PARETO ANT COLONY para Otimização Bi-objetivo do Custo em Teste de Integração de Software Orientado a Objetos

Rafael da Veiga Cabral
Departamento de Ciências da Computação
Universidade Federal do Paraná, Curitiba - Brasil
verganic@gmail.com

Relatório Técnico – Tópicos em Inteligência Artificial
Prof. Aurora Pozo

Resumo: O presente relatório apresenta os resultados obtidos na implementação do algoritmo Pareto Ant Colony (PACO) com uso de uma estratégia de busca local para otimização bi-objetivo (atributos e métodos) do custo de teste de integração de software orientado a objetos. O algoritmo foi testado para 5 diferentes sistemas. Os custos para cada um dos objetivos foram apresentados e avaliados. No presente relatório também são discutidos os algoritmos e estratégias baseadas na meta-heurística Colônia de Formiga para otimização de um e vários objetivos.

1. Introdução

No teste de integração em sistemas orientados a objeto há necessidade de verificar o nível de acoplamento entre as classes – uso de métodos e atributos de uma classe por outra. Quanto maior for esta relação maior é a dependência entre classes. Deste modo se todas as classes dependem direta ou indiretamente umas das outras seria necessário que todo o sistema estivesse pronto para realizar o teste de integração.

Alternativamente pode-se trabalhar com *stubs* e *drivers* para substituir aquelas classes dependentes que ainda não foram desenvolvidas. Entretanto, quanto maior o número de dependências violadas, maior o custo no teste de integração pois maior o número de *stubs* e *drivers* a serem desenvolvidos. Portanto, faz-se necessário gerar uma ordem para agrupar as classes de modo que o menor número de dependências (métodos e atributos) sejam violados. Dependendo do número de classes do sistema e da complexidade da relação entre elas este problema torna-se *NP-dificil*.

Este problema de otimização combinatiorial foi resolvido utilizando Algorítimos Genéticos [4], porém para cada um dos níveis de acoplamento, dependência, método e atributos foram geradas ordens de classes para teste de integração de maneira separada. Deste modo a otimização foi realizada para apenas cada um dos objetivos em separado. No presente trabalho foi realizado a implementação de um algoritmo baseado na metaheurística Colônia de Formigas que gera a ordem de classes de modo que o custo simultâneo de violação de dependências para métodos e atributos seja o mínimo possível – multiobjetivo.

De acordo com [5] algoritmos baseados na metaheurística Colônias de Formigas tem sido utilizados como métodos alternativos aproximados para resolução de problemas de otimização combinatorial multiojetvo. Sua aplicação foi realizada com sucesso em problemas como seleção de portfólio transporte e roteamento de veículos, agendamento e em problemas de engenharia.

Na seção 2 é apresentada a metaheurística Colônia de Formigas e os principais algoritmos que à utilizam para otimização de problemas com um objetivo. Na seção 3 são introduzidos os conceitos multiobjetivo e dominância de Pareto e que foi utilizado no contexto da implementação realizada no presente trabalho. Na seção 4 são apresentas as principais abordagens de Colônia de Formigas para otimização multiobjetivo. Na seção 5 é apresentado algoritmo implementado Pareto Ant Colony com uma estratégia de busca local pare otimização bi-objetivo dos custos de métodos e atributos para teste de integração de software orientado a objetos. Na seção 6 são apresentados os resultados obtidos para resolução.

2. Metaheurística Colônia de Formigas

A metaheurística Colônia de Formigas baseia-se na inteligência coletiva das formigas utilizada para procurar comida - objetivo - e uma vez encontrada retornar ao ninho. Esta metaheurística foi introduzida por [1], e consiste em simular a construção de caminhos que as formigas utilizam para encontrar comida e retornarem ao ninho. Ao longo do trajeto as formigas depositam uma substância chamada feromônio e utilizam-na como guia até a comida e o posterior retorno ao ninho.

Experimentos mostraram que mesmo com obstáculos no caminho as formigas encontravam a menor trilha [1] de ida e retorno. Vários algoritmos foram projetados para simular esse procedimento na busca por soluções ótimas para problemas de otimização que consistem na construção de caminhos ou percurso em um grafo. O problema do caixeiro viajante (*Travelling Salesman Problem*) é um exemplo. Entre os algoritmos baseados em Colônias de Formigas mais utilizados descantam-se: o *Ant System* (AS), *Ant Colony System* (ACS) e o *Max-Min Ant System*- (MMAX*)*.

