Computação Natural - Trabalho Prático 2
Um algoritmo baseado em Colônia de Formigas para o problema de Job-Shop Scheduling

Paulo Viana Bicalho[1]

[1]Departamento de Ciência da Computação – Universidade Federal de minas gerais (UFMG)

{p.bicalho}@dcc.ufmg.br

Resumo. *Este relatório descreve a implementação de um algoritmo de Colônia de Formigas para o problema de Job-Shop Scheduling (JSS). O JSS é um problema de otimização NP-Hard que consiste em alocar operações de tarefas à máquinas, de forma a minimizar o tempo de processamento necessário para terminar a execução de todas elas. O principal objetivo deste trabalho é o de entender e praticar conceitos e técnicas de algoritmos baseados em Colônia de Formigas e comparar os resultados encontrados com os resultados obtidos pelo algoritmo genético implementado no trabalho anterior.*

1. Introdução

O Job-Shop Scheduling (JSSP) é um problema de otimização que consiste em alocar um determinado número tarefas e suas operações à máquinas. A alocação deve ser realizada levando em consideração um conjunto de regras e restrições previamente estabelecidos.

As regras deste problema definem que nenhuma tarefa pode ser alocada duas vezes para uma mesma máquina. Cada máquina só pode processar uma operação por vez sendo que esta operação nao pode ser interrompida, e não existe nenhum tipo de dependência e relação entre operações de dois jobs diferentes.

Além destas regras, cada instancia do problema possui restrições próprias. Tais restrições informam o número de operações de cada tarefa, em qual máquina que cada operações deve ser executada e o tempo total de execução de cada operação.

Desta forma o problema do JSSP consiste em determinar a sequencia de execução de operações que, ao serem processadas, resultem em um menor tempo de processamento total (*Makespan*) total.

Esta documentação descreve a implementação de um algoritmo baseado em Colônias de Formigas (ACO) para o problema exposto.

O ACO como próprio nome dize, baseia-se no comportamento de colonias de formigas principalmente na tarefa de busca por alimentos.

Durante o inicio da busca por comida, as formigas começam a explorar o território de maneira aleatória. Quando encontram algum tipo de alimento, elas retornam para o formigueiro liberando feromônio no trajeto de regresso. Esta trilha de feromônio são interpretadas por outras formigas como um *feedback* positivo e, ao se depararem com uma destas trilhas, tendem a segui-la aumentando o reforço positivo.

Como os feromônios são substancias liberadas pelas formigas, ele evapora como o passar do tempo. Desta forma caminhos mais curtos tendem a possuir uma quantidade de maior de feromônio do que caminhos longos o que faz com que o sistema convirja.

Esta documentação esta organizada da seguinte forma: A seção 2 descreve como o ACO foi modelado para lidar com o problema do JSSP apresentando decisões de implementação, a seção 3 apresenta a logica principal proposta, os detalhes de implementação e execução do algoritmo desenvolvido são apresentados na seção 4, a seção 5.1 é responsável por analisar o impacto dos parâmetros no comportamento do ACO enquanto na seção 6 discute-se sobre os resultado obtidos comparando-os com os resultado do algoritmo genético implementado no trabalho prático anterior.

2. Modelagem

O ambiente em que as formigas caminham foi modelado como um grafo. Este grafo possui $(m \times n) + 1$ vértices onde n é igual ao número de tarefas e m igual ao número de operações de cada tarefa. Desta maneira, uma solução para o problema do JSSP seria uma caminho que percorra todos os vértices.

Estruturalmente este grafo é um grafo completo, mas devido as restrições do JSSP uma aresta entre duas operações i e j somente pode ser adicionada ao caminho se todas as operações anteriores da tarefa que possui a operação j já foram adicionadas ao caminho.

Foi adicionado um vértice a mais no grafo, o vértice **0** com a única finalidade de ser o ponto de partida das formigas.

2.1. Caminho

A formiga inicia o caminho a partir do vértice **0**. Este é um vértice especial que foi introduzido no grafo com a única finalidade de ser o ponto de partida das formigas. O próximo passo da formiga seria o de decidir qual o próximo vértice irá visitar. Ao tomar esta decisão, a aresta entre estes vértices é inserida no caminho.

