Algoritmo Heurístico Otimização de Colônia de Formigas e Programação Paralela para a resolução do Problema do Máximo Conjunto Independente

Marco Aurélio Deoldoto Paulino[1]

[1]Departamento de Informática – Universidade Estadual de Maringá (UEM)
Maringá – PR – Brazil

marco23_aurelio@hotmail.com

Abstract. *This article aims in developing parallelized algorithm to find viable solutions to the Maximum Independent Set Problem(MISP) using heuristic algorithm with meta-heuristic Ant Colony Optimization by [Dorigo et al. 1996]. To verify the efficiency of the parallel code produced in comparison to a sequential version of the same is performed. From the data obtained a reflection on the effectiveness in the use of multiple threads to solve large problems it will be done.*

Resumo. *Este artigo tem como objetivo na elaboração de algoritmo paralelizado para encontrar soluções viáveis ao Problema do Máximo Conjunto Independente(MISP) utilizando algoritmo heurístico com a meta-heuristica Otimização de Colônia de Formigas introduzida por [Dorigo et al. 1996]. Para verificar a eficiência do código paralelo produzido, será realizada a comparação com uma versão sequencial do mesmo. A partir dos dados obtidos será feito uma reflexão sobre a efetividade na utilização de múltiplas threads para a resolução de problemas de grande porte.*

Introdução

O Problema do Máximo Conjunto Independente, traduzido de Maximum Set Independent Problem (MISP), é um problema na Teoria dos Grafos cujo objetivo é encontrar o maior conjunto possível de vértices que não possuem arestas entre si. O MISP pode ser considerado um problema de importância para a computação devido a sua aplicabilidade a várias áreas, tais como, Reconhecimento de Padrões, Escalonamento,Biologia Molecular e *Map Labeling*. Neste artigo iremos propor uma solução para o MISP utilizando a meta-heurística Otimização de Colônia de Formigas e programação paralela. Para fins de avaliação de perfomance, será produzida uma versão sequencial da solução.

A meta-heurística de propósito geral Otimização de Colônia de Formigas, do inglês, Ant Colony Optimization (ACO), introduzido por [Dorigo et al. 1996], se baseiou no comportamento das colônias de formigas para a elaboração da meta-heurística. As questões observadas foram , como animais praticamente cegos conseguem chegar em seus destinos, como funciona a substância química feromônio que utilizam para demarcar o caminho. Porém a representação da formiga e do feromônio não é necessariamente fiel a realidade, por exemplo, o tempo é representado discretamente e o feromônio pode ser atualizado ao final do caminho. O ACO se propõe a resolver aos mais variados problemas,

tais como, os problemas do Caixeiro Viajante, Coloração de Grafos e Mochila Binária e Mochila Fracionária.

Utilizado neste trabalho, o algortimo em paralelo vem com o objetivo de utilizar toda a potência de processadores multicores, visto que ao dividir o fluxo de execução em vários fluxos, permite terminar o problema em menor tempo, além de utilizar todo o potencial de sua máquina. Por outro lado, é necessário haver maior controle e atenção para a produção de códigos paralelos corretos, principalmente que a criação de códigos paralelos é de maior dificuldade que códigos sequenciais, visto que a grande maioria dos programadores estão acostumados a apenas programar sequencial e a elaboração de código paralelo possue maior complexidade em relação aos códigos sequenciais.

Para a confeção do código em paralelo será utilizado a linguagem de programação C com a biblioteca pthread. Será utilizada a abordagem de memória compartilhada, onde os dados serão visíveis a todos os fluxos, portanto será necessário o gerenciamento de sincronismo, seja por semáfogos, barreiras, monitores ou *lock*. Há disponíveis algumas linguagens desenvolvidas no paradigma de programação concorrente, como, Erlang, Limbo e Occam, porém, foi escolhida uma linguagem de programação sequencial devido a familiaridade com a sintaxe da mesma.

Este artigo é organizado da seguinte forma. Na Seção 2, será descrito de forma detalhada o Problema do Máximo Conjunto Independente. Na Seção 3, será apresentado trabalho relacionados, seja pela resolução do Problema do Máximo Conjunto Independente, a utilização da Otimização de Colônia de Formigas ou por trabalhos utilizando algoritmo paralelos. Na seção 5 será apresentada como e foi realizado os testes além das especificações da máquina em que foi realizada os testes. Após, será apresentado na Seção 4 de resolução do MISP e a partir da proposta, será analisado na Seção 6 os resultados obtidos. Por fim, na Seção 7 será descrito as conclusões obtidas e possíveis trabalhos futuros.