2.1. Ant-System

O *Ant-System* foi introduzido por Dorigo [1], e consiste em colocar cada *n* formigas da colônia em um ponto qualquer na trilha – nó do grafo. Cada formiga decide para qual outro ponto deve seguir considerando o feromônio que já está depositado nos pontos vizinhos e um valor heurístico do ponto que revela o quão bom seria se aquele ponto - atributo da solução - fosse visitado. Deste modo, a

formiga *h* estando em um ponto *i* decide probabilisticamente para qual ponto *j* deve ir através da fórmula 2.1.

$$p_{ij}^{h} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i^h}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}},\ se\ j \in N_i^h \quad (2.1)$$

Onde τ_{ij} representa o feromônio depositado entre os pontos (i, j) - aresta do grafo - e η_{ij} o valor heurístico da aresta. α e β são valores que definem a importância do feromônio e da informação heurística na decisão da formiga. O conjunto *N* representa os pontos *j* viáveis para os quais à formiga *h* pode ir estando *i*. Se a formiga decide ir para o ponto *j* ela deixa feromônio entre na aresta (*i,j)* bem como faz-se necessário simular a evaporação do feromônio nas trilhas durante as iterações. O depósito e a evaporação dos feromônios nas trilhas são dados pela fórmula 2.1.

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \sum_{h=1}^{m} \Delta\tau_{ij}^{h} \quad (2.2)$$

Onde:

$$\Delta\tau_{ij}^{h} = 1/L_h,\ se\ a\ formiga\ h\ usou\ a\ aresta\ (i,j),$$

$$\Delta\tau_{ij}^{h} = 0,\ caso\ contrário \quad (2.3)$$

O valor $\rho \in$ *(0,1[* e representa o coeficiente de evaporação e *Lh* representa o comprimento total do caminho percorrido pela formiga *h*. *m* é número de formigas na colônia. Quando o critério de parada estabelecido é alcançado o *Ant-System* pára e retorna o melhor caminho encontrado segundo um determinado objetivo de minimização ou maximização.

2.2. Ant Colony System

O algoritimo *Ant Colony System* foi introduzido por Dorigo e Grambella [2] com fins de melhorar aspectos de diversificação e intensificação em relação ao AS na construção de caminhos ótimos pelas formigas. Esses aspectos são explorados através de esquemas de atualização local e global das trilhas de feromônio e através de um procedimento denominado "regra de proporção pseudo-randômica" (rpp) utilizado pelas formigas para construir o caminho e definido na fórmula 2.4.

$$j = argmax_{l \in N_i^h}\{[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}\},\ se\ q \le q_0$$
$$j = equação(2.2),\ caso\ contrário \quad (2.4)$$

Onde *argmax* significa selecionar uma aresta *j* tal que a composição entre os valor de feromônio e o valor heurístico dos pontos viáveis seja o maior possível. Considerando a importância dos valores de feromônio e do valor heurístico dado por α e β respectivamente. Esse tipo de seleção é controlada pelo parâmetro *q0*. Caso este seja maior ou igual a *q* (valor randômico entre 0 e 1) usa-se *argmax,* caso contrário *j* é selecionado probabilisticamente conforme a fórmula 2.1.

A atualização local é realizada sempre que uma formiga visita uma aresta *(i,j)* e segundo a regra da fórmula 2.5.

$$\tau_{ij} \leftarrow (1-\varphi)\tau_{ij} + \varphi\tau_0 \quad (2.5)$$

Onde φ é a taxa de evaporação e τ_0 é o valor de inicialização das trilhas de feromônio. Com o decaimento do feromônio nos locais onde as formigas já passaram outros locais ainda não explorados passam a ser mais atrativos para visitação para as formigas (diversificação). A atualização global é realizada sempre considerando o melhor caminho encontrado na iteração por todos as formigas (*iteration-best*) ou o melhor caminho até a iteração atual (*best-so-fart*) e realizada segundo a fórmula 2.6.

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \rho\Delta\tau_{ij}^{best} \quad (2.6)$$

Onde, ρ é a taxa de evaporação e $\Delta\tau_{ij}^{best}$ é igual a *1/Lbest* tal que *Lbest* é o valor do melhor caminho até então ou da iteração atual. Deste modo, observa-se que todos os outros pontos da trilha de feromônio sofrem evaporação, porém com um acréscimo *1/Lbest* se a aresta *(i,j)* faz parte de um caminho *iteration-bes*t ou *best-so-far* - (intensificação).

2.3. Max-Min Ant System

Um dos problemas observados no algoritmo AS é a convergência prematura na qual as formigas tendem a passar somente por um trajeto e não tentam explorar outros novos. Isto acontece por serem evaporados todos os valores

de feromônios nos pontos pouco ou não alcançáveis e também pela intensificação das trilhas comuns as formigas.

Para resolver esse problema foi introduzido o MMAX por [7] que utiliza um valor mínimo τ_{min} tal que os valores de feromônio nunca podem ser inferiores, caso contrário assumem o valor mínimo estabelecido. Esse valor normalmente é determinado empiricamente.