Desta maneira o caminho é construído de forma incremental e a cada passo a formiga escolhe qual aresta irá seguir. Esta escolha e realizada de maneira tal que seja baseada na quantidade de feromônio depositada na aresta e na qualidade local de se escolher aquela aresta.

Pode-se encarar a qualidade local como sendo uma heurística para a solução do problema e a idéia de utilizar tal heurística é a de tentar combinar uma avaliação global com uma avaliação local.

2.2. Escolha

Como visto, a cada passo uma formiga tem que decidir qual aresta ira seguir. Esta escolha é uma escolha estocástica e é guiada pela probabilidade de se escolher cada aresta.

A probabilidade de se escolher a aresta ij é dada pela seguinte fórmula:

$$p_{ij} = \frac{(\tau_{ij})^{\alpha}(\eta_{ij}^{\beta}}{\sum\limits_{k \in N(i)} (\tau_{ik})^{\alpha}(\eta_{ik})^{\beta}}$$

Onde τ_{ij} representa a quantidade de feromônio na aresta, η_{ij} a qualidade local, $alpha$ e $beta$ são pesos e $N(i)$ é o conjunto de vizinhos do vértice i.

Note que ao fazer $(\tau_{ij})^{\alpha}$ se $\alpha > 1$ e $0 < \tau_{ij} < 1$, o resultado obtido será menor que τ_{ij}, por outro lado se $\tau_{ij} > 1$ o resultado será maior que τ_{ij}. Assim, para evitar comportamentos indevidos do programa, tanto τ quanto η são sempre normalizados para ficarem no intervalo entre [0,1].

Considerando γ o tempo atual da máquina onde a operação j deve ser executado e θ o tempo de término da operação $j-1$ da mesma tarefa a qual j pertence, a qualidade local de uma aresta ij é dada da seguinte forma:

$$\eta_{ij} = \begin{cases} \gamma - \theta & \text{if } \gamma > \theta \\ 0 & \text{caso contrário} \end{cases}$$

Ou seja, a melhor aresta, do ponto de vista local, ij é aquela que, ao ser inserida no caminho, irá deixar a máquina a qual a operação j está associada, ociosa pelo menor tempo possível.

2.3. Avaliação do caminho

Por decisão de implementação, a qualidade do caminho é avaliada a cada passo a medida que novas arestas são acrescentadas. Esta qualidade é medida conforme a função de fitness definida como o Makespan associado ao caminho. Desta maneira ao terminar a construção do caminho a fitness do mesmo ja foi calculada.

2.4. Atualização do Feromônio

A atualização de feromônio ocorre ao final de cada iteração do ACO e acontece em duas etapas principais:

1. Primeiramente realiza-se a evaporação do feromônio em cada aresta conforme a fórmula:

   $$\tau = (1 - \rho) \times \tau$$

   Onde ρ corresponde à taxa de evaporação do feromônio.
2. Realizado a evaporação, o caminho percorrido por cada formiga da iteração é avaliado e o feromônio de cada aresta no caminho é incrementado conforme a qualidade da solução:

   $$\tau = \tau + \frac{1}{makespan(formiga)}$$

   Onde $makespan(formiga)$ corresponde ao makespan final gerado pelo caminho percorrido pela formiga.

O algoritmo 1 sintetiza esta etapa do ACO.

Algorithm 1: Atualização do feromônio

```
for Para cada aresta ij no grafo do
    τ_ij ← (1 − ρ) * τ_ij;
end

for Para cada formiga k da iteração do
    for Para cada aresta ij no caminho percorrido pela formiga k do
        τ_ij = τ_ij + 1/makespan(formiga_k)
    end
end
```

3. ACO

O algoritmo 2 a seguir sintetiza o funcionamento de um ACO:

Algorithm 2: ACO

```
while i < NUM_IT do
    for j < NUM_ANTS do
        f ← novaFormiga();
        f → caminha()
    end
    atualizaFeromônio;
end
```

4. Implementação

A implementação do ACO foi realizada em c++, o código fonte acompanha esta documentação e esta estruturado da seguinte maneira:

- Constraints: Responsável por armazenar as restrições da instância do problema;
- Schedule: Implementa a classe que representa o schedule gerado a partir de um caminho percorrido pela formiga;
- ACO: Clase que implementa a logica principal do ACO.
- main: Modulo principal do programa;

4.1. Formato entrada

O algoritmo foi testado utilizando instancias disponíveis em: `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop1.txt`.Portanto o formato do arquivo de entrada é o mesmo descrito na página.

