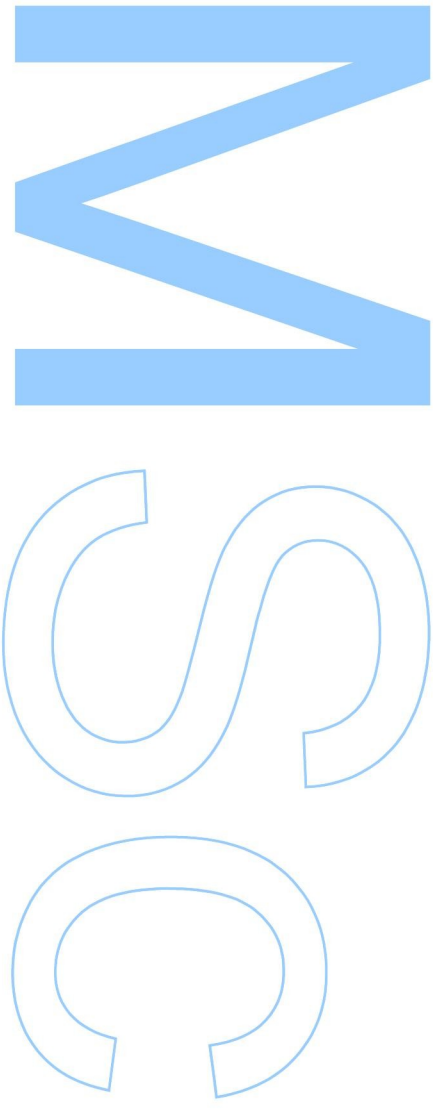
Particionamento Automático de Redes de Restrições para Execução Paralela

Xu Yi
Mestrado em Ciência de Computadores
Departamento de Ciência de Computadores
2013

Orientador
Inês de Castro Dutra
Professora Auxiliar
Faculdade de Ciências da Universidade do Porto



U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, / /

Acknowledgments

First I would like to thank my supervisor that made this thesis possible, Prof. Dr. Inês Dutra

My friends that helped through this part of my life.

Finally my family for their never ending support.

Resumo

Este trabalho concentra-se no desenvolvimento de um método de particionamento de restrições baseado na bisseção recursiva espectral de Hendrickson e Leland. As restrições são representadas num grafo que é particionado através de cálculos de vetores próprios e valores próprios de uma matriz laplaciana associada ao grafo. Utilizamos dez grafos com características variadas para o particionamento e comparamos os resultados com um segundo método baseado em "min-cut" (corte mínimo do grafo) chamado Max Aggregation. As métricas de avaliação utilizadas foram número de arestas, número de vértices, densidade e grau médio dos vértices. Os resultados mostram que o método de bisseção espectral recursiva, em geral, produz um número maior de grupos do que o método Max Aggregation, o que pode favorecer um melhor aproveitamento dos processadores e pode permitir a troca de mensagens simultânea entre os vários grupos.

Abstract

This work focuses on the development of a partitioning method for constraints networks based on the recursive spectral bisection of Hendrickson and Leland. Constraints are represented in a graph that is partitioned by calculation of eigenvectors and eigenvalues of a Laplacian matrix associated with the graph. We used ten graphs with various characteristics for partitioning and compared the results with a second method based on min-cut called Max Aggregation. The evaluation metrics used were number of edges, number of nodes, density and average degree of nodes. The results show that the spectral recursive bisection method generally produces a number of groups greater than the Max aggregation method, which can promote better use of the processors and can allow the exchange of simultaneous messages between the various groups.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Computação paralela é uma forma de computação em que vários cálculos são realizados simultaneamente, operando sob o princípio de que grandes problemas geralmente podem ser divididos em problemas menores, que então são resolvidos concorrentemente (em paralelo).

À medida que a complexidade do problema cresce, voltamo-nos para os computadores paralelos para nos ajudar a encontrar uma solução aceitável.

Neste trabalho nos concentramos na solução de problemas de satisfação de restrições e em métodos de decomposição do conjunto de restrições para execução paralela.

1.1 Contextualização

Um problema de satisfação de restrições (CSP, do inglês Constraint Satisfaction Problem) é definido como uma tripla $\langle X, D, C\rangle$, onde X é um conjunto de variáveis, D é o domínio dos valores que cada variável pode assumir e C é um conjunto de restrições. Toda restrição $c \in C$ é um par $\langle t, R\rangle$, onde t é um tuplo de variáveis e R é uma relação matemática entre as variáveis. Uma avaliação das variáveis é uma função de variáveis para o domínio de valores $v : X \rightarrow$ D. Uma avaliação v satisfaz as restrições

$\langle (x_1, \ldots, x_n), R \rangle$ se $(v(x_1), \ldots, v(x_n)) \in R$. Uma solução é uma avaliação que satisfaz todas as restrições.