Um valor máximo τ_{max} de feromônio também é estabelecido sendo que as trilhas que o excederem são igualados a este valor. De acordo com [7] o valor mínimo é dado por *(1/ρ.L*)* onde *L** é o custo do melhor caminho conhecido até então durante as iterações. O valor máximo também pode ser utilizado para inicializar os valores de feromônios τ_0 para aumentar a diversificação na escolha dos caminhos nas fases iniciais de execução do MMAX.

A atualização dos feromônios é realizada normalmente pela formiga que construiu o melhor caminho na iteração usando a formula 2.6.

Os algoritmos apresentados na seção 2 são voltados para otimização de problemas com um objetivo. Na seção 3 são apresentados algoritmos que consideram a otimização de problemas com mais de um objetivo.

3. Maximização e Minimização de Problemas Multiobjetivo

O problema TSP em sua versão clássica é um problema de minimização de apenas um objetivo – realizar um trajeto por todas as cidades percorrendo á *menor distância possível*. Para transformar este problema em um problema de minimização multiobjetivo bastaria acrescentar uma variável que seria o custo de viagem entre as cidades observando que nem sempre as menores distâncias têm o menor custo. Deste modo o algoritmo de minimização deve buscar um trajeto que contém a menor distancia *x* entre todas as cidades *(i,j)* com o menor custo *x'* entre todas as cidades *(i,j)*.

Este problema *p*ode ser definido como: Dado um conjunto de *n* cidades e um conjunto $\{D_1, D_2\}$ de matrizes $n \times n$ com $D_h = (d^h_{i,j})$, minimizar

$$f(\pi) = (f_1(\pi), f_2(\pi)) \quad \text{com,}$$

$$f_i(\pi) = (\sum d^i_{\pi(j),\pi(j+1)}) + d^i_{\pi(n),\pi(1)}$$

onde π é a permutação no conjunto $(1,2,\ldots,n)$.

Quando há mais de um objetivo a ser minimizado concorrentemente um valor minimo global para um objetivo não significa que o valor para o outro objetivo também será o menor possível, pois há uma relação de dependência entre os dois que pode fazer com que a minimização de um implique em maximização do outro. Então para determinar um conjunto de soluções ótimas para ambos faz-se necessário definir o conceito de dominância de Pareto.

Alternativamente para determinar as boas soluções para ambos objetivos podem ser utilizados esquemas de ordenação lexicográfica e agregação de valores objetivos. Entretanto, no presente trabalho será visto apenas o conceito de dominância de Pareto amplamente utilizado em otimização multiobjetivo e que também está sendo utilizado no estudo prático apresentado na seção 5.

3.1. Dominância de Pareto

O conceito de dominância de Pareto define relações entre os valores objetivos considerando se são maiores, menores e iguais entre os diferentes conjuntos soluções.

No presente estudo será utilizado o conceito de soluções não-dominadas ou não-dominantes em que uma solução *s* é dita não dominada em relação a outra *s'* se a primeira possui para ao menos um dos valores objetivos um valor menor (minimização) que os valores objetivos de outra solução *s'* e vice-versa para *s'*. O conjunto de soluções não-dominadas encontradas por um algoritmo de otimização multiobjetivo é dito *fronteira de Pareto*. Na figura 3.1 é apresentado um exemplo de fronteira de Pareto.

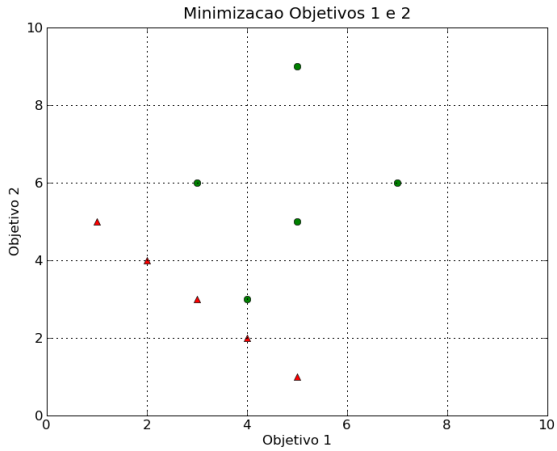

Figura 3.1 - Fronteira de Pareto

É possível observar na figura 3.1 que os triângulos na cor vermelha possuem ao menos um valor para o objetivo 1 ou objetivo 2 que é menor em relação aos demais triângulos - esta é a fronteira de Pareto e são ditas soluções não-dominantes. Os pontos verde possuem valores no mínimo iguais ou superiores as soluções não-dominadas, então os pontos verdes são ditos as soluções dominadas.

Na seção 4 será visto os principais algoritmos baseados em Colônias de Formigas para otimização multiobjetivo e que utilizam o conceito de fronteira de Pareto.

