Paralelização do Algoritmo Colônia de Formigas para Resolução do Problema 15-Puzzle

Pedro V. Q. Rocha

[1]Ciência da Computação – Universidade Estadual De Maringá (UEM)
Departamento de Informática – PR - Brasil

ra68740@uem.br

Resumo. *Este artigo tem como objetivo a paralelização de um algoritmo que aplica Colônia de Formigas para resolução do problema 15-Puzzle. A métrica de desempenho é feita através de gráficos de Speedup, eficiência e tabelas com resultados de experimentos com 2, 4 e 8 threads. O trabalho irá expor também os problemas e dificuldades em utilizar técnicas de programação paralela para resolver o problema apresentado.*
Palavras-chave*: colônia de formigas, n-puzzle, algoritmo paralelo.*

Abstract. *This article aims at parallelizing an algorithm that applies Any Colony Optimization to solve the 15-Puzzle problem. The performance is measured by Speed Up and Efficiency charts and tables with the experiments results using 2, 4 and 8 threads. This work also exposes the difficulties and problems of using parallel programming techniques to solve the presented problem.*
Keywords*: ant colony system, n-puzzle, parallel algorithm.*

1. Introdução

É comum na área da ciência tentar reproduzir comportamentos da natureza para solucionar problemas computacionais, tais meios são chamados de Computação Natural. Esta nomenclatura é usada na literatura para descrever todos os sistemas computacionais implementados com inspiração de algum mecanismo natural ou biológico processador de informação [De Castro].

O algoritmo tratado neste artigo, *Ant Colony Optimization* (ACO), é um exemplo de algoritmo inspirado na natureza e utiliza de técnicas desenvolvidas durante milhões de anos a favor da computação para resolver problemas. O 15-Puzzle já foi resolvido por [Culberson et al. 1994] usando *Iterative Deepening Improvement A** (IDA*) e por [Harsh Bhasin 2012] usando Algoritmos Genéticos. A técnica utilizada neste trabalho será a meta-heurística *Ant System*.

1.1. Colônia de Formigas

O algoritmo de Colônia de Formigas é baseado no comportamento cooperativo de formigas reais, ele simula um grupo de formigas agindo em conjunto para atingir um objetivo. Formigas se comunicam através de uma subtância chamada feromônio, também presente no algortimo. A trilha de feromônio é o que dita quais os possíveis caminhos para o objetivo proposto. As formigas depositam a substância conforme percorrem um caminho válido, guiando as formigas subsequentes. A quantidade depositada é relacionada

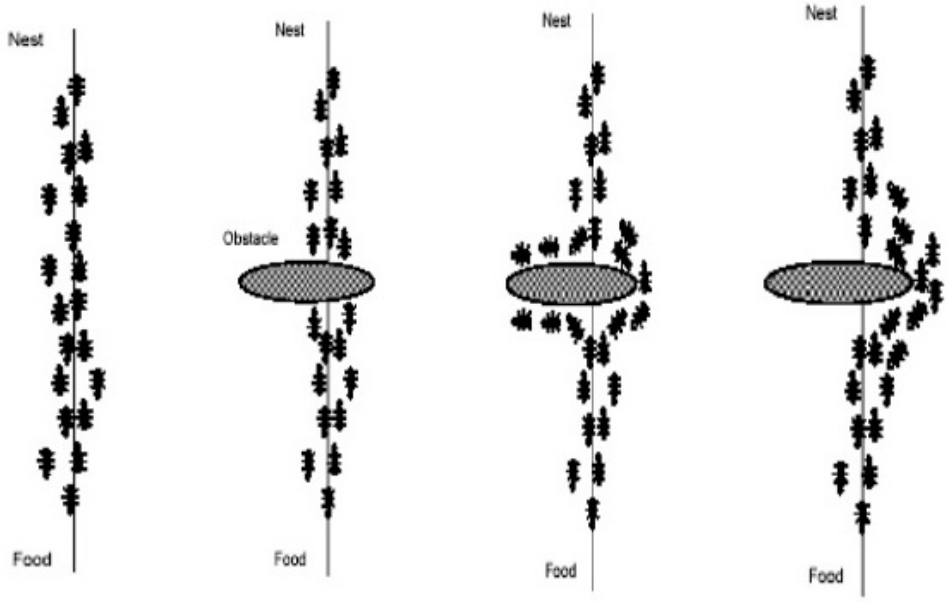

Figura 1. Caminho das formigas sendo definido através da trilha de feromônios

com a qualidade do caminho percorrido, fazendo assim com que melhores caminhos se destaquem e tenham probabilidade de atrair mais formigas, como mostra a Figura 1.