É possível representar um CSP por um grafo de restrições, onde cada vértice representa uma variável e cada arco pode representar uma restrição entre variáveis. Qualquer CSP com restrições contendo n variáveis pode ser convertido em um CSP envolvendo apenas pares de variáveis [10] (restrições binárias).

Como exemplo de um CSP com restrições binárias, podemos citar o problema de coloração de mapas. Ele pode ser classificado como um CSP com domínio finito. Neste problema precisamos colorir cada região de um mapa com uma cor (de um conjunto de cores), tal que duas regiões adjacentes não tenham a mesma cor.

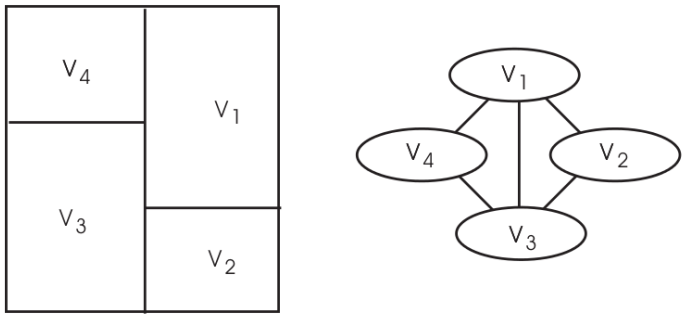


Figura 1.1: Exemplo de um problema de coloração de mapas e seu grafo de restrições equivalente [10]

A Figura 1.1 mostra um exemplo do problema de coloração de mapas e seu CSP equivalente, representado por um grafo de restrições tradicional. O mapa tem quatro regiões que devem ser coloridas com as cores verde, azul ou vermelho (domínio de possíveis valores para uma variável). O CSP equivalente tem uma variável para cada uma das quatro regiões do mapa (V_1, V_2, V_3, V_4). O domínio de cada uma das variáveis é dado pelo conjunto de cores. Para cada par de regiões que são adjacentes no mapa, há uma restrição binária ($V_i \neq V_j$) entre as variáveis correspondentes que não permite atribuições idênticas para as duas variáveis. No grafo da Figura 1.1, cada vértice representa uma variável do problema (V_1, V_2, V_3, V_4) e as arestas representam as

restrições existentes entre as variáveis ($V_1 \neq V_2$, $V_1 \neq V_3$, $V_1 \neq V_4$, $V_2 \neq V_3$, $V_3 \neq V_4$).

Neste exemplo, o domínio das variáveis é finito e categórico, mas em CSPs, os domínios podem ser reais, racionais etc. Dependendo do tipo de domínio, as soluções podem ser encontradas utilizando algoritmos matemáticos ou algoritmos de procura.

1.2 Motivação

Para a solução de um CSP, no domínio finito, diversos algoritmos, baseados em consistência de arcos, foram propostos como AC-1 [10], AC-2 [11], AC-3 [10], AC-4 [13], AC-5 [7], AC-6 [2] e AC-7 [3]. O algoritmo AC-5 (uma generalização dos algoritmos de AC-1 a AC-4) possui a menor complexidade $O(ed)$, onde e é o número de restrições e d é o tamanho do domínio. Em geral, a complexidade destes algoritmos é exponencial, o que pode levar a que o espaço de procura cresça de forma a que não se consiga encontrar uma solução em tempo viável. Desta forma é necessário obter alternativas para acelerar a sua execução ou mesmo conseguir executar instâncias maiores de problemas. Uma alternativa é particionar um dos componentes do problema: conjunto de variáveis, domínios ou restrições. Pode-se ainda combinar estes particionamentos. Este trabalho concentra-se no particionamento de restrições, isso porque trabalhos anteriores mostraram resultados promissores [16, 19].

A obtenção de subconjuntos de restrições independentes é um problema NP-completo. O particionamento do conjunto de variáveis ou do domínio também é um problema NP-completo. Heurísticas precisam ser utilizadas para a que o overhead do algoritmo de particionamento seja minimizado.