Problema

De acordo a Teoria dos Grafos, o Conjunto Independente é um conjunto de vértices em um grafo, em que estes vértices não podem possuir arestas entre si. Enquanto o problema do Máximo Conjunto Independente tem como objetivo encontrar o maior conjunto independente de um grafo. Também existe outros problemas envolvendo conjuntos independentes, por exemplo, o problema de decisão para verificar se há um conjunto independente de tamanho n e o maximal conjunto independente que não são subconjuntos de outros conjuntos independentes. A formulação matemática do MISP será mostrado adiante.

Dado um Grafo G = (V, E), em V representa o conjunto de vértices e E representa o conjunto de arestas do grafo. O Problema do Máximo Conjunto Independente tem como objetivo encontrar um subconjunto V* $\subseteq$ V, em que $\forall$i, j $\in$ V*, a aresta (i, j) $\notin$ E, além de que V* deve ser máximo.

A formulação da programação inteira para o MISP, pode ser definido da seguinte maneira:

$$\max \sum_{i=1}^{|V|} \text{ci*xi}$$

$$x_i + x_i \leq 1, \qquad \forall \text{ (i, j)} \in \text{E}$$

$$x_i \; in \; 0, 1, \qquad \text{i} = 1, 2, ..., |V|$$

O problema de decisão para verificar se há um conjunto de vértices independentes de tamanho n é classificado como NP-Completo. Enquanto o Problema Máximo Conjunto Independente é classificado como um problema de Otimização NP-Difícil.

O MISP é um problema muito similar ao problema do clique máximo, que resumidamente, busca encontrar o maior subconjunto de vértices adjacentes, em outras palavras, a cada par de vértices dentro do subconjunto, é necessário que haja uma aresta os interligando. É possível resolver o MISP através do clique máximo, dado o grafo de entrada no MISP, basta acharmos o seu complemento e utilizá-lo como entra para o clique máximo. Também é válido a operação inversa, utilizarmos o MISP para resolvermos o clique máximo. Com isso, dado um grafo G qualquer e seu complemento o grafo X, temos que:

MISP(G) = CliqueMaximo(X),

ou então,

CliqueMaximo(G) = MISP(X)

Na literatura é mais comum encontrarmos soluções e trabalhos envolvendo o clique e relacionados, como o Clique Máximo, porém devido a possibilidade de redução de um problema ao outro, foi de grande valia para o desenvolvimento deste trabalho alguns artigos com o clique como tema, tanto é que as instâncias encontradas para a realização dos testes foram criadas para testar solução para o Clique Máximo.

Trabalhos Relacionados

A meta-heurística Ant Colony Optimization foi apresentado por [Dorigo et al. 1996] a partir de observações sobre o comportamento de formigas reais a fim de descobrir como animais de pouca visão conseguiam se locomover e atingir seus destinos. Como o resultado das observações foi modelada e desenvolvida a ACO. Ao longo do tempo foi desenvolvida extensões para a ACO, como por exemplo, Recursive Ant Colony Optimization e Elitist Ant System.

Foram encontrados diversos trabalhos com o Máximo Conjunto Independente, sendo o mais antigo do ano 1977, desenvolvido por [Tarjan and Trojanowski 1976] apresentou um algoritmo de solução eficiente para o problema. Também foi encontrado trabalhos que apresentavam soluções utilizando heurísticas, bem como o trabalho de [Resense et al.] que utilizaram a meta-heurística GRASP e Algortimos Evolucionários por [Back and Khuri].

Direcionando as pesquisas para encontrar trabalhos que utilizaram a meta-heurística Ant Colony Optimization para a resolução do Máximo Conjunto Independente, foi encontrado dois trabalhos que propôs esse desafio [Choi et al. 2007] e [Leguizamon et al. 2011]. Para o presenta trabalho, utilizaremos propostas enunciadas neste trabalho, bem como a função probabilidade. e parâmetros.