4. Algoritmos Colônia de Formigas para Resolução de Problemas Multiobjetivo

Em [3] foi realizado um estudo empírico e comparativo entre os principais algoritmos para otimização de problemas multiobjetivo baseados em Colônias de Formigas para resolução do problema TSP de dois objetivos. Este estudo revelou resultados promissores em termos da fronteira de Pareto para os algoritmos BicriterionAnt e PACO (Pareto Ant Colony) e que serão revisados na sessão 4.1 e 4.2.

4.1. Algoritmo BicreterionAnt

De acordo com [3] o algoritmo BicreterionAnt foi originalmente proposto para resolução do problema de roteamento de veículos. O algoritmo utilizada duas matrizes de feromônio τ e τ' e duas funções para determinar valores heurísticos η e η' para cada um dos objetivos *k*. As formigas utilizam a regra probabilística apresentadas na formula 4.1.1 para construir o caminho.

$$p_{ij}^{h} = \frac{\tau_{ij}^{\lambda\alpha}\,\tau_{ij}^{',(1-\lambda)\alpha}\,\eta_{ij}^{\lambda\beta}\,\eta_{ij}^{',(1-\lambda)\beta}}{\sum_{l\in N_i^h}\tau_{il}^{\lambda\alpha}\,\tau_{il}^{',(1-\lambda)\alpha}\,\eta_{il}^{\lambda\beta}\,\eta_{il}^{',(1-\lambda)\beta}}, \quad se\ j\in N_i^h \qquad (4.1.1)$$

Onde α e β regulam a influência de τ e η. *N* são os vizinhos viáveis de formiga *h*. O λ é utilizado para priorizar os objetivos de modo a fazer com que cada formiga explore uma região da fronteira de Pareto que esteja mais para um objetivo que para a outro. Na fórmula 4.1.2 é apresentado o cálculo de λ.

$$\lambda_h = (h-1)/(m-1)\ para\ h = \{1,2..,m\} \qquad (4.1.2)$$

O valor *m* representa a quantidade de formigas da colônia. Portanto, para a formiga 1 λ vale *0* fazendo com que o primeiro objetivo seja priorizado na escolha do próximo vizinho e para a última formiga λ vale 1 significando prioridade total para o segundo objetivo.

Uma vez que todas as formigas construíram seus caminhos, as trilhas de feromônios são atualizadas segundo a fórmula 4.1.3 (evaporação).

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij},\ \tau'_{ij} \leftarrow (1-\rho)\tau'_{ij}\quad onde\ \rho\in(0,1] \qquad (4.1.3)$$

Posteriormente o conjunto Pareto de soluções não-dominadas é atualizado com as soluções encontradas na iteração atual e apenas estas atualizam a trilha de feromônio (Intensificação) segundo a regra: $\tau_{ij} \leftarrow \tau_{ij} + 1/l$ tal que *l* é quantidade de formigas que podem atualizar a trilha na iteração atual.

O algoritmo BicreterionAnt foi modificado para que cada formiga apenas atualize uma das matrizes de feromônio afim de intensificar ainda mais a exploração em locais da fronteira de Pareto. Este algoritmo chama-se BicreterionMC e não será tratado no presente trabalho.

4.2. Algoritmo Pareto Ant Colony

O Pareto Ant Colony (PACO) é baseado no algoritmo ACS apresentado na Seção 2.2. De acordo com [3] foi originalmente proposto para resolução do problema multiobjetivo de seleção de portfólios. Trabalha com *k* matrizes de feromônios conforme o número de objetivos *k* e utiliza apenas uma função heurística resultante da agregação dos valores heurísticos para os *k* objetivos. A regra de transição de um ponto na trilha para outro é dado pela fórmula 4.2.1.

$$j = arg\,max_{j\in U}\left[\sum_{k=1}^{K} p_k\cdot\tau_{ij}^{k}\right]^{\alpha}\cdot\eta_{ij}^{\beta},\ se\ q\le q_0,$$

$$j = p(j),\ caso\ contrário \qquad (4.2.1)$$

Onde *p(j)* é dado pela distribuição de probabilidade, representada na fórmula 4.2.2.

$$p(j) = \frac{\left[\sum_{h=1}^{K} p_k\cdot\tau_{ij}^{k}\right]^{\alpha}\cdot\eta_{ij}^{\beta}}{\sum_{h\in U}\left[\sum_{h=1}^{K} p_k\cdot\tau_{ij}^{k}\right]^{\alpha}}, se\ j\in U \qquad (4.2.2)$$

$$p(j) = 0,\ caso\ contrário$$

A atualização dos feromônios é realizada localmente cada vez que uma formiga segue de um ponto para outro na trilha, e utiliza a seguinte regra: $\tau_{ij}^{k} = (1-\rho)\cdot\tau_{ij}^{k} + \rho\cdot\tau_0$ sendo que ρ é a taxa de evaporação e τ_0 é o valor inicial dos feromônios em todas as trilhas.