1.3 Objetivos

Nosso objetivo é portanto obter partições de um CSP representado na forma de um grafo, de forma a poder acelerar a procura por uma solução. Nosso problema

então resume-se a particionamento de grafos. Há várias abordagens na literatura para particionamento de grafos [14]. Neste trabalho utilizamos abordagens livres de coordenadas (Coordinate-Free). Estas abordagens incluem técnicas espectrais e baseadas em teoria dos grafos e geralmente produzem particionamentos de melhor qualidade pois se baseiam na estrutura de conectividade do grafo, capturando as características estruturais de dependências nas aplicações. Estas técnicas são bastante utilizadas em várias aplicações, visto que controlam melhor os custos de comunicação (no nosso caso, dependências entre restrições são modeladas nas arestas no grafo). Neste trabalho, nos concentramos num algoritmo baseado em corte mínimo [8] e no algoritmo de bisseção espectral recursiva (Recursive Spectral Bissection) [17, 21, 6].

1.4 Contribuições

Implementamos o algoritmo de Bisseção Espectral Recursivo baseado nos trabalhos de Simon [21] e Hendrickson e Leland [6], na linguagem Python. Aplicamos este algoritmo a 10 grafos com características diferentes e utilizamos como métricas de avaliação o número de vértices, o número de arestas, a densidade, o grau médio dos vértices e número de arestas entre blocos particionados para comparar os resultados da nossa implementação com um outro algoritmo baseado em corte de grafos chamado Max Aggregation. Resultados mostram que o método de bisseção espectral recursiva, em geral, produz um número maior de grupos do que o método Max Aggregation, o que pode favorecer um melhor aproveitamento dos processadores e pode permitir a troca de mensagens simultânea num CSP.

1.5 Organização da Dissertação

Este documento está organizado em seis capítulos:

Capítulo 1 – Este capítulo introduziu o tema da dissertação, assim como revelou a

motivação e objetivos inerentes a este trabalho.

Capítulo 2 – Neste capítulo apresentamos a fundamentação teórica sobre métodos de bisseção. Neste capítulo, também efetuamos um levantamento das redes de restrições e paralelização de redes de restrições.

Capítulo 3 – Neste capítulo apresentamos a descrição do método de bisseção espectral utilizado neste trabalho, explicando a estrutura de dados utilizada, bem como a sua implementação.

Capítulo 4 – Neste capítulo apresentamos os Materiais e Métodos utilizados para a realização das experiências. Apresentamos os grafos classificados em categorias relacionadas com a sua densidade e explicamos o método de avaliação do particionamento.

Capítulo 5 – Neste capítulo são apresentados e analisados os resultados obtidos após a execução das experiências, de acordo com as métricas de desempenho.

Capítulo 6 – Finalmente, este capítulo apresenta as considerações finais, onde é efetuado um balanço sobre todo o trabalho realizado, com especial destaque para os objetivos propostos. O capítulo termina com uma abordagem ao trabalho futuro.

Capítulo 2

Fundamentação Teórica

Neste capítulo, apresentamos os conceitos fundamentais para o entendimento do restante do trabalho. Introduzimos problemas de satisfação de restrições, sua representação em grafos, algoritmos para resolvê-los e suas complexidades, assim como, exemplos de problemas. Também neste capítulo, apresentamos os principais métodos de particionamento em grafos.

2.1 Problemas de Satisfação de Restrições

Um modelo que envolva variáveis, seus domínios e restrições entre variáveis é chamado de problema de satisfação de restrições ou rede de restrições [18] . Neste texto é utilizada a notação problema de satisfação de restrições (*Constraint Satisfaction Problem* - CSP).

Um CSP é um tipo especial de problema de busca que possui estados e domínios. Os estados são conjuntos de variáveis. O estado inicial é um conjunto de variáveis com valores possíveis iniciais. O estado final é um conjunto de variáveis com valores que respeitem as restrições do problema. O domínio é o conjunto possível de valores que uma variável pode assumir, que pode ser discreto ou contínuo e finito ou infinito.

O CSP define restrições sobre variáveis e um domínio que relaciona cada variável a um conjunto de valores.

Um *solver* é um método para resolver CSPs. Vários *solvers* de CSP possuem complexidade polinomial. O objetivo dos *solvers* é transformar um CSP, que tem espaço de busca exponencial, em outro equivalente com os domínios menores para as variáveis [15]. Para encontrar as soluções o *solver* passa por fases de eliminação de valores do domínio das variáveis, de acordo com as restrições.