Devido a semelhança com o MISP, houve pesquisas sobre o problema Clique Máximo e o resultado de maior relevância encontrado foi *DIMACS Implementation Challenges* criado pela *Rutgers University*, em que o segundo desafio proposto foi a

elaboração de soluções para o Clique Máximo, o website contendo maiores informações: http://dimacs.rutgers.edu/Challenges/.

Proposta

Para a resolução do problema, poderíamos utilizar algoritmos determinísticos, porém ao utilizar algoritmos heurísticos provavelmente encontraremos boas soluções com menor tempo em relação aos algoritmos determinísticos e com o uso de heurísticas adequadas a queda na qualidade das soluções não será tão sentida.

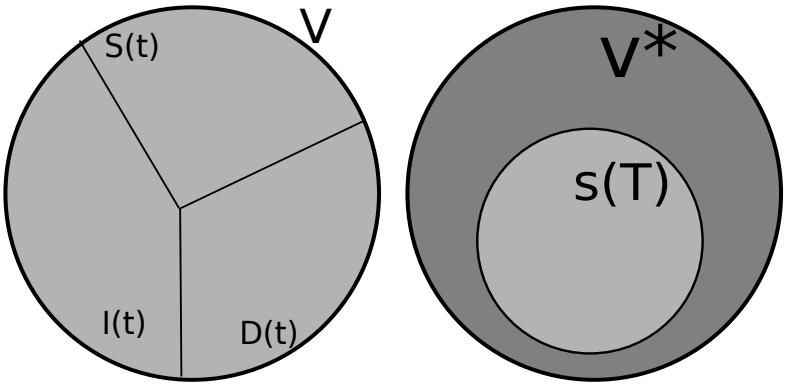
O código foi produzido na linguagem de programação C com a biblioteca pthread, foi escolhido essa opção ao invés de uma linguagem de programação desenvolvida no paradigma de programação concorrente devido ao maior conhecimento com a linguagem C, bem como suas características e sintaxe. A abordagem utilizada é de memória compartilhada, ou seja, dados como o vetor feromônio estão visíveis para todos os fluxos de execução, exigindo assim um gerenciamento dos dados, no caso, usaremos uma barreira para que todos os fluxos de um ciclo cheguem ao fim da execução do laço para que o próximo laço seja liberado. Também foi utilizando um mutex para gerenciar o vetor de melhor formiga de cada ciclo, pois pode acontecer a situação em que dois fluxos diferentes querem escrevem a sua melhor formiga como a melhor formiga do ciclo. Por exemplo, temos que a melhor formiga tem o valor 30, e um fluxo com melhor formiga com valor 31 e outro fluxo com melhor formiga com 32, ambos leem o valor 30 na melhor formiga, e ambos entram na condição do if, o segundo fluxo escreveu a sua formiga como a melhor, porém, como o primeiro fluxo já estava dentro do laço, não percebeu que o valor agora é melhor que o seu e assim sobrescreveu a formiga e no final, a melhor formiga ficou com o valor de 31, o que estaria incorreto, portanto, o mutex é necessário para gerenciar o recurso compartilhado. Agora, vamos detalhar a solução construída.

O único conhecimento que temos para resolver o problema é o grafo, e assim temos acesso a sua constituição, como por exemplo, o número de vértices e arestas, densidade e quantidade de cada vértices adjacentes de cada vértice.

Para resolvermos o problema, precisamos utilizar três conjuntos ao longo do algoritmo, os conjuntos são:

- S(t): Conjunto de Vértices presente na resposta em um determinado tempo t.
- I(t): Conjunto de Vértices que **não** podem estar mais na solução em um determinado tempo t.
- D(t): Conjunto de Vértices que **ainda** podem estar solução em um determinado tempo t.

Como vemos na Figura 1(a), os três conjuntos devem compor completamente o conjunto V, ou seja, S(t) + I(t) + D(t) = V em qualquer instante de tempo e um vértice não pode estar contido em mais de um conjunto. Na Figura 1(b), temos que a S(t) $\subseteq$ V*, ou seja, tendo V* como a solução final, a cada instante de tempo, o conjunto S(t) deve ser um subconjunto de V*, assim sendo, seja qual instante de tempo for, todos os elementos de S deverão estar contidos em V* no final.



(a) Composição dos conjuntos utilizados na solução.

(b) Composição do conjunto solução em um determinado tempo t.

