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

Marco Aurélio Deoldoto Paulino[1]

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

marco23_aurelio@hotmail.com

Abstract. *This meta-paper describes the style to be used in articles and short papers for SBC conferences. For papers in English, you should add just an abstract while for the papers in Portuguese, we also ask for an abstract in Portuguese ("resumo"). In both cases, abstracts should not have more than 10 lines and must be in the first page of the paper.*

Resumo. *Este meta-artigo descreve o estilo a ser usado na confecção de artigos e resumos de artigos para publicação nos anais das conferências organizadas pela SBC. É solicitada a escrita de resumo e abstract apenas para os artigos escritos em português. Artigos em inglês deverão apresentar apenas abstract. Nos dois casos, o autor deve tomar cuidado para que o resumo (e o abstract) não ultrapassem 10 linhas cada, sendo que ambos devem estar na primeira página do artigo.*

Introdução

O Problema do Máximo Conjunto Independente, do inglês *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 o paradigma de programação paralela com padrão de comunicação *Message Passing Interface* (MPI). A versão produzida será comparada com a versão sequencial e com a versão paralela com a biblioteca pthread produzida por [Paulino 2016].

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.

O algortimo em paralelo utilizando o padrão de comunicação MPI tem como objetivo a divisão de processamento entre diversos processos, estes processos podem estar em uma única máquina ou em máquinas diferentes interconectadas. A função do MPI é justamente trocar mensagens entre os processos. O MPI tem suportabilidade com a comunicação assíncrona e programação modular, através de mecanismos de comunicadores que permitem ao usuário MPI definir módulos que encapsulem estruturas de comunicação interna. Com a divisão de processamento entre vários processos somado com processadores multicores e a possibilidade da utilização de *cluters* permite um ganho de potência para a execução de processos mais complexos. Por outro lado, a elaboração de código em paralelo é mais complexo em relação aos códigos sequencias e com a biblioteca OpenMPI é necessário maior cuidado para gerenciar a comunicação entre os processos e um balanceamento correto de carga entre os processos.

Será utilizada a abordagem de memória distribuída neste trabalho, ou seja, cada processo terá a sua própria porção de memória alocada, ao contrário da abordagem de memória compartilhada utilizada em [Paulino 2016] que todas as threads possuíam a mesma porção de memória e com isso era necessário o gerenciamento de sincronismo. A maior dificuldade neste trabalho será na uniformidade do vetor de feromônio para cada processo, visto que a proposta de trabalho é ter um vetor feromônio global.

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 4 será apresentada a resolução proposta para o MISP, seguindo na seção 5 será apresentada como e foi realizado os testes além das especificações do ambiente em que foi realizada os testes. 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. A partir do Conjunto Independente foi elaborado alguns problemas a serem resolvidos, como o problema de decisão para verificar se há um conjunto independente de tamanho n e o maximal conjunto independente, que consiste em encontrar um conjunto independente que não seja subconjunto de outros conjuntos independentes e o Máximo Conjunto Independente.

Neste trabalho iremos abordar o problema do Máximo Conjunto Independente, do inglês *Maximum Set Independent Problem* (MISP). O MISP tem como objetivo encontrar o maior conjunto independente de um grafo. 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 MISP encontra-se classificado como um problema de Otimização NP-Difícil, que é uma classe de problemas tão difíceis quanto os problemas mais difíceis em NP, enquanto o problema de decisão é dito como um problema NP-Completo, que são os problemas mas difíceis em NP. A classe NP-Completo está contida na classe 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 problema do clique e relacionados, como o Clique Máximo, devido a essa maior disponibilidade e a possibilidade de redução do problema do clique máximo ao MISP, e vice versa, foi de grande valia para o desenvolvimento deste trabalho artigos com o clique como tema, inclusive as instâncias encontradas para a realização dos testes foram criadas originalmente para testar soluções 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].

Para o estudo do padrão de comunicação MPI, foi utilizado o livro de [Snir 1998], que contém conteúdo bastante abrangente sobre o funcionamento do MPI.

