produção
Universidade Federal de São Carlos (UFSCar)
São Carlos – SP
tavares@dep.ufscar.br

Moacir Godinho Filho
Departamento de Engenharia de Produção
Universidade Federal de São Carlos (UFSCar)
São Carlos – SP
moacir@dep.ufscar.br

* *Corresponding author* / autor para quem as correspondências devem ser encaminhadas

Recebido em 01/2008; aceito em 12/2008 após 1 revisão
Received January 2008; accepted December 2008 after one revision

Resumo

O estudo de sistemas multiagentes muitas vezes se inicia com a implementação de um algoritmo-base, com variações conforme a necessidade do objeto de estudo. Porém, a comparação entre técnicas propostas se torna difícil, pois não existe uma metodologia de implementação de algoritmos. Deste modo, o presente artigo propõe um *framework* computacional que permita a prototipagem de um grande conjunto de variações de heurísticas baseadas em sistemas de formigas. Como exemplificação desta proposta de *framework*, escolheu-se quatro algoritmos considerados significativos na literatura. Então, realizou-se a implementação dos mesmos, analisando o esforço de implementação necessário. Os resultados mostraram uma redução significativa no tempo de implementação com o uso do *framework* proposto.

Palavras-chave: sistema de colônia de formigas; *framework*; ACS; AS; MMAS.

Abstract

The study of multi-agent systems usually begins by implementing a base-algorithm, which is changed as required by the aim of the research. In this context, carrying out different algorithms, which have already been established, is not a trivial task as it requires implementing these algorithms. This article proposes a computer framework that enables one to prototype a large set of heuristics based on the Ant Colony System metaheuristic. As an example of the proposed framework, four important AS algorithms were implemented. The results show a significant reduction in the implementation time by using the proposed framework.

Keywords: ant colony system; framework; ACS; AS; MMAS.

1. Introdução

Desde 1991, ano em que o algoritmo *Ant System* (AS) foi apresentado por Colorni *et al.* (1991), muitas pesquisas vêm desenvolvendo a ideia da simulação do comportamento de formigas e aplicando-a em problemas clássicos e novos da literatura. A publicação do algoritmo *Ant Colony System* (ACS – Sistema de Colônia de Formigas) por Gambardella & Dorigo (1996) aprimorou o uso do algoritmo AS, originalmente proposto, trazendo melhores resultados à solução de problemas combinatórios.

Muitos pesquisadores aplicaram as heurísticas propostas por Gambardella & Dorigo (1996) em problemas específicos (muitas vezes com alterações para aumentar a performance para a aplicação-alvo). Seu primeiro uso, a resolução do problema clássico do caixeiro viajante simétrico e assimétrico (Dorigo *et al.*, 1996; Gambardella & Dorigo, 1996) foi um ponto de partida para várias outras aplicações, como o problema do Sequenciamento Ordenado (SOP – *Sequential Ordering Problem*; Gambardella & Dorigo, 2000), Roteamento de Veículos com Capacidade (CVRP – *Capacitaded Vehicle Routing Problem*), resolvido originalmente por Bullnheimer *et al.* (1997), e outros problemas combinatórios como os problemas de *scheduling* (Bauer *et al.*, 2000; Stutzle & Hoos, 2000; Blum, 2002) e outros (Dorigo & Stuzle (2004) listam mais de 60 publicações sobre AS e ACS atuando em diversos outros problemas).

Ao se analisar os algoritmos resultantes dessas pesquisas, nota-se que, conforme Stutzle & Linke (2002), todas elas se baseiam em um mesmo algoritmo-base, modificado parcialmente para que se incluam novas características, e ao mesmo tempo se mantenham as funcionalidades básicas da colônia de formigas. Porém, cada implementação é realizada por meio de uma ferramenta computacional distinta, sendo comum encontrar menções a códigos em C/C++, JAVA e até mesmo ferramentas de processamento matemático, como o Matlab, de acordo com a disponibilidade de recursos e as preferências técnicas de cada pesquisador. Uma dificuldade na execução destas pesquisas são as comparações entre os diversos algoritmos existentes e a não padronização da estruturação da codificação, que pode até mesmo obrigar o pesquisador a realizar a reimplementação de códigos provenientes de outras pesquisas, para que seja possível realizar seus objetivos. Com isso, tem-se a implementação do mesmo algoritmo em várias linguagens, para vários ambientes computacionais diferentes.

Neste contexto, o presente artigo tem como objetivo propor e implementar uma estrutura computacional que facilite a pesquisa e a troca de informações entre os pesquisadores que atuam com o algoritmo AS e suas variações. Por meio da análise das alterações necessárias em um único algoritmo-base para a implementação das diferentes heurísticas, é possível se estabelecer um relacionamento e, desta forma, uma classificação entre elas. Ao mesmo tempo, a implementação de vários algoritmos-base, compartilhando um único modelo de representação de dados de entrada, torna muito mais ágil a experimentação do uso de técnicas já desenvolvidas com diferentes classes de problemas. Deste modo, aproveitando-se da alta robustez do algoritmo de otimização por colônia de formigas, o pesquisador pode se focar na modelagem do problema e no estabelecimento de parâmetros de configuração, deixando assim a execução do mesmo a cargo de um algoritmo já previamente implementado. Trabalhos semelhantes são relatados na literatura – por exemplo, Andreatta *et al.* (2002) apresentam um *framework* computacional para criação de heurísticas de busca local; Laguna (1997) mostra alguns softwares para otimização com algoritmos genéticos; por fim, os projetos "Genetic Algorithms Framework" (GA-Fwork, 2008) e "COIN" (Hunsaker, 2008), disponibilizam respectivamente um *framework* multiplataforma para solucionar problemas por meio de algoritmos genéticos e uma infraestrutura computacional para problemas de

pesquisa operacional. Porém, na literatura não foi encontrado nenhum *framework* aplicado para prototipagem de sistemas baseados na heurística da colônia de formigas artificiais.

Para atingir os objetivos propostos, toma-se como base os trabalhos de Colorni *et al.* (1991) e Dorigo *et al.* (1996), que apresentam o algoritmo *Ant-Cycle*, assim como os algoritmos *Ant-Density* e *Ant-Quantity*. Os resultados obtidos pelos *surveys* presentes em Stovba (2005) e Dorigo & Blum (2005) serão usados como base para a exemplificação de como o sistema proposto pode ser utilizado para a implementação de variações de otimizadores baseados em AS com uma única colônia. Adicionalmente, por meio do uso do trabalho de Ellabib *et al.* (2007), mostra-se como sistemas otimizadores por AS de colônias múltiplas podem ser implementados na estrutura computacional apresentada.

Este artigo está estruturado da seguinte forma: na seção 2 é apresentada uma definição do algoritmo AS e suas principais características bem como suas principais variações. Na seção 3, apresenta-se a estrutura computacional proposta que tem como objetivo possibilitar a rápida prototipagem de otimizadores por colônia de formigas. A seção 4 apresenta as escolhas de projeto da implementação da estrutura apresentada, assim como as modificações necessárias para a implementação de alguns algoritmos descritos na seção 2. Por fim, na seção 5, são apresentadas algumas considerações finais e ideias para trabalhos futuros.

2. Revisão de algoritmos otimizadores baseados em colônia de formigas

2.1 Conceituação e estrutura geral do algoritmo

Quando se busca a definição de uma "formiga" na ótica de sistemas de formigas (AS – *Ant Systems*), é interessante citar o trabalho de Dorigo *et al.* (1996), que as define como agentes com capacidades muito modestas que, de alguma forma, buscam imitar o comportamento de formigas reais. Para compreender o conceito de agentes computacionais, cita-se Franklin & Graesser (1996) que definem: "Um agente autônomo é um sistema simulado dentro e fazendo parte de um ambiente, que sente este e age sobre este mesmo ambiente, buscando objetivos próprios e afetando sua percepção futura do meio."

Assim sendo, ao analisar o trabalho de Dorigo *et al.* (1996), pode-se definir uma Formiga como um agente computacional com as seguintes características:

1. Existe em um meio ou ambiente representado matematicamente como um grafo: uma "Formiga" sempre possui uma posição em um nó de um grafo que representa o espaço de busca. A este nó, chamaremos de *nf*.
2. Possui um estado inicial.
3. Embora não consiga "sentir" todo o grafo, consegue ter dois tipos de informação sobre as redondezas: primeiro, o peso de cada trilha ligada ao *nf*; segundo, as características de cada feromônio depositado nesta trilha por outras formigas da mesma colônia.
4. Adicionalmente, notou-se que a implementação computacional pode ser beneficiada se a formiga possuir mais informações sobre o problema, como por exemplo, o número de elementos esperados na solução final. Neste caso, é possível mostrar que existe um ganho computacional na alocação de memória de um vetor, representando a resposta de tamanho fixo em vez de se redimensionar o mesmo a cada passo da construção de uma resposta individual.