4.2. Formato Saida

A saída o programa ocorre na saída padrão do sistema e apresenta o seguinte formato:

```
Iteration 0: <melhorMakespanSempre> <melhorMakespan> <MakespanMédio> <PiorMakespan>
Iteration 1: <melhorMakespanSempre> <melhorMakespan> <MakespanMédio> <PiorMakespan>
Iteration 2: <melhorMakespanSempre> <melhorMakespan> <MakespanMédio> <PiorMakespan>
....
```

4.3. Execução

O ambiente de desenvolvimento escolhido foi o linux distribuição Ubuntu 12.04. Um Makefile acompanha o código para facilitar a compilação do mesmo.

Os parâmetros esperados pelo módulo principal do programa são:

1. arquivoEntrada: arquivo de entrada com uma instância do JSSP;
2. numFormigas: número de formigas;
3. numIter: número de iterações;
4. eRate: taxa de evaporação do feromônio;
5. alpha: peso para qualidade global;
6. beta: peso para qualidade local;

A chamada do programa se da da seguinte forma:

```
./ACO_JSS <arquivoEntrada:string> <numFormigas:int> <numIter:int>
   <eRate:double[0,1]> <alpha:int> <beta:int> <arquivoLog:string>
```

5. Análise Experimental

Para avaliar a qualidade do ACO proposto foram utilizados quatro instâncias do problema do JSSP. As instâncias foram retiradas do site `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop1.txt` e são elas:

1. la29: makespan ótimo = 1157;
2. la40: makespan ótimo = 1222;
3. ft06: makespan ótimo = 55;
4. la01: makespan ótimo = 666;

As duas primeiras instâncias foram utilizadas para calibrar os parâmetros do algoritmo. Todos os valores apresentados nesta seção são médias dos resultados de 30 execuções.

5.1. Parâmetros

Encontrar os valores de parâmetros mais adequados para se utilizar no ACO não é uma tarefa trivial. A qualidade da solução encontrada, normalmente, esta diretamente relacionado aos parâmetros e por isso deve-se realizar um sequência de testes como o intuito de determinar a melhor configuração.

Os parâmetros do ACO desenvolvido são: número de formigas, número máximo de iterações, taxa de evaporação, alpha e beta.

O número de formigas e de iterações definem quantas soluções serão geradas e avaliadas. De maneira geral, definem qual a capacidade máxima de exploração do conjunto de soluções poderá ser alcançada.

Os valores de alpha e beta ralacionan-se com a formula do calculo de probabilidade em cada passo da construção de um caminho. Tais valores regulam a importância do feromônio (solução global) e da solução local na escolha de uma aresta.

A taxa de evaporação controla a quantidade de feromônio que será ”evaporado“ a cada iteração. O objetivo é fazer como que o feromônio, que representa a qualidade

global, diminua com o passar tempo caso a aresta não seja visitada recorrentemente. A intuição por trás deste parâmetro é que com o passar das iterações, caminhos melhores são visitados com mais frequência do que caminhos piores e por isso deve existir um mecanismo que redução do feromônio com o passar das iterações.

O valor ideal de cada parâmetro apresentado varia muito de acordo com cada instância do problema e, em geral, determinar estes valores ideais torna-se uma tarefa quase que inviável pois pode envolver a realização de testes com todas as combinações de valores possíveis. Desta forma, geralmente tenta-se realizar pequenas combinações a fim de se encontrar uma configuração que alcance um resultado aceitável.