A atualização global de feromônios é realizada uma vez que cada formiga da população construiu seu caminho e para tal é utilizada a regra $\tau_{ij}^{k} = (1-\rho)\cdot\tau_{ij}^{k} + \rho\cdot\Delta\tau_{ij}^{k}$ em que $\Delta\tau_{ij}^{k}$ assume os seguintes valores: *15* se *(i,j)* pertence ao melhor e ao segundo melhor caminho da iteração, 10 se *(i,j)* pertence

somente ao melhor caminho da iteração, *5* se *(i,j)* pertence somente ao segundo melhor caminho da iteração.

Uma extensão do PACO foi elaborada para resolução do problema bi-objetivo *Flow Shop Scheduling* [5]. Este algoritmo foi utilizado como base para nossa implementação para otimização bi-objetivo do custo em teste de integração de software orientado a objetos e será apresentado na seção 5.1.

5 – Minimizando custos no teste de integração de classes em software orientado a objeto

Uma das fases essenciais no desenvolvimento de software é o teste. Esta visa revelar defeitos no software depois de implementado para que sejam corrigidos e para que falhas não ocorram quando em produção. Além do teste unitário que testa independente os métodos é necessário realizar testes no sistema como um todo quando já integrado nas suas demais partes. Este é o teste de integração.

Para sistemas orientados a objeto o custo do teste de integração deve levar em consideração o nível de acoplamento entre as classes – uso de métodos e atributos de uma classe em outra.

De acordo com [4] dependendo do nível de acoplamento entre as classes há uma relação de dependência entre *clusters* de clases e quanto mais estas dependências são violadas maior é o custo de teste de software pois há maior necessidade de gerar *drivers* e *stubs* que supram a falta das classes faltantes. Portanto, faz-se necessário gerar uma ordem de integração entre as classes para que o mínimo de dependências seja violados. Gerar esta ordem de classes considerando o mínimo de violação de dependências no acoplamento é um problema *NP* completo considerando grandes quantidades de classes e acoplamentos complexos.

Na seção 5.1 é apresentada a implementação do algoritmo (PACO) para geração da ordem das classes para teste de integração otimizando dois objetivos: mínimo de violações de dependência de métodos e de atributos no acoplamento entre as classes.

5.1 - Algoritmo Pareto Ant Colony Implementado

A nossa implementação de PACO para resolução do problema de custo de teste de integração em software orientado a objetos é baseado no algoritmo apresentado na seção 4.2 e também na implementação do mesmo usada para otimização de dois objetivos do problema *Flow Shop Scheduling* abordado em [5]. A escolha de implementação do PACO entre os algoritmos multiobjetivo baseados em Colônias de Formigas foi devido a seus bons resultados apresentados no estudo comparativo [3] e em [5] para resolução do problema já citado. A figura 5.1.1 apresenta o procedimento básico do PACO implementado em pseudo-código.

```
Inicializa_Feromonio (F1,F2, t0)
Equanto nr_iter < max_iter
   Para cada Formiga
      p1 = rand(0,1)
      p2 = 1 – p1
      ini =  Gera_Candidatas ()
      s = Gera_Caminho_Inicial()
      s = Constroi_Caminho(s,q,q0,p1,p2,F1, F2)
      Atualiza_Feromonio_Local(s, F1, F2)
      s = Busca_Local(s)
      s' = Busca_Local(s)
      b  = Melhor_Iteracao()
      b' = Segundo_Melhor_Iter()
      Atualiza_Feromonio_Global(b, b', F1, F2)
      Atualiza_Pareto(P, s, s')
nr_iter += 1
```

Figura 5.1.1 – Pseudo-Código

O procedimento *Gera_Candidatas* retorna um conjunto inicial de classes que não possui qualquer restrição quanto à precedência. Este subconjunto é permutado randomicamente para gerar a parte inicial do vetor de classes – procedimento *Gera_Caminho_Inicial*. Este procedimento foi utilizado para facilitar a geração de vetores de classes viáveis pelo algoritmos PACO. Os demais procedimentos do algoritmo são explicados por partes a seguir:

- **Inicialização das matrizes de feromônios**: foi utilizado o valor 1.0 para as duas matrizes de feromônio – procedimento *Inicializa_Feromonio* da figura 5.1..
- **Construção do Caminho:** Cada formiga foi colocada inicialmente na ultima classe, em *s* resultante do procedimento *Gera_Caminho_Inicial* e a construção restante do caminho foi realizada utilizando o procedimento *Constroi_Caminho* – figura 5.1.1. Neste procedimento inicialmente é gerada uma lista de classes candidatas a serem incluídas no vetor pela formiga e que não violem restrição de precedência. Desta lista uma das classes foi selecionada de acordo com as fórmula 5.1.1 e 5.1.2.