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 presente trabalho, utilizaremos propostas enunciadas neste trabalho, bem como a função probabilidade. e parâmetros. Enquanto o trabalho de [Leguizamon et al. 2011] utilizou técnicas de paralelas com memória compartilhada, não foi encontrada na literatura, trabalhos envolvendo MISP com abordagem de memória distribuída, portanto, este trabalho se encontra como o primeiro trabalho a propor tal estratégia.

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 MISP é possível utilizar algoritmos determinísticos, apesar da garantia de encontrar a solução ótima, o tempo de cálculo para a descoberta do resultado seria grande. Para contrapor o gasto de tempo com soluções boas, a solução é utilizar algoritmos heurísticos que utilizam de uma determinada informação do problema para realizar escolhas, se tal informação for boa, será encontrado soluções boas muito próximas a solução ótima.

A informação utilizada para calcular a função heurística foi o número de vértices adjacentes para cada vértice. Para determinamos o valor da heurística precisamos utilizar três conjuntos:

- 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.

Estes três conjuntos compõem totalmente o conjunto de todos os vértices do problema, ou seja, a união entre os três conjuntos devem resultar no conjunto de todos os vértices, além de que nenhum vértice pode estar em mais de um conjunto em um determinado tempo t. Ao final do processamentos devemos ter o conjunto D vazio e o conjunto S será a solução encontrada para a instância dada. Apresentado as informações necessárias para calcular a função heurística, agora será mostrada um exemplo prático do cálculo.

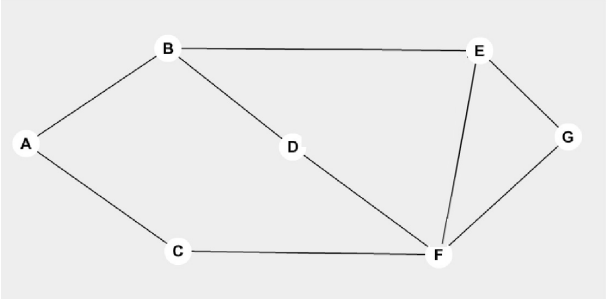

Figura 1. Grafo de Entrada para o MISP

Dado o grafo mostrada na figuna 1. No determinado tempo t, temos no conjunto solução S(t) = $\{$A$\}$. Como o vértice A está no conjunto solução, todos os vértices adjacentes a ele não pode estar mais na solução, logo, I(t) = $\{$B, C$\}$ e o cálculo da função heurística só ocorre para os vértices que ainda podem estar na solução, que são D(t) = $\{$D, E, F, G$\}$. O cálculo ocorre da seguinte maneira, para cada vértice presente em D(t), calculamos o número de vértices que o mesmo **não** é adjacente, com isso utilizamos a informação que foi dita necessária para calcularmos a 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 calculamos o valor da função heurística de cada vértice, a escolha do vértice a ser adicionado ao conjunto solução é o vértice que possuir a função heurística de maior valor, neste exemplo foi o vértice D. Após adicionar o vértice escolhido na solução, devemos atualizar os demais conjuntos, para o conjunto D(t+1) devem ser removidas os vértices adjacentes ao vértice D e os mesmos devem ser adicionados ao conjunto I(t+1), lembrando que em qualquer período de tempo, todos os vértices devem pertencer ao um único conjunto. O procedimento deve prosseguir até o momento de tempo t' em que não resta mais elementos no conjunto D(t'), tendo no conjunto S(t') a solução do problema.