No caso dos algoritmos otimizadores baseados em colônias de formigas, o meio em que os agentes estão inseridos é representado por meio de um grafo composto normalmente de duas informações numéricas: uma fixa, estabelecida na definição do problema, e outra alterada no tempo de execução do algoritmo. Ambas as informações são independentes entre si e relacionadas apenas com a ligação entre os pontos *i* e *j* do grafo. Como exemplos de informação fixa, temos distâncias entre cidades (para modelagem de problemas como o caixeiro viajante – TSP), tempos necessários para realização de operações (para problemas de *scheduling*), entre outros. Conforme será apresentado a seguir, a informação alterada no decorrer do tempo é relacionada com valores de "feromônios artificiais", em uma clara analogia com feromônios liberados por formigas durante sua movimentação.

O funcionamento do algoritmo AS, ilustrado no Quadro 1, é baseado no conceito de inteligência emergente: o problema não é resolvido diretamente por um ou mais agentes, mas sim é resultado de um comportamento emergente das várias interações entre os agentes e o meio em que estão inseridos. Esse comportamento se dá pela comunicação entre os agentes e o meio, não pela comunicação entre os próprios agentes. Segundo Dorigo *et al.* (1996), se sabe que formigas reais se utilizam da liberação de substâncias químicas chamadas feromônios para se comunicarem. Ainda segundo os autores, enquanto uma formiga isolada se move aleatoriamente, uma formiga que encontra uma trilha de feromônios depositada anteriormente pode detectá-la. A probabilidade da formiga se mover por uma trilha delimitada por feromônios é maior. Além disso, ao escolher o caminho com feromônios, a formiga deposita mais, reforçando este caminho. Assim sendo, as trilhas mais percorridas irão ter cada vez mais feromônios. Ao possuir uma quantidade maior de feromônios, essas trilhas serão mais escolhidas pelos agentes, que, durante suas passagens, irão ter seus níveis de feromônios aumentados indefinidamente caso não exista um mecanismo de regulação deste *reforço* positivo. De acordo com Blum & Dorigo (2004a), os algoritmos otimizadores que se baseiam em formigas artificiais tentam, durante sua execução, atualizar os valores dos feromônios de forma que a probabilidade de se gerar soluções de alta qualidade aumente no decorrer do tempo. Isso é possível por meio da concentração dos esforços da resolução do problema em regiões do espaço de busca em que se supõe existirem soluções de alta qualidade. Dorigo *et al.* (2000) definem estes feromônios artificiais como variáveis sinergéticas, inspirados no conceito de sinergia observado em insetos sociais como cupins e formigas.

Quadro 1 – Estrutura básica de funcionamento do AS.
(Fonte: Dorigo & Gambardella, 1997)

```
1. Inicialize
2. Execute /* neste nível cada loop é chamado de iteração */
3.         Cada formiga é posicionada no nó inicial
4.         Execute /*neste nível cada loop é chamado passo */
5.                 Cada formiga aplica a regra de transição de estado para gerar a solução
   de forma incremental
6.                 Aplicação de regra de atualização local de feromônio
7.         Até que todas as formigas tenham construído uma solução
8.         Aplicar regra de atualização global de feromônio
9. Até que condição de parada seja satisfeita
```

No Quadro 1, nota-se, respectivamente nos passos 5, 6 e 8, três elementos que são fundamentais nos algoritmos baseados em colônias de formigas:

1. Probabilidade de escolha de um caminho por uma formiga (regra de transição, passo 5);
2. Atualização da trilha de feromônio durante o processo de construção das soluções (passo 6);
3. Regra de atualização global de feromônio, que contém uma regra de evaporação (reforço negativo) e, normalmente, uma regra de depósito de feromônio a ser aplicada após a construção de uma solução individual (reforço positivo, definido no passo 8).

Conforme visto, existem dois momentos onde o feromônio pode ser atualizado: o primeiro ocorre durante a movimentação das formigas, antes que a construção da solução seja finalizada. O segundo ocorre após todas as formigas já tiverem obtido uma solução.

É importante que se tenha em mente que o comportamento previsto pelos passos 6 e 8 do Quadro 1 é crucial para todo o processo de comunicação do agente. Isso porque não existe comunicação direta, e sim uma comunicação sinergética que é instintivamente utilizada para a coordenação das ações entre as formigas. Esta comunicação indireta ocorre ao se incrementar e decrementar os níveis de feromônios, conforme ilustrado na Figura 1. Nesta figura, são mostrados dois caminhos que ligam dois pontos ("Origem" e "Destino"). Inicialmente os níveis de feromônio de ambos os caminhos (representado pelo comprimento das setas correspondentes) estão equilibrados (Fig. 1a). Com isso, a chance de escolha de cada caminho é igual. Assim sendo, um número igual de formigas passará pelos dois caminhos. Porém, as formigas que escolherem o caminho maior demoram mais para voltar, causando uma maior evaporação do feromônio por elas depositado. Computacionalmente, isso é geralmente modelado de forma a penalizar os depósitos de feromônio em caminhos mais longos. Em ambos os casos, o nível de feromônio do caminho mais longo se reduz em relação ao nível de feromônio do caminho mais curto (mostrado na Fig. 1b). Após algumas passagens das formigas ("iterações"), tem-se uma dominância do caminho mais curto (mostrado na Fig. 1c). No caso de um grafo, entende-se este caminho como sendo uma trilha, ou seja, um trecho por onde um agente pode percorrer. Uma trilha liga dois nós *i* e *j*, permitindo a passagem de *i* para *j* e vice-versa.

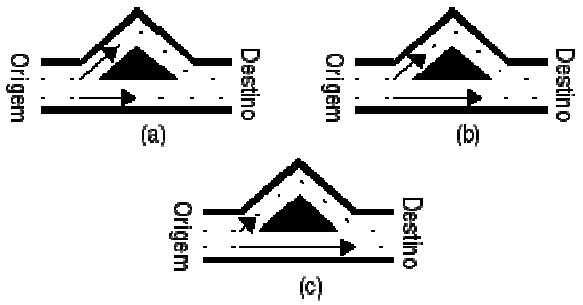


Figura 1 – Exemplo da evolução do nível de feromônio em um caminho simples. O comprimento das setas é proporcional à probabilidade da escolha da rota.
Fonte: adaptado de Dorigo *et al.* (1996).

É por meio de mecanismos como os descritos, que se busca conseguir o equilíbrio entre a exploração e o uso de novas soluções. Conforme Dorigo *et al*. (2000), isso é possível com o uso de variáveis que permitem uma comunicação indireta entre as formigas artificiais. Tais variáveis, chamadas de variáveis sinergéticas, permitem às formigas adaptar seu comportamento para construir soluções de um problema especificado.

É conveniente frisar que esta variável sinergética – a quantidade de feromônio existente em uma trilha – é "sentida" pela formiga e influencia sua tomada de decisão. Desta forma, seguindo o indicado por Dorigo *et al*. (1996), pode-se enumerar algumas características adicionais em relação à formiga artificial:

- Existe em um caminho c_{ij} de um grafo que permite o "depósito de níveis de feromônio"
- Consegue "sentir" os níveis de feromônios de todos os caminhos c_{ij} que saem do nó i
- Consegue determinar quais os caminhos "proibidos"
- Possui um comportamento pseudoaleatório que permite a escolha entre os diversos caminhos permitidos, sendo esta escolha podendo ser influenciada pelos níveis de feromônio sentidos
- Consegue se mover de um nó i para um nó j

2.2 Os principais algoritmos presentes na literatura

No caso do feromônio real, este reforço negativo é realizado pela evaporação do mesmo. Assim, a taxa de feromônio de um caminho não usado tende a decair com o tempo, ao mesmo tempo em que novas soluções geram novos depósitos de feromônio no grafo (passo 8 do Quadro 1). Colorni *et al*. (1991) modelaram matematicamente este comportamento de acordo com a equação 1, também usada por Dorigo *et al*. (1996):

$$\tau(t+n) = \rho \cdot \tau(t) + \Delta\tau_{ij} \qquad (1)$$

Onde:

- $\tau(t+n)$: a quantidade de feromônio de uma trilha no instante $t+n$
- ρ : a constante de evaporação, sendo $0 < \rho < 1$
- $\tau(t)$: quantidade de feromônio existente em uma determinada trilha no instante t
- $\Delta\tau_{ij}$: quantidade de feromônio depositada no caminho ij pelas formigas entre os instantes t e $t+n$.

Por meio da equação 1, o feromônio tende a ser maior nas trilhas em que mais formigas passaram recentemente. Paralelamente a este comportamento, os traços antigos de feromônios tendem a desaparecer devido ao primeiro termo da equação.

O comportamento do ciclo depósito-evaporação do feromônio já foi modificado por diversas pesquisas no campo do AS. Por exemplo, Blum & Dorigo (2004a), se utilizam da equação 2.

$$\tau(t+n) = (1-\rho) \cdot \tau(t)_{ij} + \sum_{s \in G | c_{ij} \in s} F(s) \qquad (2)$$

Onde:

- $\tau(t)_{ij}$ é a quantidade de feromônio existente em uma determinada trilha entre os pontos *i* e *j* no tempo *t*
- *s* é uma solução pertencente ao grafo *G* gerada pela colônia
- $F(s)$ é a função qualidade da solução *s*, podendo ser entendida também como função objetiva do problema.