$$i = \begin{array}{c} argmax_{i\ pertenceU} \sum_{k=1}^{2} p_i^k \cdot (\tau_{il}^k)^{\alpha} \cdot (\eta_{ij}^k)^{\beta} \\ se\ q \le q0 \\ i = p(i), caso\ contrário \end{array} \quad (5.1.1)$$

Onde:

$$P(i) = \frac{\sum_{k=1}^{2} p_i^k \cdot (\tau_{il}^k)^{\alpha} \cdot (\eta_{ij}^k)^{\beta}}{\sum_{u \in U} (\sum_{k=1}^{2} p_i^k \cdot (\tau_{il}^k)^{\alpha} \cdot (\eta_{il}^k)^{\beta})} \quad (5.1.2)$$

Sendo que *p1* e *p2* são pesos gerados randomicamente entre 0 ou 1 de modo que *p1 + p2 = 1.0*. Este pesos representam a força de cada objetivo *k* (k=1 métodos, k=2 atributos) na tomada de decisão da formiga *h*.

- **Atualização local das matrizes de feromônio**: Esta atualização foi realizada sempre após a construção do caminho completo por cada formiga conforme explicado na seção 4.2, procedimento *Atualiza_Feromonio_Local* da figura 5.1.1.
- **Busca Local**: A busca local é uma heurística de refinamento para encontrar ótimos locais no espaço de soluções. Entre as mais conhecidas está o *Hill Climbing* que utiliza estrutura de vizinha geradas através de determinados operadores. A combinação de busca local com algoritmos de colônias de formigas tem apresentados melhores resultados [5]. No experimento foi utilizado inicialmente uma busca local baseada em Pareto e que considera ambos objetivos a serem otimizados e de maneira a criar um conjunto Pareto de soluções não-dominantes para cada execução da busca local. Porém, este procedimento não apresentou resultados satisfatórios e então optou-se por utilizar um algoritmo de busca local para cada objetivo para melhorar as soluções construídas pelas formigas em função de cada objetivo (método e atributo) – procedimentos *Busca_Local* da figura 5.1.1 As configurações utilizadas foram operador de vizinhança *2-opt* com 20 vizinhos e 20 iterações sem melhora para todos os sistemas apresentados na seção 6.

- **Atualização global das matrizes de feromônio**: Esta atualização foi realizada conforme explicado na seção 4.2 sempre considerando a melhor e a segunda melhor solução da iteração, procedimento *Atualiza_Feromonio_Global* da figura 5.1.1..

- **Critério de Parada** – foi estabelecido um número máximo de iterações proporcional ao número de formigas (normalmente o dobro).

6 – Resultados

A metodologia utilizada foi rodar a implementação do PACO, seção 5.1.1, 5 vezes para cada um dos sistemas (A,B,C,D,E) usado nos testes realizados em [4]. Naquele trabalho são utilizados Algoritmos Genéticos para fazer otimização de apenas um objetivo independente (dependência, método e atributos) para cada um dos sistemas.

As soluções da fronteira de Pareto encontradas pelo nosso algoritmo estão de acordo com o conceito estabelecido na seção 3.1. Para cada sistema foram utilizados como parâmetros do algoritmo PACO aqueles indicados em [5]. $q_0 = 0.75$, $\alpha = 1$, $\beta = 1$, $\tau_{min} = 0.0001$ e $\tau_0 = 1.0$ com exceção do número de formigas e do número de iterações que foi estabelecido respectivamente de acordo com número de classes e aproximadamente o dobro do número de classes de cada sistema. A seguir são apresentados os resultados obtidos.

Observa-se na figura 6.1 que os valores ótimos para os objetivos de métodos e atributos foram encontrados.

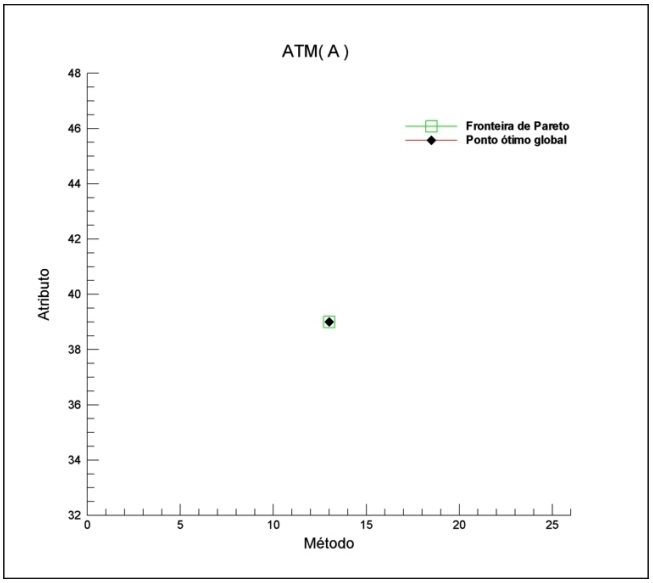

Figura 6.1 – Fronteira de Pareto para o Sistema ATM (A)

Observa-se na figura 6.2 que para o sistema B foram encontradas as soluções com um objetivo ótimo respectivamente para atributos na parte superior e para métodos na parte inferior dos extremos da fronteira de Pareto.