O feromônio é uma substância volátil, ou seja, evapora em uma determinada taxa. A notação adotada para esta taxa de evaporação é ρ (rho). Esta característica é importante pois faz com que caminhos inúteis ou menos atraentes desapareçam com o tempo.

No algoritmo, este processo pode ser descrito como um grupo de agentes cooperando para encontrar a solução de um problema, no caso, a configuração final do 15-Puzzle. A paralelização do algoritmo permite dividir grupos de formigas que procurarão soluções ao mesmo tempo, propondo aumentar o desempenho e eficiência do método dependendo da quantidade de *threads*.

1.2. O problema 15-Puzzle

O problema 15-Puzzle inventado por Sams Loyd é comumente usado para testar algoritmos baseados em heurísticas devido à sua fácil implementação e por sua configuração final ser checada em tempo constante.

O problema consiste em um tabuleiro 4x4 com 16 espaços e 15 peças. O objetivo do jogo é, partindo de uma configuração aleatória válida, chegar a uma configuração final deslocando apenas uma peça por vez para o espaço vazio, representado pelo 0 no tabuleiro. Neste caso, a configuração final é uma matriz com os números em ordem crescente em sequência, demonstrado abaixo.

$$TabuleiroInicial = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 0 & 15 \end{bmatrix}$$

$$TabuleiroFinal = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 0 \end{bmatrix}$$

2. Trabalhos Relacionados

Regina propõe usar de Computação Natural em seu trabalho [Regina 2007] para solucionar problemas computacionais. Para isso, utiliza o algoritmo de Colônia de Formigas para resolução do problema 8-Puzzle, uma versão menor do 15-Puzzle. O problema é modelado de forma similar a este artigo, no entanto, o espaço de possibilidades do 8-Puzzle é de $\frac{9!}{2}$ (181 mil) enquanto o 15-Puzzle é de $\frac{16!}{2}$ (10 trilhões), uma diferença exorbitante.

Esta diferença na quantidade de possibilidades é de extrema importância, pois o número de ciclos e formigas necessários para achar uma solução é muito menor no caso do 8-Puzzle. Por esse motivo, as formigas geram uma solução inicial mais rapidamente, consequentemente a trilha de feromônios é atualizada e a solução é encontrada sem necessidade de muitas iterações e sem *starvation*.

Como a árvore de possibilidades é pequena, o tempo gasto em cada ciclo não aumenta conforme a progressão do algoritmo, pois a árvore estará completamente aberta após poucas iterações e a trilha para uma boa solução já definida. No caso do 15-Puzzle, a árvore de possibilidades é aberta gradativamente pelas formigas conforme os ciclos, e, mesmo com uma grande quantidade de ciclos, não será aberta completamente, aumetando o tempo de execução e a dificuldade em achar uma solução.

3. Referencial teórico

3.1. Modelagem do Problema

A árvore de possibilidades usada no algoritmo contém todos os caminhos já percorridos pelas formigas e também os caminhos que foram abertos mas ainda não visitados. A atualização do feromônio feita pelas formigas a cada final de iteração é realizada nesta árvore global, que será utilizada para as formigas do ciclo seguinte.

A árvore aumenta a cada vez que uma formiga visita um nó sem filhos. Para cada nó, temos de dois a quatro novos nós a serem gerados, como mostrado na Figura 2.

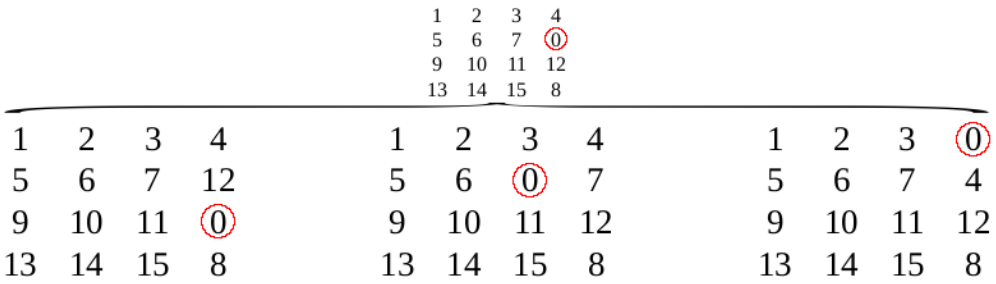

Figura 2. Node pai gerando 3 filhos

A árvore de possibilidades aumenta exponencialmente a cada troca de peças, completando no final $16!$ total de configurações, das quais apenas metade são válidas, resultando em aproximadamente 10^{13} configurações. Em uma formulação geral do problema, a extensão do *puzzle* é n x k e o problema de decisão que envolve saber se a combinação k de movimentos para levar a solução ótima existe é NP-Completo.