Os *solvers* dependem do domínio das variáveis. No caso de restrições no domínio real (infinito), são utilizados métodos matemáticos como eliminação de Gauss e Fourier-Moutzkin ou métodos computacionais como o Simplex [22]. No caso de domínios finitos, uma classe importante dos algoritmos para resolver CSPs é a classe dos algoritmos de consistência de arcos[12]. Estes algoritmos têm complexidade exponencial dependente do número de variáveis, número de restrições e tamanho do domínio de cada variável.

Para solucionar CSPs que possuem domínios finitos são necessárias duas escolhas: a da variável e a do valor da variável. A escolha da variável pode ser *most-constrained*, *most-constraining* ou *least-constrained*. Na escolha *most-constrained* é escolhida a variável de menor domínio. Na *most-constraining*, a variável escolhida é a que restringe ao máximo os domínios das outras variáveis. Na *least-constrained* a variável com maior domínio é escolhida. Além disso, *least-constrained* utiliza uma heurística baseada na minimização do número de falhas para evitar *backtracking*.

A escolha do valor da variável, também, pode ser feita utilizando-se vários métodos: *least-constraining*, menor valor, valor médio, maior valor ou valor sequencial. Pelo princípio *least constraining*, o valor escolhido é aquele que afeta menos o conjunto de valores das outras variáveis. A escolha dos valores menor, médio ou maior consiste em escolher, respectivamente, o menor, o valor médio ou o maior valor do conjunto de valores do domínio. A escolha de um valor seqüencial consiste em selecionar o próximo valor do domínio que ainda não foi escolhido.

As heurísticas utilizadas para a seleção da próxima variável ou do próximo valor a ser atribuído a uma variável têm como objetivo principal reduzir o espaço de procura tentando encontrar soluções de uma forma mais rápida. Mas ainda para muitos problemas, ou estas heurísticas não são aplicáveis ou não temos informação suficiente sobre o problema para aplicá-las da forma mais eficaz. Uma alternativa para acelerar a execução passa, portanto por encontrar subgrupos (semi)independentes de restrições que possam ser processadas em paralelo.

Considerando apenas problemas de satisfação de restrições que possuem um conjunto de variáveis, um domínio finito para cada variável e um conjunto de restrições unárias ou binárias, é possível representar o CSP por um grafo de restrições, onde cada nó representa uma variável e cada arco representa uma restrição entre variáveis.

Para exemplificar um problema de satisfação de restrições sobre domínios finitos, suponha o problema de se colocar N rainhas em um tabuleiro de xadrez $N \times N$ de tal forma que as rainhas não se ataquem. As rainhas se atacam se estiverem na mesma linha, na mesma coluna, na mesma diagonal ascendente ou na mesma diagonal descendente. Cada rainha deve ser colocada em uma linha do tabuleiro. O problema consiste em selecionar uma coluna para cada rainha, de forma que elas não se ataquem. Este é um problema clássico em satisfação de restrições representativo de uma série de aplicações (por exemplo, controle de tráfego).

Uma representação em satisfação de restrições deste problema consiste em associar cada rainha a uma variável e fixar cada rainha numa linha do tabuleiro. Cada variável pode assumir valores de 1 a N, que são os valores das colunas que as rainhas podem ocupar e que correspondem ao domínio do problema. As restrições correspondem às condições necessárias e suficientes para que as rainhas não se ataquem. Para exemplificar o conjunto de restrições, suponha que X e Y sejam duas rainhas. As restrições para que a rainha Y seja colocada no tabuleiro de xadrez de forma que não ataque a rainha X, colocada anteriormente, podem ser escritas da seguinte forma:

$Y \neq X$, rainhas X e Y não se atacam na mesma coluna;

$Y \neq X + I$, rainhas X e Y não se atacam numa das diagonais e

$Y + I \neq X$, rainhas X e Y não se atacam na outra diagonal,

onde $I \in 1, ..., N$ corresponde à diferença entre as linhas que as rainhas coupam [16].

Para uma representação em grafo, podemos colocar as variáveis ou as restrições nos vértices. Como o nosso objetivo é fazer o particionamento do grafo para execução paralela e diminuir a comunicação (número de arestas que cruzam de um vértice a outro), o mais natural é ter uma representação em grafo, onde os vértices são as restrições e as arestas são as variáveis compartilhadas entre as restrições (considerando que as restrições são unidades de execução).