O algoritmo de Colorni *et al*. (1991) define matematicamente a probabilidade de que um caminho c_{ij} seja escolhido pela formiga *k* (P_{ij}^{k}). Esta probabilidade (passo 5 do quadro 1), também referida na literatura como regra de transição, é mostrada na equação 3:

$$P_{ij}^{k} = \left\{ \begin{array}{c} \dfrac{\left[\tau_{ij}(t)\right]^{\alpha} \cdot \left[\eta_{ij}(t)\right]^{\beta}}{\sum_{m \in permitido_k} \left[\tau_{im}(t)\right]^{\alpha} \cdot \left[\eta_{im}(t)\right]^{\beta}} \quad se\ j \in permitido_k \\ 0 \quad caso \quad contrário \end{array} \right\} \tag{3}$$

Onde:

- α e β: parâmetros que controlam a importância relativa do feromônio e da visibilidade do caminho. É comum encontrar estes dois parâmetros assumindo valores próximos a 0,1 e 2,0 , respectivamente;
- *k*: o índice (identificador único) da formiga;
- $\tau_{ij}(t)$: a quantidade de feromônio entre os nós *i* e *j* no tempo *t*;
- $\eta_{ij}(t)$: a visibilidade da trilha *ij* no tempo *t*. Embora de acordo com cada pesquisa esta variável tenha um significado diferente, é comum se utilizar a definição de Dorigo *et al*. (1996). De acordo com os autores, a visibilidade da trilha *ij* é o inverso da distância entre os pontos *i* e *j* do grafo a ser percorrido.
- $permitido_k$ é o conjunto de nós para onde a formiga pode se mover. No caso do algoritmo original, esse é o conjunto de nós do grafo em que a formiga *k* ainda não passou. Outros trabalhos, como o de Tavares Neto & Coelho (2005) alteram este comportamento.

O último comportamento descrito no algoritmo original AS (passo 6 do quadro 1) diz respeito ao depósito de feromônio. Para tal, Colorni *et al*. (1991) propõem a aplicação do modelo matemático *ant-cycle*, descrito nas equações 4 e 5:

$$\Delta\tau_{ij} = \sum_{n=0}^{k} \Delta\tau_{ij}^{n} \tag{4}$$

$$\Delta\tau_{ij}^{k} = \left\{ \begin{array}{c} \dfrac{Q}{L_k} \quad se(i,j) \in tourAnt_k \\ 0 \quad caso\ contrário \end{array} \right\} \tag{5}$$

Onde:

- $\Delta\tau_{ij}^{k}$ representa a variação de feromônios na trilha que liga os pontos *i* e *j* devido à ação da formiga *k*, conforme equação 5.
- *Q* representa uma constante a ser depositada na trilha
- L_k representa a distância percorrida pela formiga k
- $tourAnt_k$ representa o conjunto de nós que são percorridos pela formiga *k* para a construção de sua solução.

Colorni *et al.* (1991) definem outras formas como a trilha pode ser atualizada. Estas duas formas, denominadas *ant-density* e *ant-quantity*, são descritas nas equações 6 e 7, respectivamente.

$$\Delta\tau_{ij}^{k}(t,t+1) = \begin{cases} \dfrac{Q}{d_{ij}} & \text{se a formiga k vai de i a j entre os instantes t e t+1} \\ 0 & \text{caso contrário} \end{cases} \quad (6)$$

$$\Delta\tau_{ij}^{k}(t,t+1) = \begin{cases} Q & \text{se a formiga k vai de i a j entre os instantes t e t+1} \\ 0 & \text{caso contrário} \end{cases} \quad (7)$$

Onde:

- $\Delta\tau_{ij}^{k}(t,t+1)$ representa a variação de feromônios na trilha que liga os pontos *i* e *j* devido à ação da formiga *k* entre os instantes *t* e *t*+1
- *Q* representa uma constante a ser depositada na trilha
- d_{ij} representa a distância entre os pontos i e j

É importante notar que as equações 6 e 7 se diferenciam do *ant-cycle* em um ponto importante: o momento de atualização do feromônio. Enquanto as modificações das equações 6 e 7 propõem a atualização do feromônio sendo realizado durante a movimentação dos agentes no espaço de busca (chamado de regra de atualização local de feromônios), o *ant-cycle* determina que o grafo deve se manter estático até que todas as soluções da iteração sejam construídas (regra de atualização global de feromônios).

É importante que se perceba que a existência de uma regra de atualização local de feromônios não necessariamente exclui a regra de atualização global: por exemplo, o algoritmo *Ant Colony System* proposto por Gambardella & Dorigo (1996) e Dorigo & Gambardella (1997) se utiliza de ambas as regras (equações 8 e 9 para atualização global e equações 10 e 11 para local).

$$\tau(r,s) = (1-\rho_1)\cdot\tau(r,s) + \rho_1\cdot\Delta_1\tau(r,s) \quad (8)$$

$$\Delta_1\tau(r,s) = \begin{cases} L_{gb}^{-1}, & \text{se (r,s)} \in \text{ao melhor caminho} \\ 0, & \text{caso contrário} \end{cases} \quad (9)$$

$$\tau(r,s) = (1-\rho_2)\cdot\tau(r,s) + \rho_2\cdot\Delta_2\tau(r,s) \quad (10)$$

$$\Delta_2\tau(r,s) = \gamma\cdot\max_{z\in J_k(s)}\tau(r,z) \quad (11)$$

Onde:

- $\tau(r,s)$ é a quantidade de feromônios na trilha que conecta os nós r e s
- L_{gb} é a distância percorrida pela formiga que encontrou a melhor solução
- $J_k(s)$ é o conjunto de nós não percorridos pela formiga
- ρ_1, ρ_2 e γ são parâmetros que podem variar entre 0 e 1

Nos casos estudados por Colorni *et al*. (1991), o algoritmo AS obteve resultados promissores. Porém, é bem conhecido que o mesmo, sem mudanças, possui uma eficiência menor que outras técnicas.

Stovba (2005) cita como causas da ineficiência do AS três fatores:

1. A melhor solução pode ser perdida devido à regra probabilística de seleção de rotas;
2. A convergência de uma solução próxima a ótima é baixa devido à contribuição aproximadamente igual de soluções boas e ruins à atualização dos níveis de feromônios;
3. A memória da colônia armazena variantes obviamente não-promissoras, o que nos leva a um espaço de busca muito extenso quando tratamos de problemas multidimensionais.

Para resolver estes problemas, Stovba (2005) e Dorigo & Blum (2005) mencionam algumas técnicas para melhorar o algoritmo AS:

- "*Elite Ants*" (ASElite): Mencionada pela primeira vez em Dorigo *et al*. (1996), esta técnica se diferencia do AS original ao permitir o depósito de feromônio apenas nos melhores caminhos encontrados.
- Bullnheimer *et al*. (1997) propõem uma variação – o *Ant System with Elitist Strategy and Ranking* (ASrank). Este algoritmo permite que apenas um conjunto de melhores soluções sejam utilizadas para atualizar o feromônio.
- Gambardella & Dorigo (1996) e Dorigo & Gambardella (1997) propõem o *Ant Colony System*. Neste algoritmo, assim como o AS_{Elite}, altera-se a regra de depósito de feromônio original ao permitir a atualização apenas da melhor rota. Adicionalmente, a regra de transição original é modificada. Segundo Stovba (2005), a regra de transição do ACS força a formiga a buscar a solução ótima em um espaço de busca próximo da melhor solução encontrada anteriormente.
- Stützle & Hoos (2000) desenvolveram o *MAX-MIN Ant System* (MMAS). De acordo com os autores, o MMAS impõe limites explícitos de máximo e mínimo aos níveis de feromônio depositados nos caminhos do grafo a serem percorridos. Adicionalmente, assim como ocorre com o ASElite, apenas o agente com a melhor solução pode atualizar os níveis de feromônio. De acordo com Dorigo & Blum (2005), juntamente com o algoritmo ACS, o MMAS é um dos algoritmos de maior sucesso.
- *Best-Worst Ant System* (BWAS), apresentado por Cordon *et al*. (2000), se diferencia do algoritmo AS em três aspectos: primeiro, a atualização do feromônio se dá por meio do reforço positivo da melhor solução e do reforço negativo da pior solução. O segundo aspecto diz respeito ao reinício do processo de busca quando se nota a estagnação. Por fim, no algoritmo BWAS, é inserido uma alteração no tratamento da matriz de feromônios.

Um tópico importante na evolução dos AS é o *Hyper-Cube Framework* (HCF), proposto por Blum & Dorigo (2004b). Sobre ele, Dorigo & Blum (2005) afirmam que não é uma variação do AS, mas sim um *framework* para implementar algoritmos AS com vários benefícios. O principal benefício citado pelos autores é referente ao espaço das soluções do problema. Nas variações do ACO, seu espaço de soluções é limitado em função da qualidade do problema modelado. Com isso, problemas diferentes têm um espaço de busca bem diferente. No HCF, o espaço de soluções é independente da função de qualidade e da instância do problema em si.