Para resolver este problema de forma mais eficiente não é recomendado o uso de uma busca cega, mas sim algoritmos baseados em heurísticas. Durante a implementação do algoritmo foram desenvolvidas três heurísticas: Peças no Lugar, Distância Manhattan e uma Heurística Combinada.

A heurística de Peças no Lugar, como o nome já diz, indica quantas peças do tabuleiro estão na posição correta comparadas a configuração final, sendo 0 todas fora do lugar e 16 quando o tabuleiro é igual à solução.

A Distância Manhattan indica a somatória do valor absoluto da distância retangular de uma peça para o seu lugar correto, sem obstáculos. Para calcular o valor da heurística considere A como a matriz avaliada e B como a matriz solução.

$$DM(A,B) = \sum_{x=0}^{16} \left(\left| dist\left(i_x^A, i_x^B\right) \right| + \left| dist\left(j_x^A, j_x^B\right) \right| \right)$$

A última heurística é a soma das duas anteriores multiplicadas por um peso definido, neste caso, 50% para cada um.

$$HC = (DM(A,B) * 0.5) + (Pe\text{ç}asNoLugar * 0.5)$$

Com o valor da heurística definido, as formigas podem caminhar pela árvore de possibilidades escolhendo probabilisticamente caminhos a serem adicionados para formar sua solução ou, em alguns casos, morrer de fome, também chamado de *starvation*.

Uma formiga pode morrer de fome enquanto procura alimento por muito tempo em um ambiente real. Analogamente, *starvation* ocorre no algoritmo quando uma formiga procura uma solução por muito tempo sem sucesso.

3.2. Feromônio e Caminhos

O feromônio é um dos principais componentes da meta-heurística de *Ant System*. Após a formiga percorrer um caminho para achar uma solução, ela depositará uma quantidade de feromônio equivalente a qualidade deste caminho, a qualidade é definida de acordo com a quantidade de movimentos feitos para chegar a solução. A cada ciclo de execuçã o do algoritmo, caminhos com mais feromônios vão se destacando, enquanto caminhos de qualidade inferior somem, chegando a um ponto em que a maioria das formigas convergem para um único caminho, que seria supostamente a melhor solução encontrada. Equação de atualização do feromônio

$$Fer_{ij}^A = Fer_{ij}^A * (1 - \rho) + \Delta C$$

O feromônio da formiga A no caminho ij é acumulativo e influenciado por ΔC e ρ. ΔC é a relação que dita a qualidade do caminho, expressa por

$$\Delta C = \frac{16}{nMovimentos}$$

Quanto maior a quantidade de movimentos, menor a qualidade do caminho achado e consequentemente a quantidade de feromônio depositado pela formiga.

Como a escolha do caminho é feita de forma probabilística, nem sempre a formiga escolherá o melhor caminho. A cada iteração do algoritmo o valor do feromônio de cada parte do caminho é alterado, formando novas trilhas e possíveis soluções. A escolha do caminho é feita usando uma Seleção por Roleta. Este método é usado pois faz uma distribuição normal entre as possibilidades de cada novo caminho a ser escolhido de acordo com seu peso. A equação de probabilidade depende, além do feromônio do caminho e do valor da heurística, de duas variáveis dos paramêtros do algoritmo, α e β.

$$Prob_{ij} = \frac{\left[Fer_{ij}^{A}\right]^{\alpha} * \left[valorHeuristica_{ij}^{A}\right]^{\beta}}{\sum\left(\left[Fer_{ij}^{A}\right]^{\alpha} * \left[valorHeuristica_{ij}^{A}\right]^{\beta}\right)}$$

Sendo α o peso dado ao valor do feromônio no caminho e β o peso dado ao valor da heurística no caminho ij da formiga A.

A roleta usa a equação acima para escolher probabilisticamente qual o próximo caminho a ser seguido pela formiga.

4. Desenvolvimento

O primeiro passo da implementação foi desenvolver o algoritmo sequencial. O algoritmo segue o pseudo código abaixo.