Figura 1. Composições dos conjuntos.(a) e (b)

Dado o grafo descrito na Figura 2 vamos explicar a heurística utilizada. No grafo temos um total de oito vértices. Como suposição, vamos tomar que em determinado tempo t, no conjunto solução, temos S(t) = {A}. Como o vértice A está no conjunto solução, todos os vértices adjacentes a ele não pode mais estar na solução, logo precisamos descartá-los, assim sendo, I(t) = {B, C}. Para acharmos o conjunto dos vértices em que ainda podem estar na solução, temos D(t) = V - S(t) - I(t), logo D(t) = {D, E, F, G}.

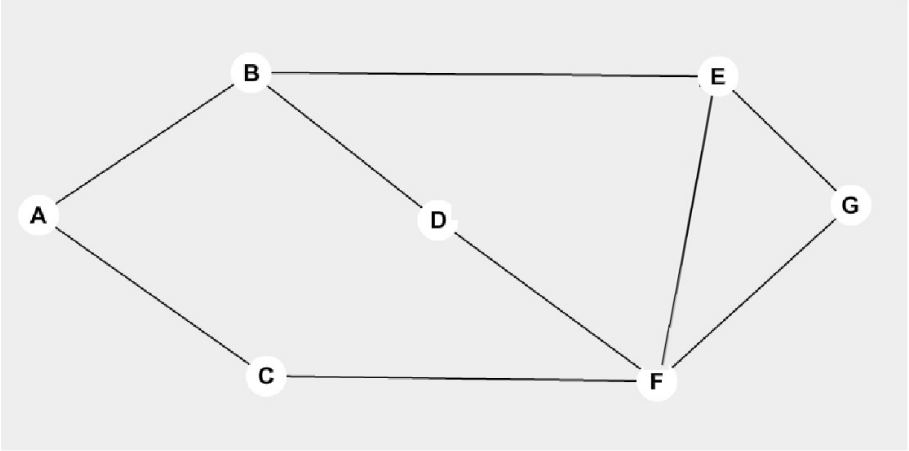


Figura 2. Grafo de Entrada para o MISP

Agora iremos encontrar o próximo vértice a ser adicionado ao conjunto solução. Para escolhermos o vértices, iremos calcular o número de vértices na qual o vértice em questão não são adjacentes, relembrando que estamos apenas trabalhando com os vértices contido no conjunto de vértices que ainda podem estar na solução. Esse número representa o valor da função heurística.

- τ_D(S(t)) = |{E, G} |= 2
- τ_E(S(t)) = |{D} |= 1
- τ_F(S(t)) = |{ } |= 0
- τ_G(S(t)) = |{D} |= 1

Agora que sabemos o valor da função heurística de cada vértice, iremos escolher o vértice com o maior valor, que no exemplo, foi o vértice D. Com isso, iremos adicionar o vértice D na solução, e os vértices adjacentes a D no conjunto de vértices que não podem estar na solução e repetir o processo com os vértices restantes disponíveis para a solução. O processo irá terminar em um tempo t' em que o conjunto D(t') = { }, lembrando que em qualquer tempo t, temos que ter a seguinte igualdade: V = S(t) + I(t) + D(t).

Algorithm 1 Pseudocódigo da Otimização de Colônia de Formigas

```
1: while (c < ciclos) do
2:     for (f = 1; f < formigas; f++) do
3:         listaFormiga[f] ← construirSolucao()
4:         verificaSolucao(listaFormiga[f])
5:     end for
6:     melhorColonia[c] ← selecionaFormiga(listaFormiga)
7:     atualizaFeromonio(melhorColonia[c])
8: end while
9: melhorGeral ← selecionaFormiga(melhorColonia)
10: returnmelhorGeral
```

No algoritmo 1 vemos a estrutura padrão da Otimização de Colônia de Formigas, esse pseudocódigo pode ser utilizado para quaisquer problemas, a diferença está contida em como modelamos as estruturas utilizadas, como por exemplo, as formigas, a respostas e as funções utilizadas No algoritmo 2 está representada a função de probabilidade que tem como utilidade determinar o próximo vértice a ser adicionado na lista solução. Por fim, no algortimo 3 é mostrado a função para atualização da taxa de feromônio dos vértices do problema, que envolve basicamente duas variáveis, a taxa de evaporação ρ e a taxa_feromonio que está diretamente vinculada a taxa de evaporação e tem como objetivo aumentar o feromônio nos vértices que pertencem ao conjunto solução da melhor formiga da colônia.