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

$feromonio \leftarrow vetorFeromonio[V]$
$heuristica \leftarrow \tau_V(S(t))$
$probabilidade \leftarrow feromonio^{\alpha} + heuristica^{\beta}$
$returnprobabilidade$

Algorithm 3 Função Atualiza Feromônio

$taxa\_feromonio \leftarrow 1 + (2 * \rho)$
for (**do**i = 1; i $<$ Nr_vertices; i++)
    $vetorFeromonio[i] \leftarrow vetorFeromonio[i] * (1 - \rho)$
end for
for (**do**i = 1; i $<$ Vertices_solucao; i++)
    $vetorFeromonio[i] \leftarrow vetorFeromonio[i] * taxa\_feromonio$
end for

Para realizar a versão paralela com abordagem de memória compartilhada, conforme podemos ver no algortimo 4 é necessário realizar uma adição de operações para realizar a comunicação entre os processos, em que após todos os processos computarem sua resposta no ciclo c, todos devem enviar sua resposta ao processo de rank 0, em que o mesmo calcula qual resposta de todos é a melhor, após encontrar o melhor resultado, o processo 0 enviará a todos os outros processos a melhor resposta e todos os processos irão atualizar o seu vetor Feromônio que deve ser igual para todos os processos. O cálculo da função probabilidade probabilidade e para atualizar feromônio permanecem o mesmo conceito do algoritmo sequencial.

Algorithm 4 Pseudocódigo da Otimização de Colônia de Formigas Paralelizado

```
1: formiga_processo ← formigas/numero_processos
2: while (c < ciclos) do
3:     for (f = 1; f < formiga_processo; f++) do
4:         listaFormiga[f] ← construirSolucao()
5:         verificaSolucao(listaFormiga[f])
6:     end for
7:     melhorColonia[c] ← selecionaFormiga(listaFormiga)
8:     for (p = 1; p < numero_processo; p++) do
9:         Send(melhorColonia[c], 0)
10:    end for
11:    if wrank == 0 then
12:        for (p = 1; p < numero_processo; p++) do
13:            Receive(Resposta[p], p)
14:            if Resposta[p] > melhorColonia[c] then
15:                melhorColonia[c] ← Resposta[p]
16:            end if
17:        end for
18:    end if
19:    Broadcast(melhorColonia[c], 0)
20:    atualizaFeromonio(melhorColonia[c])
21: end while
22: melhorGeral ← selecionaFormiga(melhorColonia)
23: returnmelhorGeral
```

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 um, dois, quatro e oito processos, e o resultado adquirido para cada processo foi obtido através da média entre dez execuções, observando-se que foi executado doze vezes, o pior e o melhor resultado encontrados foram descartados.

Foi utilizado o laboratório de Informática do Departamento de Informática da Universidade Estadual de Maringá com máquinas com as seguintes configurações:

- Processador: i7 3370K Quad Core
  Thread: 8
  L2 Cache: 1 MB
  L3 Cache: 8 MB
- Memória RAM: 4 GB

- SO: Linux Ubuntu 12.04 32 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	Resultado
p_hat1500-2	1500	568960	62
keller6	3361	4619898	63
p_hat1000-2	1000	244799	53

A seguir, na tabela REFERENCIAS 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