Além da mudança das regras de comportamento da formiga ou da colônia, existem pesquisas que vão mais além, ao tratar um conjunto de colônias que atuam em execuções paralelas para a resolução do problema. Como exemplo deste conjunto de algoritmos, cita-se o trabalho de Ellabib *et al*. (2007), que mostra seis algoritmos de ACS paralelos aplicados ao problema de roteamento de veículos com janela de tempo. Segundo os autores, a execução de um conjunto de colônias para a resolução de um problema, quando combinada com uma comunicação coordenada por meio de um módulo de troca (*exchange module* – EM), pode gerar resultados mais eficientes. O módulo de troca nada mais é que um trecho do algoritmo que determina como as informações serão passadas de uma colônia para outra – ou seja, ajudando a definir os processos de cooperação de algoritmos de colônias de formigas com múltiplas colônias.

Assim, analisando o trabalho de Ellabib *et al*. (2007), define-se o Módulo de Troca como uma entidade computacional capaz de armazenar e processar informações presentes no espaço de busca e informações obtidas nas c colônias em execução simultânea, com o objetivo de:

1. Permitir a ação coordenada entre diferentes formigas de diferentes colônias;
2. e/ou alterar o espaço de busca de uma ou mais colônias de forma a permitir alguma vantagem na execução do algoritmo (rapidez de convergência, precisão, etc).

É interessante notar que Tavares Neto & Coelho (2005) também se utilizam do conceito de módulo de troca, com regras baseadas em algoritmos culturais, mas desta vez para o planejamento de rotas de robôs de inspeção. Desta forma, define-se o módulo de troca como sendo o elemento possibilitador da implementação de algoritmos AS com múltiplas colônias.

Os algoritmos até agora apresentados são mostrados no Quadro 2. Ao analisar o Quadro 2, percebe-se que um mesmo mote inicial – o reforço positivo por meio de feromônio simulado regulado por um decaimento padrão (“evaporação”) – produziu um conjunto abrangente de heurísticas.

Porém, embora todos os algoritmos acima propostos possuam um conjunto de regras em comum, sua implementação se dá de forma diferenciada em cada pesquisa. Assim, se nota a necessidade de que cada pesquisador tenha implementado um conjunto comum de regras já bem delimitadas pela literatura. Na próxima seção, será apresentada uma arquitetura lógica que, ao ser implementada, se mostrou robusta o bastante para abranger todas as variações dos algoritmos de otimização por colônias de formigas até agora discutidas.

Quadro 2 – Algumas variações de algoritmos estudados baseados no comportamento de formigas.

Algoritmo	Fonte	Número de Colônias	Usa regra de transição original*?	Usa o ciclo depósito-evaporação original*?	Usa alguma regra de atualização de feromônio Local/ Global?	Uso de EM?
AS	Colorni *et al.* (1991)	1	Sim	Sim	Sim	Não
AS_{Elite}	Dorigo *et al.* (1996)	1	Sim	Não	Sim	Não
AS_{rank}	Bullnheimer *et al.* (1997)	1	Sim	Não	Sim	Não
ACS	Dorigo & Gambardella (1997)	1	Não	Não	Sim	Não
MMAS	Stützle & Hoos (2000)	1	Sim	Não	Sim	Não
BWAS	Cordon *et al.* (2000)	1	Sim	Não	Sim	Não
HCF	Blum & Dorigo (2004)	1	Sim	Não	Sim	Não
Múltiplo AS(**)	Ellabib *et al.* (2007)	>1	Sim	Não	Sim	Sim
AS-CC	Tavares & Coelho (2005)	>1	Sim	Não	Sim	Sim

* Como original, entende-se o algoritmo apresentado em Colorni *et al.* (1991), mostrado nas equações (1), (4) e (5)

** Conjunto de algoritmos definidos por Ellabib *et al.* (2007)

3. O *framework* proposto para prototipagem de sistemas baseados em Algoritmos de Colônias de Formigas

Para que seja possível a detecção de alguns pontos cruciais para a implementação de um *framework* genérico para vários tipos de AS, este artigo propõe uma leitura estendida da listagem mostrada no Quadro 1, conforme o Quadro 3.

É importante que seja observado que todos os passos previstos no Quadro 1 são contemplados no Quadro 3. As principais diferenças são:

- O Quadro 3 divide a regra de atualização global do feromônio presente no passo 8 do Quadro 1 em duas etapas: a primeira relaciona o efeito de cada formiga no feromônio após a construção de todas as soluções de todas as formigas de todas as colônias.

A segunda trata de uma regra de atualização geral de todos os feromônios de todas as colônias. Como exemplo de uma regra de atualização geral de feromônios, temos a evaporação, quando todos os valores são apenas multiplicados por uma constante.

- O Quadro 3 inclui um procedimento opcional de busca local, mostrado no passo 9.

Quadro 3 – Nova abordagem proposta para o algoritmo mostrado no Quadro 1.

1 Inicialize
1.1 Atribua a cada formiga o estado inicial para o agente, incluindo a propriedade de posição inicial.
1.2 Se necessário, crie mais colônias
2 Execute /* neste nível, cada execução do passo 3 é chamada iteração */
3 Se necessário, reinicialize todas as formigas ao seu estado inicial
4 Execute /*neste nível, cada execução do passo 5 é chamada passo*/
5 Faça todas as formigas de todas as colônias escolher o próximo passo e mova-as
6 Se necessário, execute a regra de atualização local de feromônios
7 Até que todas as formigas tenham construído uma solução completa
8 Execute a regra de atualização global de feromônio em dois estágios:
8.1 Um reforço individual é aplicado à trilha.
8.2 Um reforço global é aplicado (como exemplo, tem-se a evaporação descrita da equação 1)
9 Se aplicável, execute procedimentos de busca local.
10 Se necessário, execute a regra de comunicação entre colônias, respeitando o descrito no modulo de troca
11 Até que a condição de parada seja satisfeita

Englobando e agrupando todas as operações e entidades computacionais até agora descritas, temos o *framework* proposto. O *framework* proposto neste artigo é um componente independente de *software* que recebe: i) informações de configuração; ii) um problema de otimização combinatória representado através de um formato conhecido; e obtém a resposta do problema por meio de seus algoritmos internos. Este componente é composto internamente dos seguintes elementos: pelo menos uma Colônia, que contém pelo menos uma Formiga. Possui também um Espaço de Busca e, caso necessário pode possuir um Módulo de Troca.

O relacionamento entre os elementos componentes do *framework* é mostrado na Figura 2. Neste diagrama em blocos, podem-se perceber algumas características importantes:

- A Formiga não está diretamente conectada ao Espaço de Buscas ou ao Módulo de Troca. Sua ligação é com o Gerenciador da Colônia. Com isso, objetiva-se permitir que cada colônia possa ter um filtro diferente do estado atual do espaço de buscas. Desta forma, a implementação e experimentação de algoritmos de múltiplas Colônias de comportamento heterogêneo se torna possível.
- Da mesma forma, percebe-se que cada colônia c possui k_c formigas. Com isso, tem-se que a composição de cada Colônia pode ser diferente, tanto em número quanto na diversidade dos agentes.

- Nota-se também que o Espaço de Busca, o Módulo de Troca e a Colônia estão interligados.
- O Espaço de Busca é alterado e lido pela Colônia durante a criação da solução pelas formigas (linhas 2 e 8 do Quadro 3).
- O Módulo de Troca é acionado: caso necessário, são lidas informações sobre a Colônia e/ou as Formigas, alterando o espaço de buscas conforme requerido pelo algoritmo.
- O Módulo de Troca consegue alterar trechos do espaço de busca específicos de cada Colônia. O valor do feromônio percebido por Formigas de uma Colônia c_1 em um trecho qualquer deve ser independente do valor do feromônio percebido por Formigas de uma Colônia c_2 no mesmo trecho.
- Para gerenciar estes fluxos de dados, foi criado um Gerenciador do *Framework*. Este gerenciador tem como objetivo apenas implementar o algoritmo apresentado no Quadro 3, deixando a execução das ações para os elementos que já discutidos.

O fluxo de dados entre os componentes do *Framework*, assim como a execução sequenciada de forma a permitir a execução das ações seguindo o indicado pelo Quadro 3, é realizado pelo Gerenciador do *Framework*, aqui definido como a entidade computacional que induz os componentes do *Framework* a se coordenarem de forma a seguir o pseudocódigo apresentado.

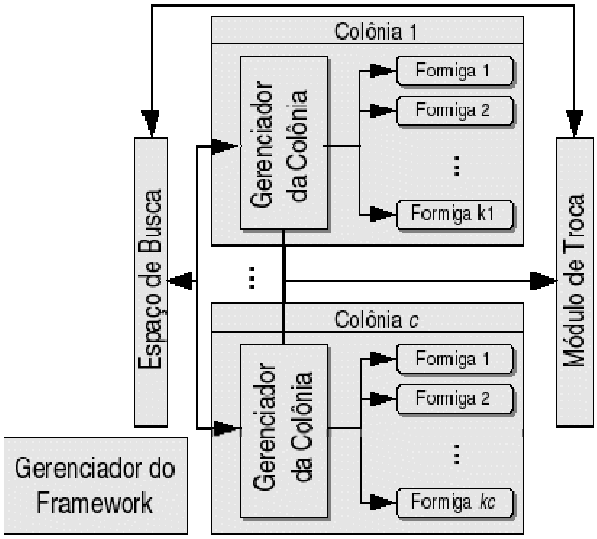


Figura 2 – Diagrama em Blocos do *Framework* Proposto.