Algorithm 2 Função probabilidade

```
feromonio ← vetorFeromonio[V]
heuristica ← τ_V(S(t))
probabilidade ← feromonio^α + heuristica^β
returnprobabilidade
```

Algorithm 3 Função Atualiza Feromônio

```
taxa_feromonio ← 1 + (2 * ρ)
for ( doi = 1; i < Nr_vertices; i++)
    vetorFeromonio[i] ← vetorFeromonio[i] * (1 − ρ)
end for
for ( doi = 1; i < Vertices_solucao; i++)
    vetorFeromonio[i] ← vetorFeromonio[i] * taxa_feromonio
end for
```

Tanto a versão paralela quanto a versão sequencial da solução produzida, bem como, as instâncias encontradas e artigos relacionados ao tema está disponível em: https://github.com/MarcoADP/ACO-Sequencial.

Agora que foi mostrada a ideia por trás da proposta de solução para o MISP, iremos apresentar na próxima seção os métodos e os meios utilizados para a obtenção dos resultados construídos a partir da proposta desenvolvida neste artigo.

Metodologia

Para os testes e a elaboração do dados utilizamos da seguinte metodologia, cada instância foi executada utilizando de uma a cinco threads, e o resultado adquirido para cada thread foi obtido através da média entre dez execuções, observando-se que foi executado doze veze, o pior e o melhor resultado encontrados foram descartados. Também foi realizado o método de doze execuções para o código sequencial.

A máquina utilizada para a realização dos teste foi um Notebook Dell Vostro 5470 com os seguintes componentes:

- Processador: i5 4210 Dual Core
  Thread: 4
  L2 Cache: 0.5 MB
  L3 Cache: 3 MB
- Memória RAM: 4 GB
- SO: Linux Ubuntu 16.04 64 bits

De todas as instâncias de grafos obtidas durante a fase de pesquisa, foram testadas no total de cinco das sessenta e novas instâncias. Esta coletânea de grafos foi obtida do website do *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle* da Universidade Livre de Bruxelas, em que Marco Dorigo, é um dos diretores. O acesso aos grafos está disponível em: $http : //iridia.ulb.ac.be/\ fmascia/maximum_clique/DIMACS - benchmark.$

Resultados

As instâncias utilizadas nos testes estão descritas na tabela 1 com o número de vértices e arestas. Assim como observaremos nos resultados obtidos, grafos com mais arestas ou vértices não garantem necessariamente maior tempo de execução ou então um conjunto máximo independente maior.

Tabela 1. As instâncias utilizas neste trabalho

Instância	Vértices	Arestas
p_hat1500-2	1500	568960
keller6	3361	4619898
p_hat1500-3	1550	847244
keller5	776	225990
hamming10-4	1024	434 176

A seguir, na tabela 2 temos os resultados obtidos a partir dos cincos grafos enunciados anteriormente, bem como as valores utilizados para as variáveis nos testes.

- Ciclos: 100
- Formigas: 120
- Alpha: 2
- Beta: 1
- Rho: 0.1

Tabela 2. Resultados obtidos

Instância	Resultado	Sequencial	1 Thread	2 Threads	4 Threads	8 Threads
p_hat1500-2	62	11m20.303s	11m21.890	6m21.890s	4m47.258s	4m50.436s
keller6	63	10m14.038s	10m07.492s	05m47.471s	04m43.365s	04m42.479s
p_hat1500-3	12	1m25.282s	1m23.475	0m48.768s	0m34.015s	0m34.567s
keller5	31	0m38.686s	0m38.110s	0m21.850s	0m17.258s	0m17.928s
hamming10-4	20	0m38.397s	0m38.007	0m21.545	0m17.093	0m18.398

Com os resultados obtidos, podemos produzir uma análise em relação ao código paralelo e o código sequencial, e assim como produzirmos o gráfico do Speedup que está apresentado na figura 6.