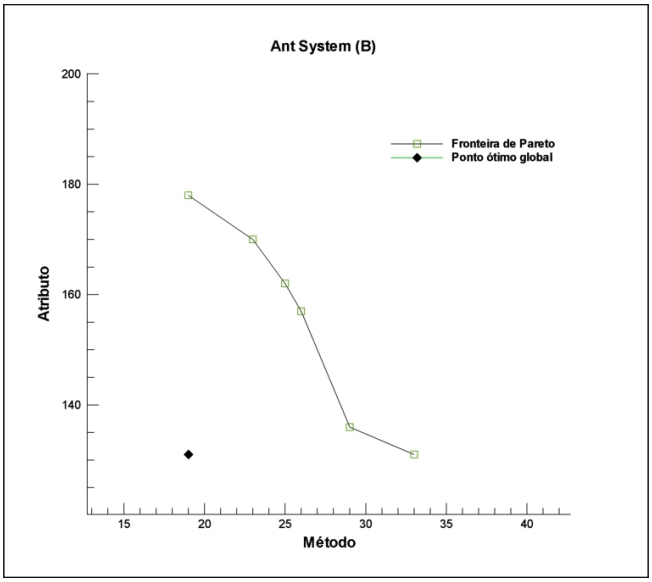

Figura 6.2 – Fronteira de Pareto para o Ant System (B)

Observa-se na Figura 6.3 que para o sistema C

duas soluções foram encontradas e as duas contém ótimos globais ou para métodos ou para atributos nos extremos da fronteira de Pareto.

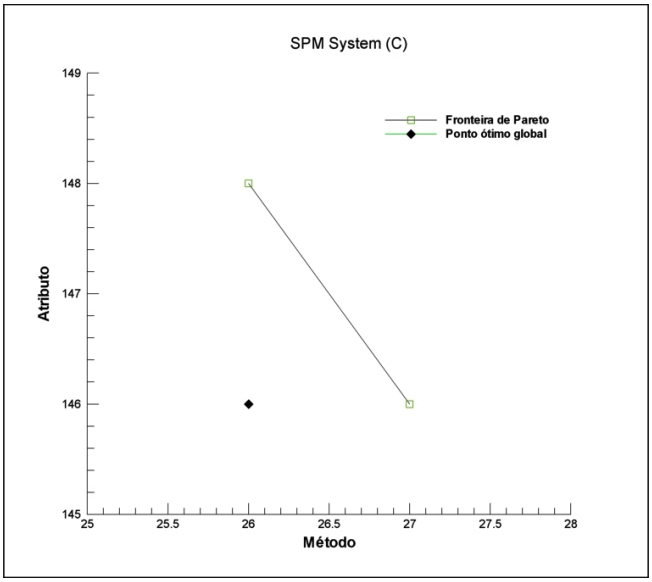

Figura 6.3 – Fronteira de Pareto para o SPM System (C)

Observa-se na figura 6.4 que os ótimos globais para cada objetivo foram encontrados e estão nos extremos e também que para este sistema há uma fronteira de Pareto um pouco mais diversificada já que o sistema D contempla um maior número de classes e também pois possui uma relação de complexidade maior entre os relacionamentos de métodos e atributos entre todos os sistemas.

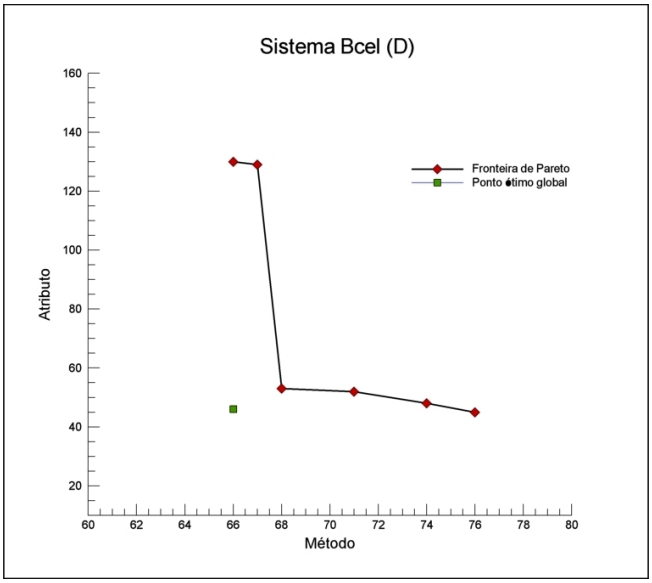

Figura 6.4 – Fronteira de Pareto para o Bcel (D)

Observa-se na figura 6.5 que os valores ótimos para métodos e atributos foram encontrados para o sistema E.