O Quadro 4 relaciona os itens de ação do Quadro 3 com as ações tomadas pelos componentes do *Framework* proposto. Neste quadro, foi usada a seguinte nomenclatura para identificar os módulos do *Framework*: FM – Gerenciador do *Framework*; SS – Espaço de Busca; CM – Gerenciador de Colônia; A – Formiga; EM – Módulo de Troca.

Quadro 4 – Ações tomadas pelos elementos do *framework* propostos relativos a cada passo do algoritmo mostrado na Listagem 2.

Item da listagem 2		Ações relacionadas
1	Inicialize	
1.1	Atribua a cada formiga o estado padrão para o agente, incluindo a propriedade de posição inicial.	
2	Execute	
3	Reinicialize todas as formigas ao seu estado padrão	- FM – Envia um sinal de reinicialização para cada Colônia - CM – Cada colônia envia um sinal de reinicialização para cada formiga - A – Volta ao estado estabelecido no item 1.1
4	Execute	
5	Faça todas as formigas de todas as colônias escolher o próximo passo e mova-as	- FM – Solicita que cada CM realize uma movimentação de um passo em cada formiga - CM – Move cada formiga um passo - A – Move um passo
6	Se necessário, execute a regra de atualização local de feromônios	- CM – Após receber a movimentação da formiga, realize o depósito do feromônio correspondente
7	Até que todas as formigas tenham construído uma solução completa	
8	Execute a regra de atualização global de feromônio em dois estágios:	
8.1	Um reforço individual é aplicado à trilha.	- FM – Solicita que cada colônia atue sobre os respectivos feromônios de acordo com suas próprias regras - CM – Aplica a regra de atualização para cada formiga
8.2	Um reforço global é aplicado	- FM – Solicita que cada colônia atue sobre os respectivos feromônios de acordo com suas próprias regras - CM – Redireciona a atualização do feromônio - SS – Aplica a regra de atualização geral para os feromônios relacionados à colônia representada pelo CM
9	Se aplicável, execute procedimentos de busca local.	- FM – Solicita a aplicação de procedimentos de busca local para cada colônia - CM – Aplica os procedimentos de busca local
10	Se necessário, execute a regra de comunicação entre colônias, respeitando o descrito no modulo de troca	- FM – solicita ao módulo de troca que realize a interação entre colônias - EM – Realiza a interação entre colônias
11	Até que a condição de parada seja satisfeita	

A seguir, será analisada a implementação resultante da estrutura descrita anteriormente.

4. Implementação e resultados iniciais

4.1 Uso do *Framework* para a implementação de variações do AS

Nesta seção, é mostrada como a estrutura computacional proposta na seção 3 foi utilizada para a implementação de algumas variações do algoritmo apresentadas na seção 2. Para a escolha de quais algoritmos deveriam ser implementados, tomou-se em consideração o quadro 2, e buscou-se determinar um conjunto inicial que permitisse a demonstração de todas as possibilidades de variação mostradas nas cinco colunas da direita do Quadro 2. A implementação tomou como ponto de partida o algoritmo *Ant-Cycle*, devido a dois motivos:

a) O *Ant-Cycle* possui todas as rotinas de controle das entidades já apresentadas, como movimentação de formigas e alteração do feromônio;
b) Como o depósito do feromônio é realizado após a determinação da solução, a conferência da execução correta por meio da depuração do programa se tornou mais simples.

Foram escolhidos então os algoritmos mostrados no Quadro 5. A execução destes algoritmos foi realizada usando 8 instâncias do problema do caixeiro viajante assimétrico encontradas em TSPLib (2007), com 17, 33, 35, 38, 53, 70 e 170 cidades. O número de formigas utilizadas foi m=10, seguindo o observado em Dorigo & Gambardella (1997). Para o caso de algoritmos que apresentam múltiplas colônias, escolheu-se aleatoriamente o número de 5 colônias. Para a implementação do *Framework*, escolheu-se a linguagem JAVA, sendo implementada uma classe para cada elemento descrito anteriormente. Conforme será mostrado a seguir, o uso de uma linguagem orientada a objeto se mostrou uma forma eficiente e segura de se realizar novas implementações e modificações de métodos já existentes tendo como base o *Ant-Cycle*.

Quadro 5 – Algoritmos escolhidos para a implementação.

Algoritmo	Fonte	Justificativa
Ant-Cycle	Colorni *et al.* (1991)	O *Ant-Cycle* foi utilizado como implementação-base.
AS (*Ant-Quantity* e *Ant-Density*)	Colorni *et al.* (1991)	Demonstrar a viabilidade da arquitetura proposta para o uso exclusivo de regra local de depósito de feromônio (passo 6 da listagem 3).
ACS (*Ant Colony System*)	Gambardella & Dorigo (1996)	Demonstrar o uso da arquitetura proposta para o uso conjunto de regras locais e globais para depósito de feromônio.
MMAS (*Max Min Ant System*)	Stützle & Hoos (2000)	Demonstrar o uso da arquitetura proposta para possíveis evoluções de algoritmos já existentes. A escolha do algoritmo MMAS é justificado neste trabalho com a análise de Dorigo & Blum (2005), que indicam este como uma das variações mais bem-sucedidas de algoritmos otimizados por formigas artificiais.
Múltiplo AS (Homogênio e heterogênio)	Baseado nos resultados de Ellabib *et al.* (2007)	Demonstrar o uso do módulo de troca para aplicações com múltiplas colônias.

4.1.1 Decisões de projeto

O uso de linguagens como o JAVA, que usam o conceito de orientação a objeto, permite o encapsulamento de código em classes. De forma resumida, uma classe é um trecho de código que contém um conjunto de dados e funções necessários para lidar com estes dados (Deitel, 2003, p.68). Uma das etapas do projeto de um *software* orientado a objeto é separar quais dados e operações serão realizados por quais classes. Para isso, criaram-se classes que representam cada um dos blocos mostrados na Figura 2. Para simplificar a documentação e a manutenção do código, escolheu-se dividir as classes em três grupos (implementados na linguagem JAVA como pacotes):

- Classes **base** (pacote *core*), que possuem a implementação base da metaheurística. Não existe previsão para alteração de código nestes pacotes. Como exemplo, tem-se as classes *Ant* e *Colony*.
- Classes **intermediárias** (pacote *mutable*), que possuem a implementação das classes que fazem parte do framework, mas que podem sofrer modificações caso sejam adicionadas novas implementações. Como exemplo, tem-se a classe *Ant Factory*, que deve ser alterada a cada nova classe implementada.
- Classes **de implementação** (pacote *implementations*), que nada mais são do que classes que realmente implementam os comportamentos das diferentes heurísticas no *framework*.

A implementação de estruturas computacionais em linguagens que se utilizem de orientação a objeto como JAVA, C++ e C# é muitas vezes guiada por padrões de projeto, com o intuito de tornar o *software* desenvolvido mais flexível e reutilizável (GAMMA *et al.*, 1995). Assim, outra decisão tomada foi a de se utilizar padrões de projeto (do inglês, *design patterns*). Neste projeto, é usado o padrão de projeto *Fábrica* (do inglês, *Factory*) (GAMMA *et al.*, 1995) para a criação de vários objetos. Na Figura 3, o diagrama UML de classes deste padrão de projeto é apresentado, já o aplicando na geração dos objetos "Ant" (cada instância da classe "Ant" é uma formiga da colônia). O diagrama da Figura 3 possui os seguintes elementos: (i) uma classe *Colony*, que representa uma colônia; (ii) uma classe abstrata *Ant*, que representa uma formiga; (iii) uma classe *ASAnt*, classe derivada de *Ant*, que representa uma formiga que segue as regras estabelecidas pelo algoritmo AS; (iv) uma classe *AntFactory*, que gera formigas para a colônia. As demais classes do projeto, até mesmo as que implementam outros algoritmos, foram suprimidas para simplificar a visualização.

A aplicação do padrão Fábrica na criação de formigas de uma colônia é benéfica, principalmente quando se percebe que uma colônia pode requerer objetos que representem formigas instanciadas de diferentes classes, ou seja, com comportamentos diversos. Neste padrão de projeto, não se instancia *Ant*, mas sim suas classes derivadas. As instâncias das classes derivadas de *Ant* são obtidas pelo método estático *createAnt* da classe *AntFactory*. Ao enviar a solicitação de criação da formiga para o método *createAnt*, é passado um identificador numérico do tipo da formiga a ser criada. Esse método cria o objeto indicado e o devolve convertido em uma instância de *Ant*. Desta forma, a colônia pode abrigar um conjunto heterogêneo de formigas.

Foi realizada ainda uma alteração no padrão de projeto Fábrica definido por Gamma *et al.* (1995). Se o projeto fosse seguir na totalidade o diagrama de classes apresentado pelos autores, o método *createAnt* estaria localizado na classe *Ant*. Desta forma, a inclusão de

nova classe significaria a necessidade da alteração da classe *Ant*. Assim, para simplificar futuras mudanças, escolheu-se remover o método de criação das formigas e colocá-lo em uma classe específica.