As subseções a seguir detalham os testes realizados com esta finalidade e os resultados obtidos. Todos os valores apresentados são médias de 30 execuções.

5.1.1. Número de Formigas e Número de Iterações

Foram testados combinações de valores destes dois parâmetros da seguinte forma:

<u>Valores</u>
Número de formigas: [25,50,75,100];
Número de iterações: [50,150,300,500];
Alpha: 1;
Beta: 2;
Taxa de Evaporação: 0.05;

Os resultados obtidos são apresentados na figura 1

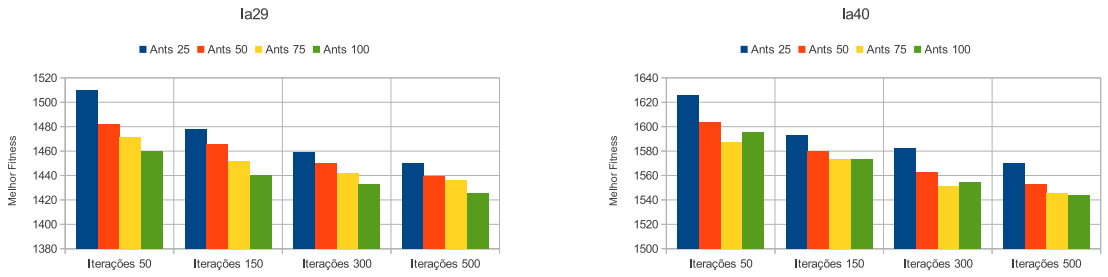

Figure 1. Variando número de formigas e iterações

O número de soluções (caminhos) avaliados pelo algoritmo é dado pela multiplicação destes dois parâmetros então, a medida que estes são aumentados o ACO é capaz de encontrar soluções melhores pois pode-se dizer que explorou mais o espaço de busca. Desta forma os valores utilizados para estes parâmetros para os demais testes são

<u>Melhores Valores</u>
Número de formigas: **100**;
Número de iterações: **500**;

5.1.2. Alpha e Beta

Valores
Número de formigas: 100
Número de iterações: 500
Alpha: [1,2,3,4]
Beta: [1,2,3,4]
Taxa de Evaporação: 0.05

A calculo da probabilidade de se seguir por uma determinada aresta é dado pela fórmula:

$$p_{ij} = \frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum_{k \in N(i)} (\tau_{ik})^{\alpha}(\eta_{ik})^{\beta}}$$

Onde τ é a quantidade de feromônio na aresta e η representa a qualidade da local desta. Ambos valores estão no intervalo de [0,1] e por isto a medida que se aumentam os valores de α e β diminui-se a importância de τ e η respectivamente.

A figura 2 apresenta os resultados obtidos.

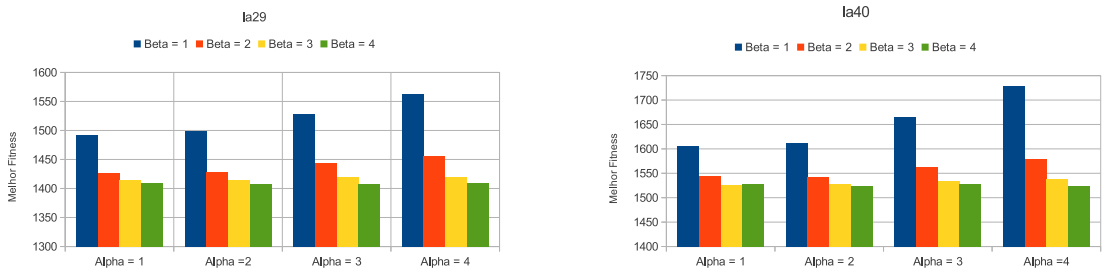

Figure 2. Variando alpha e beta

Percebe-se que β impacta mais na qualidade da solução do que α e quanto maior o seu valor, melhor a solução. Isto indica que a definição da qualidade local (ver seção 2.2) pode não ter sido a melhor possível. A melhor configuração encontrada, que será utilizada nos demais testes é apresentada a seguir:

Melhores Valores
Alpha: **2**;
Beta: **4**;

5.1.3. Taxa de Evaporação

Valores
Número de formigas: 100
Número de iterações: 500

Alpha: 2
Beta: 4
Taxa de Evaporação: [0.001,0.01,0.05,0.10,0.50]

A taxa de evaporação pode ser entendida como o tempo de uma espécie de memória de feromônio. Se uma aresta ficar muito tempo sem ser visitada, o feromônio será reduzido podendo chegar a zero.