4.1. Pseudo-Código do *Ant Colony* Sequencial

```
1  int antsystem() {
2    inicializaArvore();
3    enquanto (ciclos != max_ciclos) {
4        para (todas as formigas) {
5            inicializaFormiga();
6            geraSolucao();
7        }
8        para (todas as formigas) {
9            comparaMelhorSolucao();
10           atualizaFeromonioCaminho();
11       }
12       operacoesSecundarias();
13       atualizaFeromonioGlobal();
14   }
15   retorna melhorSolucaoEncontrada;
16 }
```

A função *geraSolução()* (linha 6) é divida em duas etapas: abrir nós na árvore global e escolher nó para adicionar em seu caminho. A seleção é feita baseada na equação de probabilidade por uma Roleta, como explicado na Seção 3.2. Para gerar um novo nó na árvore é preciso checar se ele já existe, a operação mais custosa do algoritmo inteiro, cerca de 93% do tempo total de execução.

4.2. Paralelização

Para a paralelização do algoritmo, primeiro foram identificadas todas as regiões críticas. No geral, a maior parte das regiões críticas é localizada dentro da funçao da linha 6, *geraSolução()*. Isso se deve ao fato de que toda vez que uma formiga gera um novo nó, o mesmo deverá ser adicionado na árvore global. Este evento é determinante para os resultados dos testes. Como essa função não pode ser acessada por duas formigas

simultaneamente, com risco de ambas precisarem adicionar um novo nó na árvore global, a geração de nós inteira é considerada uma região crítica. Isto quer dizer que a função mais custosa do algoritmo é constantemente trancada pelas formigas, comprometendo o desempenho do algoritmo quando executado em paralelo.

A operação de comparação da linha 9 é realizada usando uma variável *melhorLocal*. Cada *thread* produz uma solução local e apenas no final do último ciclo essa solução será comparada com a melhor global pelas threads, portando essa região só será região crítica no final do algoritmo. Como a operação de atualização de solução final é feita com uma variável global, é necessário trancar, modificar e destrancar a variável.

Há operações que só precisam ser executadas uma vez a cada ciclo, como atualização do feromônio global e contagem de ciclos. Para não criar uma região crítica e evitar condições de corrida, apenas uma formiga é responsável por tais operações.

Para garantir a sincronia das *threads*, duas barreiras são utilizadas. A primeira é localizada antes das operações de atualização de feromônio, para que quando o feromônio for atualizado, todas as formigas estejam na mesma etapa do algoritmo. A segunda barreira fica logo após a atualização de feromônio global e operações secundárias. O motivo da segunda barreira é simplesmente sincronizar as *threads* novamente após a *thread* responsável por operações secundárias e atualização de feromônio global terminar suas ações, finalizando assim um ciclo completo.

5. Resultados

Os testes foram analisados usando as informações da ferramenta *perf* e executados em um computador com as seguintes configurações:

-Sistema Operacional: Ubuntu 16.04 LTS 64-bit
-Memoria: 7,8GiB
-Processador: Intel Core i7-2600 CPU @ 3.40GHz x 8

Os testes escolhidos foram baseados na complexidade e tempo de execução. A complexidade no problema 15-Puzzle é definida de acordo com a quantidade de movimentos. O primeiro teste é de 18 movimentos para solução ótima, o segundo é de 20 e o terceiro 24 movimentos, todos tem um tempo de execução de 9 a 11 minutos. Os parametros usados no algoritmo foram testados e ajustados para chegar em valores satisfatórios dentro do intervalo de tempo definido.

Cada teste foi executado usando 1 (sequencial), 2, 4 e 8 *threads*, o objetivo era analisar a eficiência e o *Speedup* de cada caso. Os resultados exibidos são valores da média entre três execuções. As tabelas e gráficos abaixo mostram os resultados obtidos, assim como dados extraídos após a execução do caso de teste.