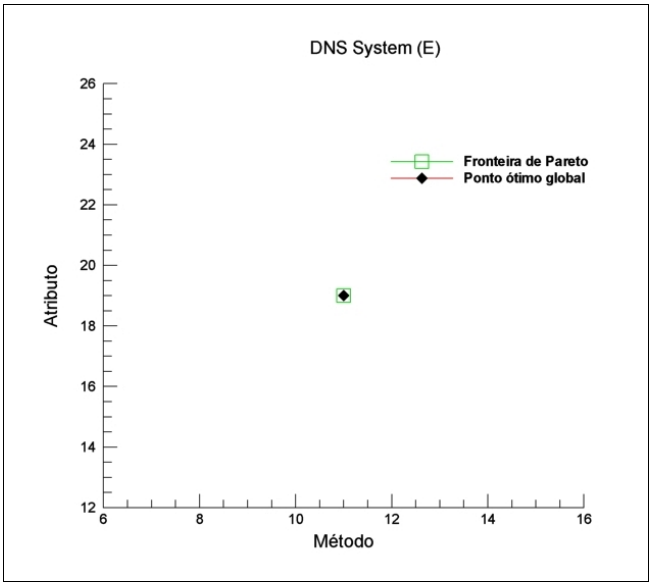

Figura 6.4 – Fronteira de Pareto para o DNS System (E)

7 - Conclusão

Para os sistemas A,C,E foram encontrados os ótimos globais em ambos os objetivos. Para o sistema B e D verificou-se que dois pontos da fronteira de Pareto continha cada um o valor ótimo para cada objetivo o que demonstra a eficiência do método PACO. Entretanto, originalmente o PACO utiliza uma busca local baseada em Pareto e está foi implementada mas os resultados obtidos não foram satisfatórios – os pontos da fronteira de Pareto para o sistema D não continha os valores ótimos para ambos os objetivos. O resultado apresentado na Seção 6 foi alcançado somente com substituição da busca local de Pareto com duas buscas locais para cada objetivo o que aumentou consideravelmente o custo computacional.

8 - Trabalhos Futuros

- Melhorar aspectos das busca local já que foram adicionada uma para cada objetivo e isso elevou o custo computacional.
- Experimentar outros tipos de busca local como *Iterated Local Search* e *Guide Local Search* afim de refinar ainda mais as soluções, o *Iterated* em especial, já que possibilita uma melhor exploração da fronteira de Pareto.
- Realizar comparativo para o problema resolvido no presente trabalho com uma implementação do BicriterionAnt e BicreterionMC [3].
- O algoritimo baseado em PAC implementado para resolução do problema de *Flow Shop Scheduling* utiliza o *Path relink* para vasculhar em vales entre as soluções da fronteira de Pareto [5]. Esta estratégia também poderia ser utilizada para buscar

mais soluções na fronteira de Pareto - sistema D figura 6.4.

9 – Referências

1. Dorigo, M., Maniezzo, V. Colorni, A., 1996 Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Trans. On System, Man, and Cybernetics Part B 26(1), 29-41*.

2. Dorigo, M. Gambrella, L. M., 1997. Ant Colonies for the Traveling Salesman Problem. *IEE Transactions on Evolutionary Computing, 1 (1), 53-66.*

3. Arca-Martnez, C. Cordon, O. Herrera F. 2004 An Empirical Analysis of Multiple Objective Ant Colony Optimization Algorithms for the TSP*. *Springer Berlin / Heidelberg*, 61- 72.

4. Briand, C., L., Feng, J., Labiche, Y., 2002 Experimenting with Genetic Algorithms and Coupling Measures to Devise Optimal Integration Test Orders. *Carleton University, Department of Systems and Computer Engineering. Technical Report SCE-02-03.*

5. Pasia, J., M., , Hartl, R,. F., Doerner K. F., Solving a Bi-objective Flowshop Scheduling Problem by Pareto-Ant Colony Optimization 2006. Springer Berlin / Heidelberg,

6. Briand L. C., Feng J., Labiche Y. Experimenting with Genetic Algorithms and Coupling Measures to Devise Optimal Integration Test Orders 2002 - Carleton University, Technical Report SCE-02-03

7. Sutlzle T., Hoos H. Improvments on the Ant System: Introucting MAX-MIN Ant System. TH Darmstadt, FB Informatik, FG Intellektik.