A figura 3 apresenta os valores obtidos para as duas bases. Percebe-se que em ambos os casos não existe uma logica do tipo, ”a medida que aumenta (ou diminui) a taxa a solução melhora“. Isto mostra que uma evaporação muito rápida ou muito lenta não são boas para o ACO.

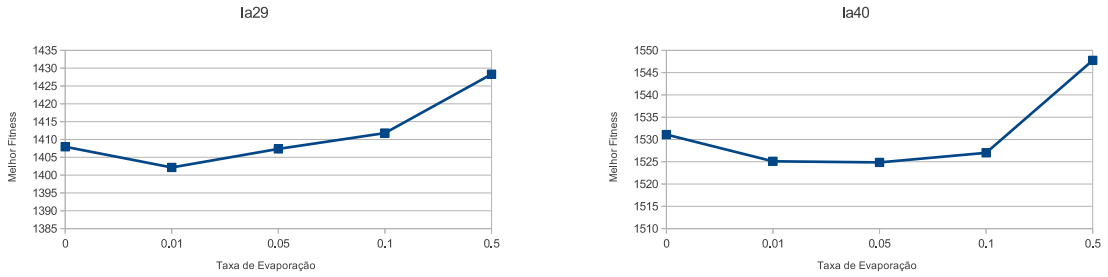

Figure 3. Variando a taxa de evaporação

O melhor valor encontrado e que será utilizado nos demais testes será de:

<u>Melhor Valor</u>
Taxa de evaporação: **0.05**;

5.2. Análise dos resultados

Realizado a escolha dos parâmetros, o ACO foi testado nas quatro instâncias com o intuito de se verificar qual o melhor valor de makespan que o algoritmo foi capaz de encontrar. A tabela 1 resume os resultados obtidos.

Instância	Makespan Ótimo	ACO		Algoritmo Genético	
		Makespan	Vezes que encontrou o ótimo	Makespan	Vezes que encontrou o ótimo
ft06	55	55	27	55	20
la01	666	666	30	666	10
la29	1157	1376	0	1331	0
la40	1222	1476	0	1368	0

Table 1. Resultados Gerais

A tabela 1 também exibe os resultados obtidos pelo Algoritmo Genético (GA).

Para as bases *ft06* e *la01* o ACO foi capaz de encontrar a solução ótima mais vezes que o GA. Para as outras duas, embora nenhum método tenha chegado ao ótimo, o GA encontrou soluções bem melhores que o ACO.

Como as primeira instâncias são mais simples, pode-se dizer que o GA é uma solução mais interessante para instâncias mais complexas do Job-Shop-Scheduling.

6. Conclusão

Esta documentação descreveu a implementação de um Algoritmo Baseado em Colônia de Formigas (ACO) para a solução do problema do Job-Shop-Scheduling (JSSP). O algoritmo proposto foi testado em quatro instâncias do problema e os resultados foram avaliados.

Os objetivos do trabalho foram alcançados e os problemas e dificuldades de se projetar e desenvolver um ACO foram superados satisfatoriamente.

Como na maioria dos algoritmos em computação natural, a escolha dos parâmetros deve ser realizada da melhor maneira possível, uma vez que a variação destes acarretam em grandes impactos na qualidade das soluções encontradas.

As soluções obtidas pelo método para as duas instâncias de teste mais complexas não foram muito boas, quando comparadas com o Algoritmo Genético (AG) implementado no trabalho anterior.

Tal constatação leva a conclusão de que o AG pode ser uma solução mais interessante para este problema que o ACO, uma vez que, geralmente em situações reais, as instâncias do JSSP são complexas.