Para estes testes os parâmetros do algoritmo foram:
α = 0.10
β = 1.00
ρ = 0.50
Formigas = 512
Ciclos = 6

Tabela 1. Resultados para o caso de teste 18 movimentos

	Solução	Tempo	Árvore	Branch-Misses	Falhas de Paginação
Sequencial	18 mov.	550s	2907166 nós	4.86%	151.262
2 Threads	20 mov.	618s	2944582 nós	4.84%	153.180
4 Threads	18 mov.	631s	2917502 nós	5.07%	151.801
8 Threads	18 mov.	640s	2940467 nós	5.21%	153.014

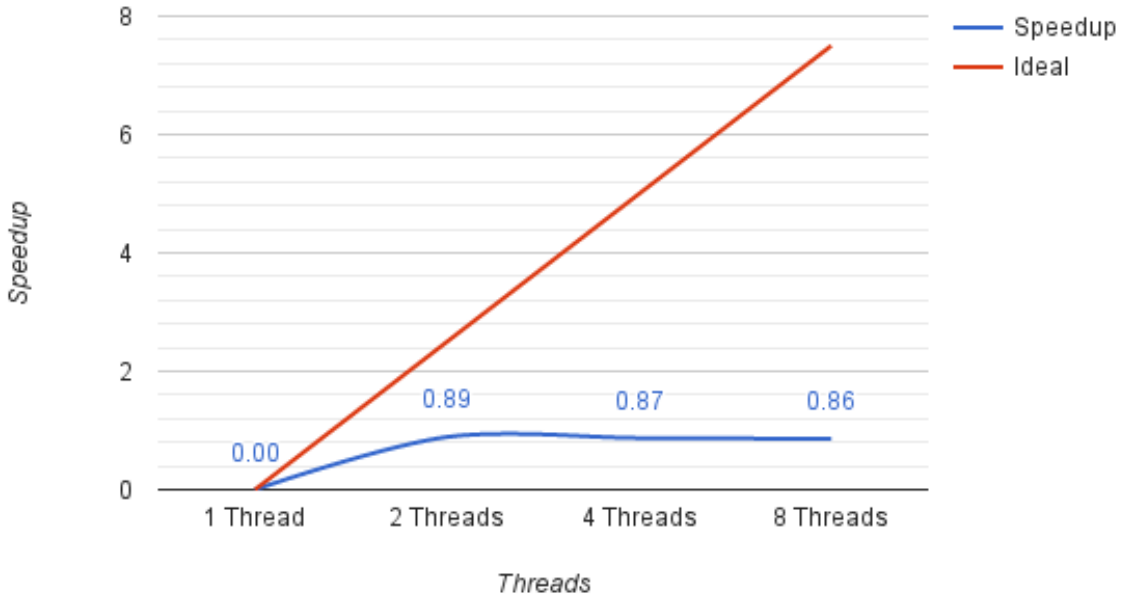

Figura 3. Gráfico de Speedup para 18 movimentos

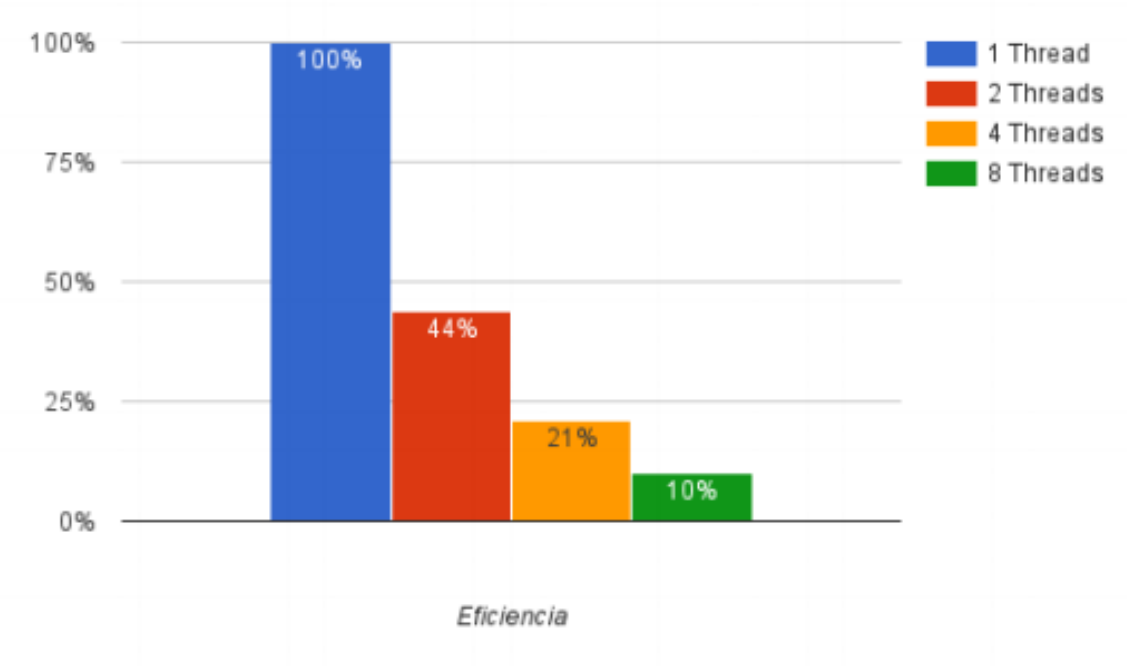

Figura 4. Gráfico de Eficiência para 18 movimentos

Tabela 2. Resultados para o caso de teste 20 movimentos

	Solução	Tempo	Árvore	Branch-Misses	Falhas de Paginação
Sequencial	20 mov.	540s	2922778 nós	4.16%	152.051
2 Threads	20 mov.	578s	2911471 nós	4.20%	151.500
4 Threads	22 mov.	604s	2874574 nós	4.51%	149.524
8 Threads	20 mov.	615s	2891471 nós	4.48%	150.541