O grafo de restrições tradicional, em CSP, representa as variáveis nos nós e as restrições nas arestas. Este tipo de grafo é usado em geral com algoritmos de consistência de arcos. Nesta representação, as dependências entre os vértices (arestas) são restrições. A Figura 2.1 exemplifica este tipo de grafo para um CSP com 4 variáveis (V_1, V_2, V_3, V_4) e 3 restrições ($V_1 = V_2 + 1$, $V_1 = V_3 + 2$ e $V_1 = V_4 + 3$).

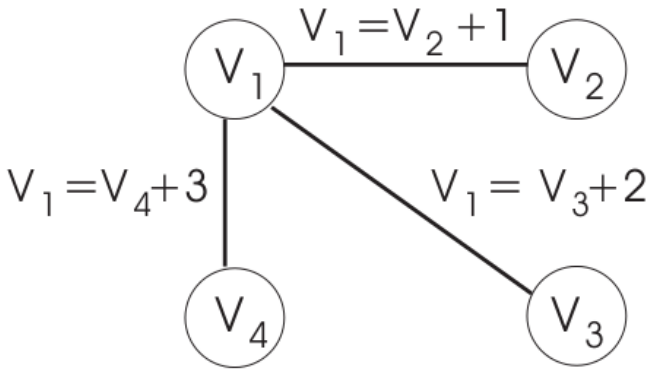


Figura 2.1: Grafo de restrições tradicional (retirado de [16])

Uma segunda forma de representar o problema considera as restrições como nós e as variáveis comuns entre as restrições nas arestas. Com esta representação, a dependência entre os nós passa ser as variáveis comuns entre eles. Considerando o mesmo

exemplo da Figura 2.1, a nova representação do grafo de restrições é apresentada na Figura 2.2. Note que, para este exemplo, as 3 restrições possuem a mesma variável em comum ($V1$). Com esta representação o particionamento das restrições pode utilizar como fator para agrupar restrições as variáveis comuns entre as restrições.

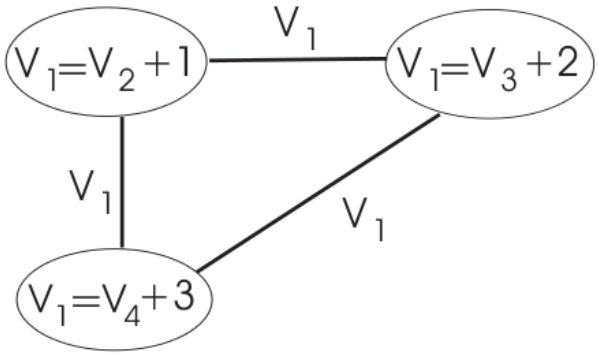


Figura 2.2: Grafo de restrições para particionamento (retirado de [16])

Este tipo de grafo pode ser particionado de forma que grupos de restrições possam ser executadas em paralelo. O nosso objetivo é decompor um grafo como este minimizando o número de arestas entre os diferentes grupos.

2.2 Grafos e Problemas Relacionados [20]

Um grafo G com pesos consiste de um conjunto de nós V e um conjunto de arestas $E \subset V \times V$ que representam as relações entre os nós, assim como duas funções de custo. Uma função atribui pesos aos vértices $c : V \to \mathbb{R}_{>0}$ e uma segunda função ω: $E \to \mathbb{R}$ atribui pesos às arestas. Em geral, adotamos a variável n para o número de nós e m para o número de arestas. Em um grafo não dirigido uma aresta $(u, v) \in E$ implica uma aresta $(v, u) \in E$ em que ambos os pesos das arestas são iguais. Usamos a notação de conjunto $\{u, v\} \in E$ no caso não dirigido. Estendemos c e ω para notação de conjuntos, isto é, $c(V') := \sum_{v \in V'} c(v)$ e $\omega(E') := \sum_{e \in E'} \omega(e)$. O conjunto $\Gamma(u) := \{v : \{u, v\} \in E\}$ denota os vizinhos do nó u. O grau de um nó é o número de seus vizinhos. Δ denota o grau máximo de um grafo. O grau ponderado de um nó é a

soma dos pesos das suas arestas incidentes. Um grafo é bipartido se seu conjunto de nós puder ser dividido em dois conjuntos disjuntos U e V de tal modo que $(u, v) \in E$ implica $u \in U$ e $v \in V$ ou vice-versa. Um subgrafo é um grafo cujo nó e conjunto de arestas são subconjuntos de um outro grafo. Chamamos de induzido a um subgrafo que tem todas as arestas possíveis.