GRAFICO DE BARRAS

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. ARRUMAR O GRAFICO

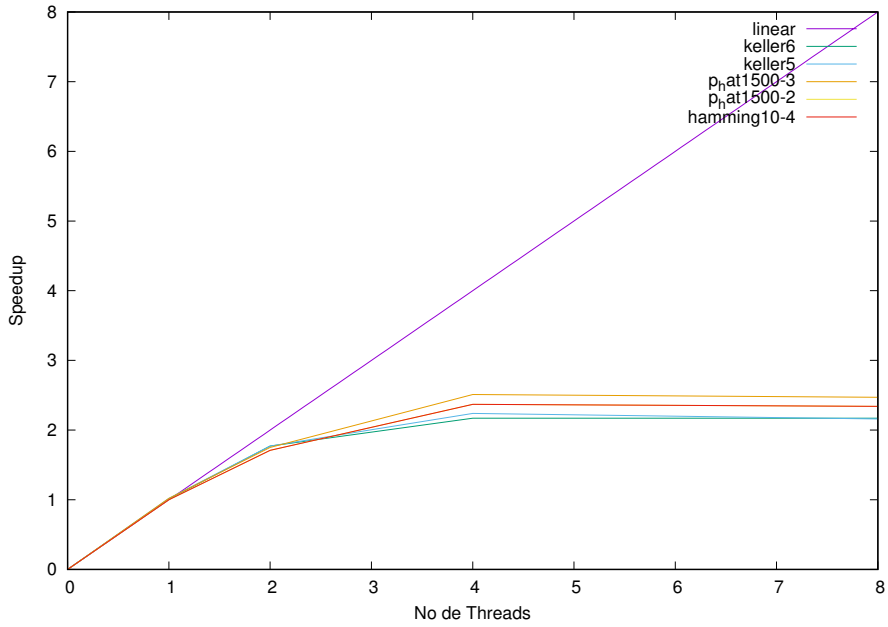

Figura 2. Gráfico Speedup

A primeira conclusão que podemos obter da figura 6, é que o speedup encontrado foi o ideal, como para essa versão do código não foi necessário utilizar mecanismos de sincronização como barreira e locks, os processos não necessitavam esperar a não ser quando estavam esperando mensagem de outro processo como o processo de rank 0 necessitava receber a melhor formiga de todos os processos para a escolha da maior, bem como, todos os processos precisavam esperar o processo de rank 0 enviar a melhor formiga para assim atualizarem o seu feromônio, essa possibilidade de não precisar utilizar tais sincronismos garantiam que uma melhora no tempo. Para todas as instâncias utilizadas o speedup foi semelhante, o que podemos inferir que o código se comporta da mesma maneira para quaisquer instância, garantindo um bom resultado de speedup.

A partir do speedup podemos também inferir que sua eficiência foi positiva, atingindo resultado ideal para todos os números de processos e para quaisquer instâncias, conforme podemos ver na figura 6.

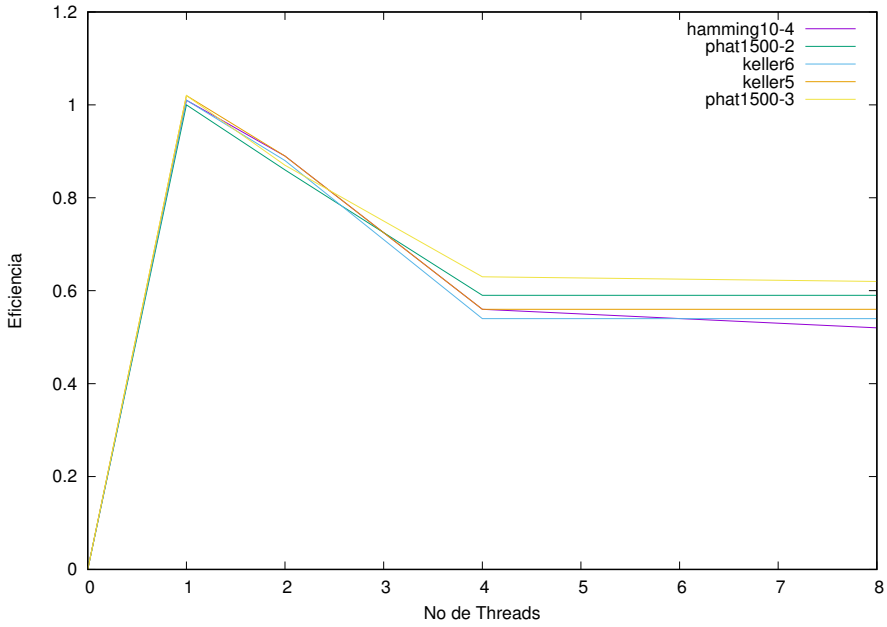

Figura 3. Gráfio Eficiência

Conclusões e Trabalhos Futuros

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.

Paulino, M. A. D. (2016). Algoritmo heuristico otimizacao de colonia de formigas e programacao paralela para a resolucao do problema do maximo conjunto independente.

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.

Snir, M. (1998). *MPI–the Complete Reference: The MPI core*, volume 1. MIT press.

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