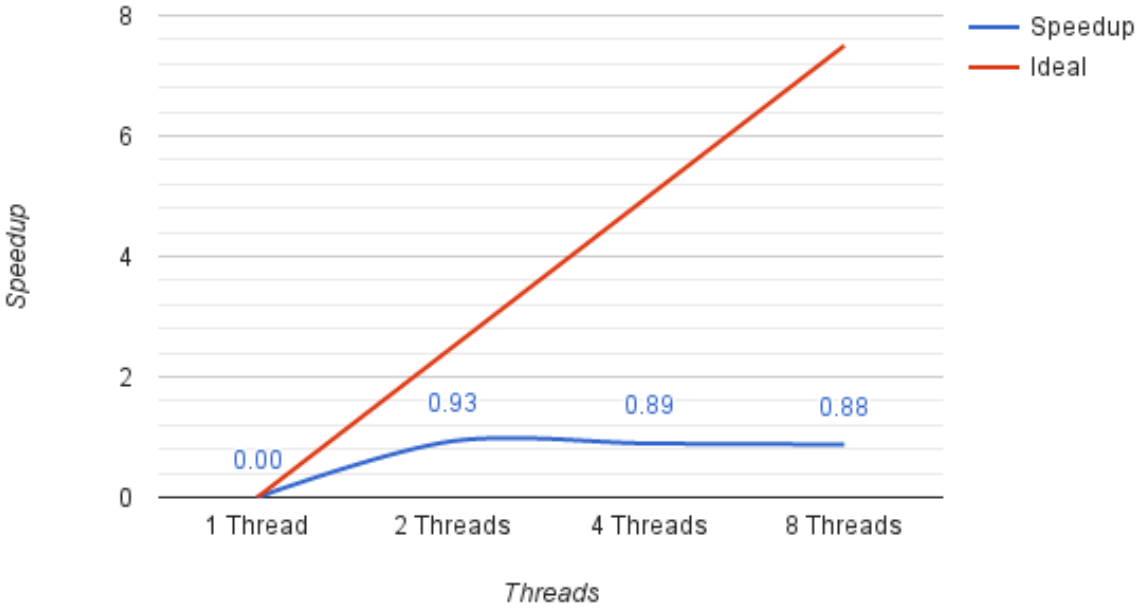

Figura 5. Gráfico de Speedup para 20 movimentos

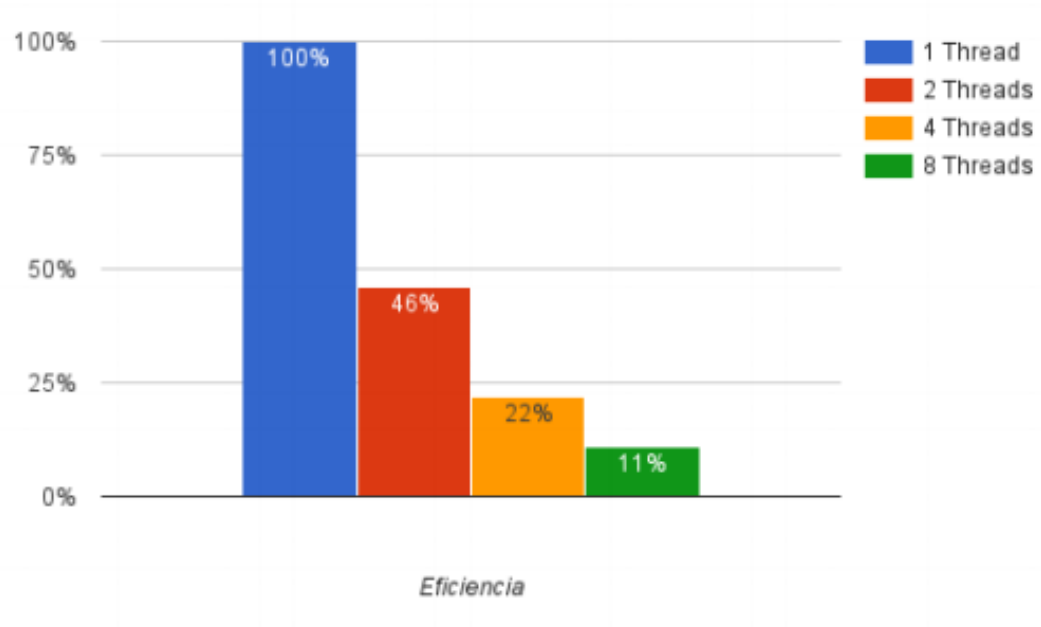

Figura 6. Gráfico de Eficiência para 20 movimentos

Tabela 3. Resultados para o caso de teste 25 movimentos

	Solução	Tempo	Árvore	Branch-Misses	Falhas de Paginação
Sequencial	43 mov.	533s	2911011 nós	4.21%	151.531
2 Threads	49 mov.	556s	2908577 nós	4.26%	151.551
4 Threads	41 mov.	622s	2925920 nós	4.67%	152.258
8 Threads	37 mov.	615s	2907309 nós	5.21%	151.330