2.2.1 Partições e Agrupamentos (Clusterings)

Dado um número $k \in \mathbb{N}_{>1}$ e um grafo não dirigido com pesos não negativos nas arestas, o problema de particionamento do grafo consiste em obter blocos de nós $V_1, \ldots, V_k$ (subgrafos) cuja união é o conjunto de nós V, Isto é,

1. $V_1 \cup \cdots \cup V_k = V$
2. $V_i \cap V_j = 0 \ \forall i \neq j$

Uma restrição de balanceamento exige que todos os blocos tenham tamanho aproximadamente igual. Mais precisamente,isto exige que, $\forall \in 1 \ldots k : |v_i| \leq L_{max} := (1 + \varepsilon)\lceil |V|/k \rceil$ para algum parâmetro de desbalanceamento $\varepsilon \in \mathbb{R}_{\geq 0}$, no caso em que a função de custo dos vértices é igual a um. No caso de $\varepsilon = 0$, também usamos o termo perfeitamente balanceado. Um bloco V_i é dito com pouca carga se $|V_i| < L_{max}$ e sobrecarregado se $|V_i| > L_{max}$. Um agrupamento é também um particionamento dos vértices do grafo, contudo k normalmente não é conhecido e a restrição de balanceamento não é levada em consideração. Note-se que um particionamento também é um agrupamento de um gráfico. Em ambos os casos, o objetivo consiste em minimizar ou maximizar uma função objetivo em particular. Duas funções bem conhecidas para o problema de particionamento serão apresentadas na próxima subseção. Um nó $v \in V_i$ que tem um vizinho $w \in V_j$, $i \neq j$ é um nó fronteira. Uma aresta entre dois blocos é também chamada aresta de corte. O conjunto $E_{ij} := \{\{u, v\} \in E : u \in V_i, v \in V_j\}$ é o conjunto de arestas de corte entre dois blocos V_i e V_j. Uma visão abstrata do grafo particionado é o chamado grafo quociente, onde os nós representam os blocos e

as arestas são induzidas por conectividade entre os blocos, ou seja, existe uma aresta no grafo quociente se existir uma aresta entre os blocos no grafo particionado original. Um exemplo é dado na Figura 2.3. Dados dois agrupamentos ξ_1 e ξ_2, o agrupamento sobreposição (*overlay clustering*) é o agrupamento em que cada bloco corresponde a um componente conexo do grafo $G_\varepsilon = (V, E \backslash \varepsilon)$ onde ε é a união das arestas do corte de ξ_1 e ξ_2, Ou seja, todas as arestas que ligam os blocos em ξ_1 ou ξ_2.

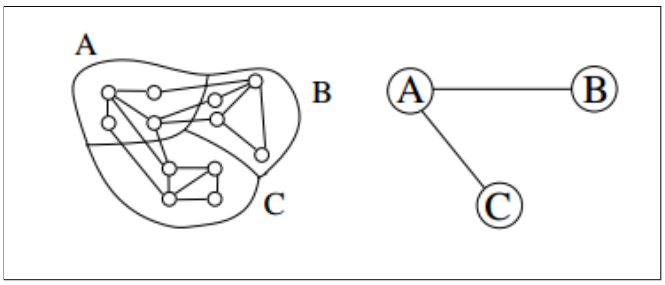


Figura 2.3: Um grafo dividido em três blocos de tamanho quatro à esquerda e o seu grafo quociente correspondente à direita. Existe uma aresta no grafo quociente se houver uma aresta entre os blocos correspondentes no grafo original. (Figura retirada de [20]).

2.2.2 Funções Objetivo

Na prática, muitas vezes procuramos encontrar uma partição que minimiza (ou maximiza) um objetivo. Provavelmente, a função objetivo mais importante é minimizar o corte total:

$$\sum_{i<j} \omega(E_{ij})$$

Sabe-se que existem funções objectivo mais realistas (e mais complicadas) que dependem da natureza do problema que está a ser resolvido, mas a função de minimização do tamanho do corte tem sido adotada como um tipo de padrão, uma vez que é geralmente correlacionada com outras formulações.

2.2.3 Algoritmos de Particionamento em Grafos

Na literatura, podemos encontrar uma quantidade de algoritmos de particionamento em grafos. Segundo vários autores, o algoritmo mais popular para o particionamento de grafos para execução paralela é o de bisseção espectral recursiva (Recursive Spectral Bissection) [17, 21, 6]. Algoritmos de particionamento em grafos podem ser divididos em dois grandes grupos: os de melhoramento local e os de melhoramento global. Os primeiros utilizam métodos de otimização local partindo de uma bisseção do grafo enquanto os métodos de otimização global partem do grafo inteiro.