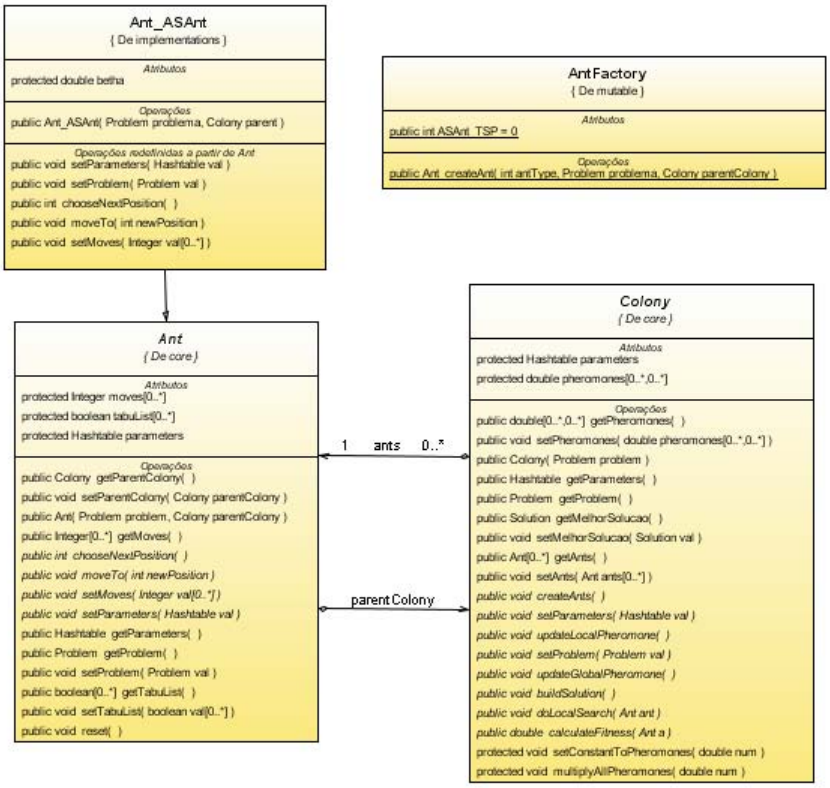


Figura 3 – Diagrama de classes para a geração e uso de formigas.

O comportamento encontrado com o uso do padrão Fábrica para a criação de formigas é muito similar ao que ocorre na criação de colônias pelo gerenciador de colônias, mostrado na Figura 4. Nesta figura, onde foram representados apenas os elementos relacionados à criação da colônia, existem os seguintes elementos: (i) uma classe *ColonyManager* que além de representar o gerenciador de colônias também as armazena; (ii) a mesma classe *Colony* da Figura 3; (iii) uma classe *Colony_AS* que representa uma colônia de formigas que respeita a regra do AS; (iv) uma classe *ColonyFactory*, análoga à classe *AntFactory* da Figura 3.

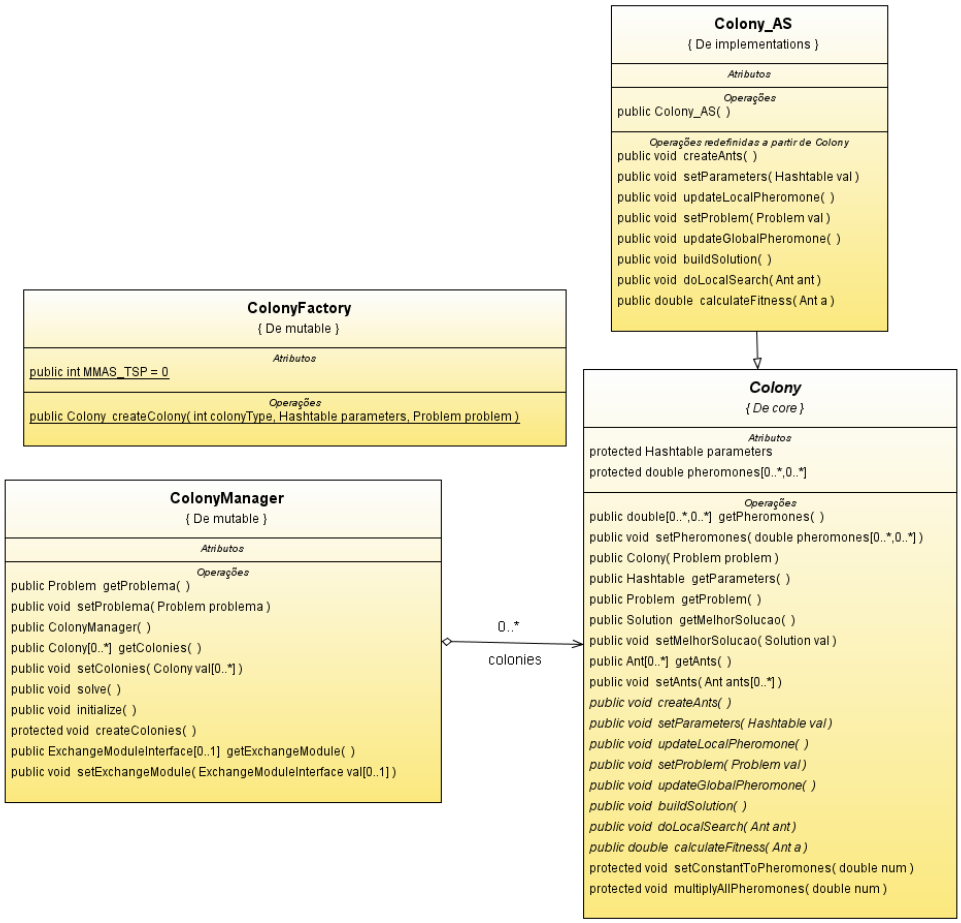


Figura 4 – Diagrama de classes para a geração e uso de colônias.

Assim, para criar um conjunto de colônias e suas respectivas formigas, são seguidos os seguintes passos:

1. O Gerenciador do *Framework* inicializa a classe *ColonyManager*. Durante o processo de inicialização, o *ColonyManager* recebe parâmetros para a criação das colônias.
   1.1 Seguindo os parâmetros determinados no passo anterior, o *ColonyManager* cria as colônias a serem usadas através da classe *ColonyFactory*.
2. O Gerenciador de *Framework* solicita à classe *ColonyManager* a criação das formigas para todas as colônias.
   2.1 A classe *ColonyManager* repassa esta solicitação a todas as colônias.
      2.1.1 Cada colônia cria suas próprias formigas, conforme mostrado anteriormente.

Caso necessária, a implementação de novas variações do algoritmo AS é realizada utilizando os seguintes passos:

1. Se necessário, crie uma nova classe derivada da classe abstrata *Colony* (ou de uma subclasse de *Colony*) que implementa a manipulação de feromônios;
2. Se necessário, crie uma nova classe derivada da classe abstrata *Ant* (ou de uma subclasse de *Ant*) que implementa o comportamento da formiga;
3. Caso tenha criado uma nova classe representando uma colônia, modifique a classe *ColonyFactory* para que seja possível a obtenção de uma nova instância da nova classe;
4. Caso tenha criado uma nova classe representando uma formiga, modifique a classe *AntFactory* para que seja possível a obtenção de uma nova instância da nova classe;

A seguir, será tratada a implementação dos algoritmos já citados no item anterior.

4.1.2 A implementação do *Ant-Quantity* e *Ant-Density*

Tendo a implementação do funcionamento descrito no Quadro 4, a implementação do algoritmo *Ant-Quantity* descrito anteriormente é realizada usando como base a entidade "Colônia". A classe referente a esta entidade é herdada pela nova classe para a implementação do algoritmo *Ant-Quantity*. As mudanças realizadas foram:

- A implementação da regra local de depósito de feromônio descrita na equação 7;
- A eliminação de regras de depósito de feromônio global.

A implementação do algoritmo *Ant-Density* foi muito semelhante à implementação do algoritmo *Ant-Quantity*. A única diferença foi na implementação das regras locais de depósito de feromônio, que obedecem à equação 6.

4.1.3 A implementação do *Ant Colony System*

A implementação do *Ant Colony System* se deu por meio da criação de duas classes distintas:

- Uma classe estende a classe referente à entidade "Colônia", de forma semelhante ao descrito na implementação dos algoritmos *Ant-Quantity* e *Ant-Density*. Porém, esta classe implementa a atualização do feromônio tanto localmente quanto globalmente, conforme descrito por Gambardella & Dorigo (1996) (equações 8 a 11);
- Uma segunda classe, que estende a entidade "Formiga", implementa a regra de transição do ACS.

4.1.4 A implementação do *Max-Min Ant System*

Com a implementação do *Max-Min Ant System*, o uso da implementação usando uma linguagem orientada a objeto se mostra uma boa escolha. Segundo Stützle & Hoos (2000), o algoritmo *Max-Min Ant System* se diferencia do *Ant-System* por dois pontos principais:

- A única atualização de feromônios existente é realizada apenas pela formiga que possui a melhor solução;
- A quantidade de feromônio existente em uma trilha é limitada entre dois valores (um mínimo e outro máximo, derivando daí o nome *Max-Min Ant System*).