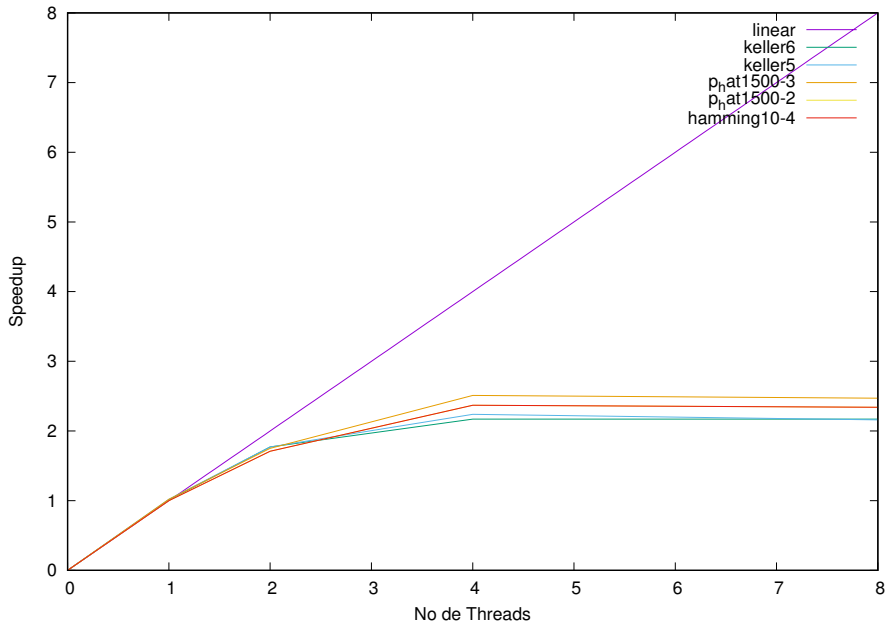


Figura 3. Gráfico Speedup

A primeira conclusão que podemos obter da figura 6, é que o tempo com 8 threads é muito similar ao tempo com 4 threads, isso se deve ao fato que a máquina utilizada nos experimento possui a capacidade de executar quatro threads paralelamente, portanto, ao executarmos com um número maior de threads não haverá ganho de perfomance. O tempo sequencial e o tempo paralelo foram idênticos em todos os testes, a relação entre os tempo variou entre 1.00 para os grafos hamming10-4 e p_hat1500-2 e 1.02 para os grafos p_hat1500-3 e keller5, resultados próximos a reta linear. Para 2 threads a eficiência sofreu um decréscimo, para o melhor caso, a eficiência foi de 0.89 ou 89% e o pior caso houve uma eficiência de 87%, na figura 6 iremos ter maiores sobre a eficiência. Por fim, para 4

threads houve um maior decréscimo no speedup, isso deve-se pelo fato, que o processador da máquina ser um dual-core, ou seja, possuir dois núcleos físicos, é suportado quatro threads devido a emulação de dois núcleos virtuais, que apesar de ocorrer um ganhou de eficiência em relação a um processador sem tal emulação, o ganho não é equivalente a um processador quad-core (quatro núcleos físicos), portanto é natural que o speedup não atinja um valor tão próximo ao speedup ideal.