Uma segunda classificação de algoritmos de particionamento está relacionada com a forma como os grafos são particionados. Desta forma, existem os métodos geométricos, que fazem o particionamento a partir da geometria do grafo (coordenadas), os métodos livres de coordenadas (coordinate-free), que consideram o grau de conectividade do grafo e os métodos dinâmicos (consideram que o grafo muda ao longo do tempo) [14].

Devido à quantidade muito grande de algoritmos de particionamento, nesta seção vamos nos concentrar apenas em dois métodos. Os dois são livres de coordenadas: Recursive Spectral Bissection [17, 6] e um algoritmo baseado no corte mínimo de um grafo com pesos [23], que se encaixa numa categoria de algoritmos que usam procura gulosa ("greedy") para decompor o grafo em grupos de vértices. Nos concentramos nos métodos livres de coordenadas porque estes fazem o particionamento levando em consideração a conectividade e a estrutura do grafo, gerando, desta forma, partições em que se consegue minimizar a quantidade de arestas entre os grupos de vértices, o que resulta num particionamento mais adequado para execução paralela, visto que a diminuição de arestas significa na prática a redução de comunicação. O problema de reduzir o número de arestas entre grupos é o mesmo que encontrar o corte mínimo de um grafo [5]. Fontes muito boas de informação sobre algoritmos de particionamento podem ser encontradas no *survey* de Fjällström [4], no livro de Padua [14] (chapter on Domain Decomposition) e no artigo de Arora, Rao and Vazirani [1].

Uma variação destes algoritmos considera que as arestas têm custo diferente de um,

ou seja, têm um peso associado. Neste caso, define-se o peso de um vértice como sendo o somatório dos pesos de suas arestas incidentes.

Os algoritmos de procura gulosa normalmente usam uma estratégia em largura combinada com heurísticas para decompor o grafo em grupos de vértices. As heurísticas são normalmente baseadas em pesos dos vértices ou pesos das arestas.

Neste trabalho, utilizamos um algoritmo chamado Heavy Decomposition Max Aggregation. Nesta solucão, o algoritmo gera os grupos com base no seguinte método [8, 23]:

1. Verificar o nó N que tem mais peso em todo o grafo (soma dos pesos das arestas é máximo dentre todos os vértices).

2. Verificar todos os vértices ligados a N e selecionar aquele (ou aqueles) que tem aresta de maior peso. Um novo grafo é gerado sem esta aresta e com um novo vértice que agrupa o vértice N com os que a ele se ligam com arestas de maior peso.

Um exemplo é mostrado na Figura 2.4.

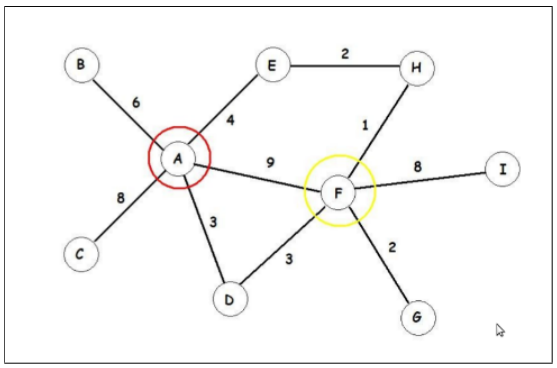


Figura 2.4: Exemplo de Max Aggregation (retirado de [23])

Neste grafo, o nó N selecionado corresponde ao vértice A (a vermelho), porque este é o mais pesado do grafo, ou seja, a soma dos pesos das arestas é o maior do grafo. Assim, $N = A$. Em seguida é procurado o vértice de peso máximo (Pmax)

em relação a A. Esse vértice, neste grafo, é o F (a amarelo) que possui uma aresta de peso 9 em relação ao nó A. Como não existe mais nenhum nó com peso igual a 9 (Pmax), terminamos a geração deste grupo que será constituído por 2 nós (A e F) e estes formarão um novo vértice no grafo. De seguida, o algoritmo irá procurar os próximos nós mais pesados que são o nó E e o nó H, ambos com peso 2 e o processo repete-se até não restarem mais vértices no grafo.