Ao recordar a regra de atualização global de feromônios do algoritmo *Ant Colony System*, percebe-se que a mesma também realiza a atualização de feromônios do melhor caminho. Desta forma, a implementação do *Max-Min Ant System* foi realizada estendendo à classe Colônia referente ao algoritmo ACS (descrita no item 4.2). A implementação do *Max-Min Ant System* foi simplificada nos seguintes passos:

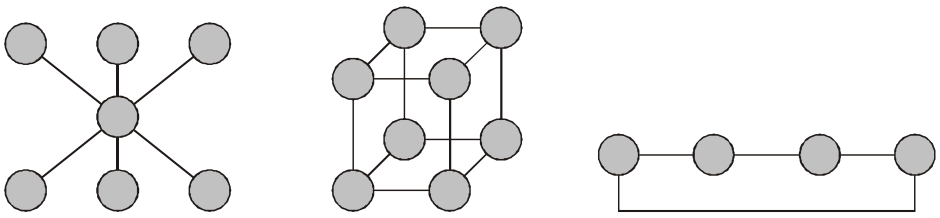
- A atualização local do feromônio (equações 10 e 11) é eliminada (ou seja, um método vazio sobrescreve o método da classe referente à entidade colônia ACS);
- A atualização global do feromônio (equações 8 e 9) é realizada conforme a regra do ACS (bastando para isso uma chamada ao método da superclasse).

Após a atualização global dos feromônios, os valores destes são limitados entre os valores determinados pelo algoritmo. Note que esta é a única implementação necessária para o *Max-Min Ant System*.

A regra de transição descrita por Stützle & Hoos (2000) pode ser entendida como sendo a mesma regra do *Ant System*. Assim, para a implementação da Formiga do *Max-Min Ant System*, é suficiente o uso da a classe referente à Formiga original.

4.1.5 A implementação de estratégias de otimização multicolônias de formigas com colônias homogêneas

Ellabib *et al*. (2007) descrevem três estratégias de troca de informação entre colônias, conforme mostrado na Figura 5. São elas: estrela (Figura 3a), hipercubo (Figura 3b) e anel (Figura 3c). Como exemplo, foi feita a troca de informação usando a estratégia de ligação por anel. O exemplo implementado se inspirou nos resultados apresentados por Ellabib *et al*. (2007), criando o seguinte comportamento: seja c colônias ligadas conforme mostrado na Figura 3c. Sendo c_m e c_n duas colônias imediatamente adjacentes, tal que $m = n$-1. Caso o melhor *fitness* encontrado na colônia m seja melhor que o *fitness* encontrado na colônia *n*, o melhor caminho de m passa para n. Para experimentação, utilizou-se colônias e formigas que implementam o algoritmo *Ant Colony System* (item 4.2). É interessante notar que, embora este algoritmo seja complexo, a estrutura computacional proposta neste artigo, em conjunto com a implementação por meio de técnicas de programação orientada a objeto, reduziu a implementação necessária a uma evolução do código obtido anteriormente. Desta forma, conseguiu-se implementar um algoritmo de múltiplas colônias de formigas reescrevendo um método de uma classe. Neste artigo este algoritmo foi denominado de *Multiple Ant Colony – Homogenius* (MAC-Ho).



(a) (b) (c)

Figura 5 – Três estratégias de troca de informação entre algoritmos ACS de múltiplas colônias. Nos três casos, cada círculo representa uma colônia.

4.1.6 A implementação de estratégias de otimização multicolônias de formigas com colônias heterogêneas

Por fim, buscou-se implementar o caso de maior variabilidade: um algoritmo baseado em múltiplas colônias de formigas artificiais diferentes atuando em conjunto para a resolução de um problema. Para isso, definiu-se o seguinte cenário:

Serão usadas três colônias distintas:

- A primeira colônia obterá a solução com o algoritmo ACS
- A segunda colônia obterá a solução com o algoritmo MMAS
- A terceira colônia obterá a solução com o algoritmo *Ant-Cycle*
- Será usada a mesma estratégia de troca de informações presente no algoritmo MAC-Ho

É importante notar que, para a execução deste cenário, não foi necessária nenhuma alteração em código-fonte. A única alteração foi em arquivos de configuração que definem a composição do otimizador.

4.2 Resultados obtidos

Nesta seção, são avaliados resultados da implementação dos algoritmos vistos na seção anterior utilizando-se o *framework* proposto. Para tal, toma-se como base o trabalho de Chidamber & Kemerer (1994) que propõe um conjunto de métricas para o esforço de implementação de sistemas orientados a objeto. Os autores propõem alguns indicadores, entre eles:

- Métodos ponderados por classe (WMC – *Weighted Method per class*), definido como a soma da complexidade de cada método implementado na classe. Nesta análise, supõe-se a complexidade de todos os métodos unitária, sendo que esta métrica torna o número de métodos por classe.
- Profundidade da árvore de herança (DIT – *Depth of inheritance tree*), definido como o maior número de classes-pai existentes no sistema.

Para realizar esta análise, é necessário entender como o sistema foi implementado permitindo o reuso, mostrado na Figura 6.

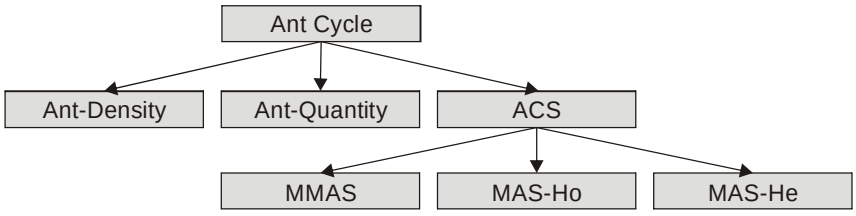


Figura 6 – Reuso de código na implementação dos algoritmos estudados.

Os resultados obtidos são interessantes, pois demonstram que, ao se obter um conjunto de algoritmos implementados por meio do *framework* proposto, implementações de sistemas muito mais complexos se tornam possíveis com um esforço ínfimo de codificação. Na

verdade, o maior esforço detectado por meio das métricas analisadas foi para gerar o algoritmo *Ant-Cycle*, um dos mais simples conceitualmente entre todos os implementados. Com esta estrutura, foram obtidos os resultados mostrados no Quadro 6. Neste quadro, se realizou duas análises da métrica WMC: na primeira, o número total de métodos (WMC Total) é um indicador da complexidade da implementação do algoritmo sem a utilização do *framework* proposto; a segunda se refere à quantidade de métodos efetivamente implementados para o algoritmo analisado.

Quadro 6 – Análise comparativa dos algoritmos estudados.

Algoritmo	WMC sem se considerar o *framework* proposto	WMC requerido ao se usar o *framework* proposto	DIT
AntCycle	38	38	0
AntDensity	38	3	1
AntQuantity	38	3	1
ACS	39	5	1
MMAS	40	3	2
Mac-Ho	38	1	2
Mac-He	42	0	2

Ao observar o Quadro 6, pode-se notar que o maior número de métodos necessários para a implementação das variações dos algoritmos AS se deu na implementação do *Ant-Cycle*, seguido pelo ACS, *Ant-Density*/*Ant-Quantity*, MMAS/MAC-Ho e, por fim, o MAC-He. Para compreender este resultado, é importante que se entenda as etapas de desenvolvimento do sistema analisado. Cada algoritmo, com exceção do *Ant-Cycle*, implementou apenas as variações necessárias em relação a um algoritmo-base.

Estes algoritmos foram implementados e executados. Na Tabela 1, são mostrados os resultados dos algoritmos aplicados a 8 instâncias do problema do caixeiro viajante assimétrico (ATSP – *Asymetric Traveling Salesman Problem*) disponível em TSPLib (2007). A parametrização foi a mesma para todos os algoritmos. O resultado é mostrado em termos de variação do valor ótimo disponível na biblioteca TSPLib. A variação do valor ótimo é definida conforme a equação 12. Os tempos médios de execução são mostrados na Tabela 2.

$$\Delta_{opt} = \frac{V_{encontrado} - V_{ótimo}}{V_{ótimo}} \tag{12}$$

Onde:

- Δ_{opt} indica a variação do ótimo
- $V_{encontrado}$ indica o valor encontrado (ou seja, a distância total percorrida) para a resposta do problema após a aplicação do algoritmo
- $V_{ótimo}$ indica o valor ótimo disponível (ou seja, a distância total percorrida) para a resposta do problema

Para a análise dos valores de Δ_{opt} é importante notar que, embora o ACS e outras meta-heurísticas como algoritmos genéticos e busca tabu, dificilmente conseguirem alcançar resultados aceitáveis quando comparamos com heurísticas desenvolvidas especificamente para o problema do ATSP. Porém, é conhecido que para problemas mais complexos, os resultados obtidos pelo ACS são competitivos.

Tabela 1 – Resultados encontrados pelos algoritmos após execução de testes computacionais.