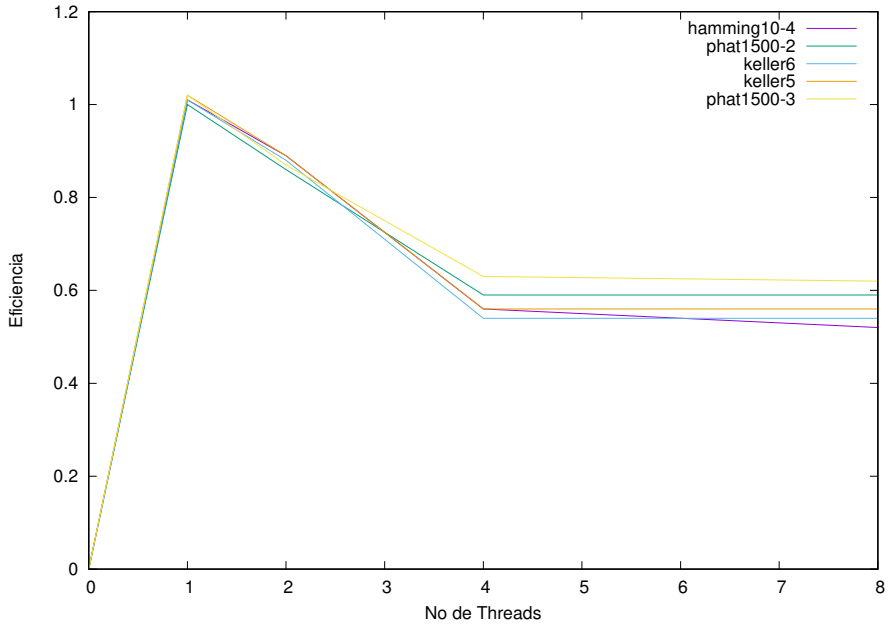


Figura 4. Gráfio Eficiência

A eficiência do código paralelo conforme mostra a figura 6 obtive um bom resultado para 1 e 2 threads, com resultado em torno de 100% e 89% respectivamente. Para 4 threads não foi conseguido obter uma eficiência tão boa quanto para 1 e 2 threads devido a máquina utilizada nos experimentos possuírem apenas 2 núcleos físicos e que os núcleos virtuais não possuem o desempenho, como já dito anteriormente. Para 8 threads a eficiência se mostrou similar a 4 threads devido, ao também já mencionado, o suporte máximo a 4 threads paralelamente.

Os resultados obtidos nas figuras 6 e 6 mostram resultados similares aos cinco grafos testados, independente da quantidade de vértices, arestas e de tempo gasto para a execução dos mesmos, a realização de mais testes com outros grafos disponíveis na coleção de grafos tendem a ser sem importância visto que as chances de resultarem em dados parecidos aos grafos trabalhos é grande, portanto, tais testes não foram executados.

Conclusões e Trabalhos Futuros

Os resultados desse trabalho mostraram que códigos paralelos feitos de foram correta é uma ferramente imprescindível para uma melhor utilização de máquinas com processadores multicores, pois assim pode-se utilizar de todas as threads disponíveis, ganhando em perfomance. Porém a codificação em paralelo não é tão comum e trivial quanto a codificação em sequencial, o que acarreta em um uso menos maciço mesmo que a técnica se mostre o quão eficiente como realmente é. Além disso, também ocorre a produção

de códigos paralelos equivocados devido a falta de conhecimento e a dificuldade gerada durante a confecção do mesmo.

A meta-heurística Otimização de Colônia de Formigas conforme já vem ocorrendo, se mostrou bastante eficiente na resolução do problema proposto no trabalho. O ACO desde sua introdução vem ganhando notoriedade por sua eficiência na resolução de problemas dos mais variados tipos. Outra importante característica é a sua portabilidade, resultando em versões modificadas visando um melhor resultado para algum problema específico.

Futuramente, uma boa hipótese de trabalho é a realização de testes em computadores com um número maior de núcleos físicos para observar a eficiência para um número maio threads. Outra hipótese é que ao ganhar uma maior bagagem ná area de algortimos paralelos, é a revisão do código paralelo confeccionado e detectar correções e melhorias possíveis.

Outras hipóteses são a criação de um código paralelo com memória distribuída e a realização de testes em *clusters*. Por fim, também é promissor promover um trabalho envolvendo programação paralela entre CPU e GPU, visto que a programação paralela em GPUs vem ganhando bastante espaço devido ao grande poder de processamento das placas gráficas atuais e o grande número de fluxos paralelos que pode ser gerenciado paralelamente.

Referências

Back, T. and Khuri, S. An evolutionary heuristic for the maximum independent set problem. Disponivel em: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=350004.

Choi, H., Ahn, N., and Park, S. (2007). An ant colony optimization approach for the maximum independent set problem. *Journal of the Korean Institute of Industrial Engineers*, 33(4).

Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 26(1):29–41.

Leguizamon, G., Schutz, M., and Michalewicz, Z. (2011). An ant system for the maximum independent set problem. Disponivel em: http://ls11-www.cs.tu-dortmund.de/downloads/papers/LeSz02.pdf.

Resense, M. G., Smith, S. H., and Feo, T. A. A greedy randomized adaptive search procedure for maximum independent set. Disponivel em: http://pubsonline.informs.org/doi/abs/10.1287/opre.42.5.860.

Tarjan, R. E. and Trojanowski, A. E. (1976). Finding a maximum independent set. *STAN-CS-76-550*.