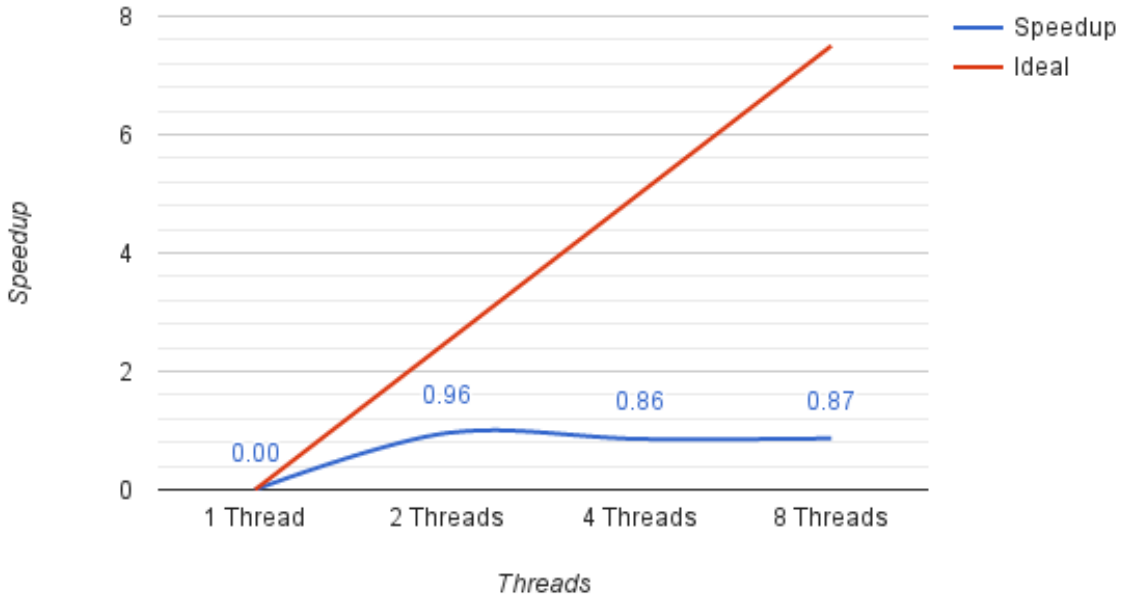

Figura 7. Gráfico de Speedup para 25 movimentos

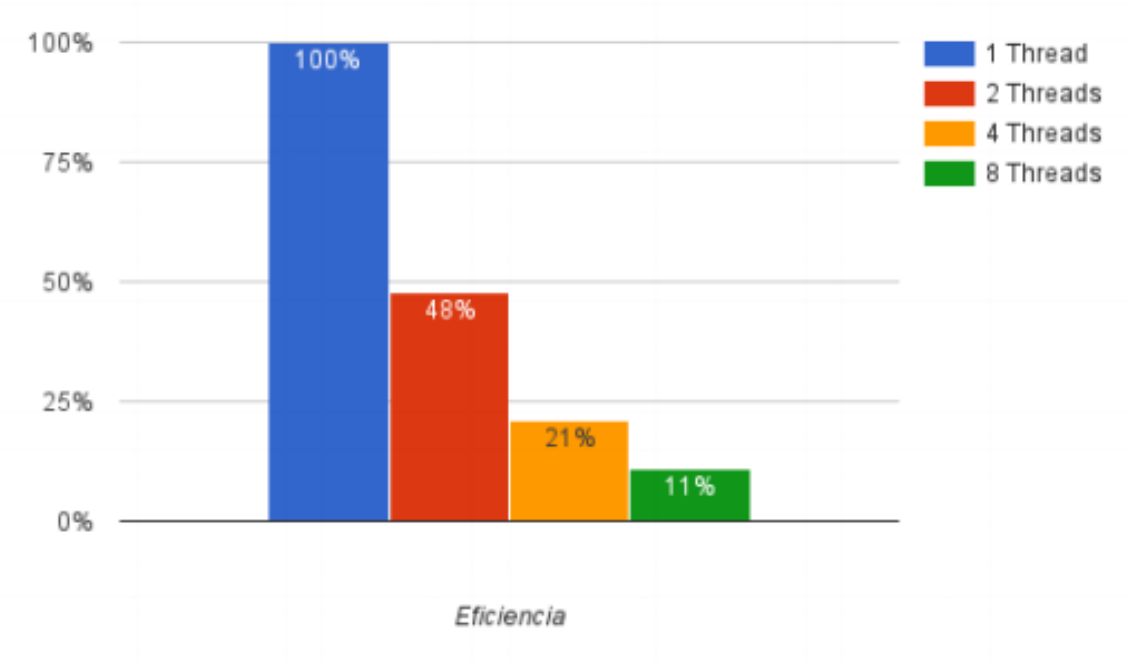

Figura 8. Gráfico de Eficiência para 25 movimentos

Em todos os testes executados a média de *stalled-cycles-frontend* foi de 83,38% e a de *stalled-cycles-backend* foi de 66,75%. Usando a ferramenta *valgrind* também foi calculado a média de *misses* na cache, que foi de aproximadamente 6%. Como explicado na Seção 4.2, o tempo de execução gasto apenas na região crítica da função que gera as possibilidades e adiciona na árvore é muito alto, a Tabela 4 demonstra os resultados de testes feitos para se obter uma porcentagem de tempo total.

Tabela 4. Tempo total gasto em uma região crítica gerando nós da árvore

	1 Thread	2 Threads	4 Threads	8 Threads
Tempo Total	524s	612s	626s	624s
Tempo Gerando Nós	519s	601s	619s	617s
Porcentagem	99%	98%	98%	98%

6. Considerações Finais

Com base nos resultados obtidos foi concluído que o algoritmo de Colônia de Formigas não é ideal para resolução do problema 15-Puzzle. A construção de soluções para casos complexos é lenta e a trilha de feromônios de um bom caminho demora a ser definida e suficientemente significativa para a maioria das formigas seguirem. Ocorre também muitos casos de *starvation*, dificultando o funcionamento do algoritmo em geral.

Como apresentado nos resultados, em nenhum dos casos de teste o *Speedup* foi satisfatório. O algoritmo é composto por um método que, constantemente, usa uma variável global para progredir, dificultando o funcionamento do mesmo em execução em paralelo. Uma mudança na implementação do algoritmo - como gerar soluções possíveis e abrir a árvore antes de iniciar os procedimentos das formigas - resolve este problema, mas o funcionamento do *Ant System* também seria alterado, já que a construção de soluções não ficaria totalmente a cargo das formigas. Mesmo que não recomendado, é possível resolver casos pequenos usando o algoritmo sequencial, mas a paralelização do mesmo o deixa mais eficiente.

Referências

Culberson, J., Culberson, J. C., Schaeffer, J., and Schaeffer, J. (1994). Efficiently searching the 15-puzzle. Technical report.

De Castro, L. *Computação Natural - Uma Jornada Ilustrada*. Livraria da Fisica.

Harsh Bhasin, N. S. (2012). Genetic based algorithm for n – puzzle problem. *International Journal of Computer Applications*, 51(22).

Regina, R. (2007). Aplicação do algoritmo “colônia de formigas” na resolução do 8-puzzle. Technical report, Faculdade de Jaguariúna.