Instância	Número de cidades	*Ant Quantity*	*Ant Density*	ACS	MMAS	MAS-Ho	MAS-He
br17.atsp	17	136%	136%	136%	0%	44%	44%
ftv33.atsp	33	22%	21%	25%	8%	21%	10%
ftv35.atsp	35	13%	15%	20%	9%	15%	6%
ftv38.atsp	38	15%	20%	14%	10%	14%	11%
ft53.atsp	53	35%	34%	35%	26%	34%	22%
ft70.atsp	70	18%	18%	12%	12%	10%	13%
ftv70.atsp	70	28%	27%	18%	17%	18%	17%
ftv170.atsp	170	42%	49%	37%	24%	33%	27%

Tabela 2 – Tempos computacionais típicos para a obtenção do resultado (em milissegundos).

Instância	*Ant Quantity*	*Ant Density*	ACS	MMAS	MAS-Ho	MAS-He
br17.atsp	398	140	93	100	483	492
ftv33.atsp	586	573	362	368	1810	1925
ftv35.atsp	658	685	411	409	2017	2324
ftv38.atsp	780	838	508	495	2440	2720
ft53.atsp	1576	1691	918	939	4693	4777
ft70.atsp	2558	2810	1709	1579	8520	8535
ftv70.atsp	2563	2850	1781	1617	7887	8587
ftv170.atsp	15496	16139	9768	9704	47746	50950

Aos observar os dados, percebe-se que os algoritmos conseguiram realizar otimização do problema em um tempo computacional viável. Com a parametrização estabelecida, o algoritmo MMAS obteve melhores resultados em grande maioria dos problemas. Nota-se claramente que a dificuldade de um problema do tipo caixeiro viajante assimétrico ser resolvido pelo algoritmo da colônia de formigas depende de fatores além do simples número de cidades. Isto é verificado quando percebemos que o ACS, aplicado de forma idêntica às instâncias br17 (17 cidades) e ft70 (70 cidades), teve um desvio do ótimo de 136% e 12%, respectivamente. A existência de um parâmetro adicional que reflita a complexidade do problema também é indicada quando se analisa apenas a resposta dos algoritmos nas instâncias de 70 cidades (ft70 e ftv70). Nestas instâncias, todos os algoritmos tiveram uma resposta melhor (ou seja, menor variação do ótimo) na instância ft70 do que na instância ftv70.

5. Considerações finais

Este artigo propõe uma estrutura computacional para a realização de experimentos em algoritmos multiagentes com comportamentos inspirados em formigas. Para tal, utilizaram-se os algoritmos AS mais conhecidos na literatura. Buscou-se então definir um conjunto mínimo de entidades independentes que, ao interagirem, pudessem estabelecer um conjunto de comportamentos presentes nos algoritmos apresentados.

Durante o desenvolvimento da arquitetura proposta, notou-se que existem ganhos em termos de custos computacionais quando se fornece à entidade Formiga informações sobre o problema a ser resolvido. No caso, percebeu-se que a execução da lista tabu – ou seja, a memória dos pontos do grafo já visitados pela formiga – se torna muito mais rápida quando se consegue estabelecer o tamanho do problema.

Ao se implementar a estrutura proposta pelo paradigma de orientação a objeto, notou-se que o conceito de herança de classes permite o reaproveitamento de código, tornando a evolução de novos algoritmos uma tarefa muito mais simples.

Com o estabelecimento das regras que são alteradas entre uma evolução e outra dos algoritmos baseados em AS, consegue-se fazer uma análise comparativa como a apresentada no Quadro 6.

A definição e validação inicial do *framework* computacional aqui apresentada permitem a proposição de um trabalho a ser desenvolvido no futuro: a implementação desta em sistemas dedicados. A implementação desta estrutura em vários núcleos de sistemas dedicados pareceu aos autores ser totalmente viável, sendo necessário um conjunto maior de experimentações para a comprovação.

Outro trabalho a ser desenvolvido no futuro é a implementação de um maior número de algoritmos neste *framework*. Embora se tenha buscado neste trabalho a validação da estrutura computacional em si, é inevitável que os próximos passos sejam a inclusão de outras evoluções de algoritmos baseados em comportamentos de formigas presentes na literatura.

Por fim, percebe-se que o *framework* proposto permitiu a rápida implementação e caracterização de algoritmos complexos, como o MMAS e até mesmo estruturas multicolônias com um esforço reduzido – foi necessária apenas a implementação de 2% a 13% dos métodos necessários para o funcionamento do sistema (desconsiderando o algoritmo Mac-He que não requereu nenhuma implementação, apenas mudança em arquivos de configuração). Assim, acredita-se que a proposta para um sistema de prototipagem de sistemas baseados em algoritmos AS foi alcançada.

Referências Bibliográficas

(1) Andreatta, A.A.; Carvalho, S.E.R. & Ribeiro, C.C. (2002). A Framework for Local Search Heuristics for Combinatorial Optimization Problems. **In**: *Optimization Software Class Libraries* [edited by S. Voß and D.L. Woodruff], 59-79, Kluwer.

(2) Bauer, A.; Bullnheimer, B.; Harlt, R. & Strauss, C. (2000). Minimizing total tardiness on a single machine using ant colony optimization. *Central European Journal of Operations Research*, **8**(2), 125-141.

(3) Blum, C. (2002). ACO applied to Group Shop Scheduling: A case study on Intensification and Diversification. *Ants 2002*, 14-27.

(4) Blum, C. & Dorigo, M. (2004a). Deception in Ant Colony Optimization. *ANTS 2004*, 118-129.

(5) Blum, C. & Dorigo, M. (2004b). The Hyper-Cube Framework for Ant Colony Optimization. *IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics*, **34**(2), 1161-1172.

(6) Bullnheimer, B.; Hartl, R. & Strauss, C. (1997). An Improved Ant System Algorithm for the Vehicle Routing Problem. *Operations Research*. Preprint.

(7) Chidamber, S. & Kemerer, C. (1994). A Metrics Suite for Object Oriented Designs. *IEEE Transactions on Software Engineering*, **20**(6), 476-493.

(8) Colorni, A.; Dorigo, M.; Maniezzo, V. (1991). Distributed Optimization by Ant Colonies. *Proceeding of ECAL91 – European Conference of Artificial Life*, Paris, France, 134-142.

(9) Cordon, O.; Viana, I.F., Herrera, F. & Moreno, L. (2000). A New ACO Model Integrating Evolutionary Computation Concepts: The Best-Worst Ant System. *Ants 2000*.

(10) Deitel, H.M. (2003). *Java, Como Programar*. Ed. 4. Bookman.

(11) Dorigo, M.; Bonabeau, E. & Theraulaz, G. (2000). Ant algorithms and stigmergy. *Future Generation Computer Systems*, **16**, 851-871.

(12) Dorigo, M. & Blum, C. (2005). Ant colony optimization theory: A survey. *Theoretical Computer Science*, **344**, 243-278.

(13) Dorigo, M.; Maniezzo, V. & Colorni, A. (1996). Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics*, **26**(1), 29-41.

(14) Dorigo, M. & Gambardella, L.M. (1997). Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, **1**(1), 53-66.

(15) Dorigo, M. & Stutzle, T. (2004). *Ant Colony Optimization*. The MIT Press.

(16) Ellabib, I.; Calamai, P. & Basir, O. (2007). Exchange strategies for multiple Ant Colony System. *Information Sciences*, **177**, 1248-1264.

(17) Frankling, S. & Graesser, A. (1996). Is it an Agent or just a Program? A Taxonomy for Autonomous Agents. *Third International Workshop on Agent Theories, Architectures and Languages*.

(18) GA-Fork (2008). *SourceForge.net: Genetic Algorithms Framework*. Disponível em <http://sourceforge.net/projects/ga-fwork>. Acesso em 10 de setembro de 2008.

(19) Gambardella, L.M. & Dorigo, M. (1996). Solving Symmetric and Asymmetric TSPs by Ant Colonies. *International Conference on Evolutionary Computation*, 622-627.

(20) Gambardella, L.M. & Dorigo, M. (2000). An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem. *INFORMS Journal on Computing*, **12**(3), 237-255.

(21) Gamma, E.; Helm, R.; Johnson, R. & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Ed. 1. Addison-Wesley Publisher.

(22) Hunsaker, B. (2008). *Computational Infrastructure for Operation Research*. Disponível em <http://www.coin-or.org>. Acesso em 10 de setembro de 2008.

(23) Laguna, M. (1997). Metaheuristic Optimization with Evolver, Genocop and OptQuest. *EURO/INFORMS Joint International Meeting 1997 Plenaries and Tutorials*, 141-150.

(24) Stovba, S.D. (2005). Ant Algorithms: Theory and Applications. *Programming and Computer Software*, **31**(4), 167-178.

(25) Stutzle, T. & Hoos, H.H. (2000). Max-Min Ant System. *Journal of Future Generation Computer Systems*, **16**(8), 889-914.

(26) Stutzle, T. & Linke, S. (2002). Experiments with Variants of Ant Algorithms. *Mathware & Soft Computing*, **9**(3), 193-207.

(27) Tavares Neto, R.F. & Coelho, L.S. (2005). Planejamento de Vistorias através de robôs móveis utilizando o algoritmo de colônia de formigas. Tese de mestrado, Pontifícia Universidade Católica do Paraná.

(28) TSPLib (2007). *Travelling Salesman Problem*. Disponível em <http://www.informatik.uni-heidelberg.de/groups/comopt/software/TSPLIB95/index.html>. Acesso em 04/07/2007.