2.2.4 Método da Bisseção Espectral Recursivo

Os métodos recursivos de bisseção espectral são métodos de particionamento globais que baseiam-se na ideia de separar os vértices do grafo em dois grupos, atribuindo valores -1 ou +1 a cada vértice. Um subconjunto de vértices é criado com os que têm valor -1 e um outro subconjunto é criado com os vértices que têm valor +1. Todo método espectral é baseado na determinação de autovalores e é utilizada a matriz Laplaciana, pois este tipo de matriz possui propriedades importantes. Seja $u_1, u_2 \dots u_n$ vetores próprios normalizados de L com valores próprios $\lambda_1 \leq \lambda_2 \cdots \leq \lambda_n$ correspondentes, a matriz Laplaciana L tem as seguintes propriedades:

1. L é simétrica e semidefinida positiva;
2. Os vetores próprios u_i são ortogonais dois a dois;
3. $u_1 = n^{-\frac{1}{2}}\mathbf{1}, \lambda_1 = 0$;
4. Se G é conexo, então λ_1 é o único valor próprio nulo de L.

Se G for conexo então λ_2, o segundo maior valor próprio da matriz L, será negativo. A magnitude de λ_2 é uma medida de conectividade do grafo e o que importa notar é que o vetor próprio u_2, associado à λ_2, dá informação direcional sobre o grafo. Estando os componentes de u_2 associados aos vértices do grafo, estes correspondem aos pesos dos vértices. Ao ordenar os vértices pelo seu peso, podemos particioná-los, calculando

a mediana dos pesos e colocando vértices com valor acima da mediana num subgrafo e vértices com valor abaixo da mediana em outro subgrafo.

A forma de solução do problema, como definida por Pothen, Simon and Liou [17] é baseada na computação de um valor próprio específico de uma matriz Laplaciana do grafo definida como:

$$l_{ij} = \begin{cases} +1 & \text{se } (v_i, v_j) \in E \\ -deg(v_i) & \text{se } i = j \\ 0 & \text{caso contrário} \end{cases}$$

onde $deg(v_i)$ representa o grau do vértice v_i e E representa o conjunto de arestas do grafo. Podemos notar que $L(G) = -D + A$, onde A é a matriz de adjacência do grafo e D é a matriz diagonal dos graus dos vértices.

Uma forma alternativa de representação da matriz Laplaciana é apresentada por Hendrickson e Leland [6], onde a matriz é definida como:

$$l_{ij} = \begin{cases} -1 & \text{se } (v_i, v_j) \in E \\ deg(v_i) & \text{se } i = j \\ 0 & \text{caso contrário} \end{cases}$$

Nesta definição podemos notar que $L(G) = D - A$, diferente da definição dada acima $L(G) = -D + A$.

Estas representações são equivalentes e produzem os mesmos vetores próprios.

Capítulo 3

Utilização do método de Bisseção

Neste capítulo apresentamos mais detalhadamente o algoritmo de bisseção utilizado para particionar os grafos de restrições.

3.1 Algoritmos

Considere o grafo G(V,E) não orientado e conexo. Atribua uma variável x_i a cada vértice v_i tal que $x_i = \pm 1$ e $\sum_i x_i = 0$. A primeira condição estipula uma partição de v em dois conjuntos disjuntos. A segunda condição requer que os conjuntos sejam de tamanho igual (assumindo um número par de vértices). A matriz Laplaciana do grafo G, $L(G) = L_{i,j}$, é uma matriz de ordem n, como definida na Seção 2.2.4.

A função $f(x) = \frac{1}{4}\sum_{|E|}(x_i - x_j)^2$ conta o número de arestas entre os conjuntos particionados. A função $f(x)$ pode também ser definida como $f(x) = \frac{1}{4}x^T Lx$. Juntando este facto com restrições de x, é definido o problema da bisseção, caso discreto, como:

Minimizar $\frac{1}{4}x^T Lx$

Sujeito a: $x^T 1 = 0$, $x_i = \pm 1$, onde 1 é o n-vetor$(1, 1, \ldots, 1)^T$.

O particionamento de grafos é um problema NP-difícil. Portanto, há a necessidade de relaxar as restrições dos valores discretos sobre o vetor x e definir o problema de

bisseção do vetor x com restrições com valores contínuos da seguinte forma.

Minimizar $\frac{1}{4}x^T Lx$

Sujeito a: $x^T 1 = 0$, $x^T x = n$

onde os elementos de x podem assumir qualquer valor satisfazendo as restrições.

O algoritmo de bisseção espectral definido e implementado por Hendrickson e Leland [6] pode ser visto abaixo.

Algoritmo BISSECAO(L,n);

1. *Calcular o segundo vetor próprio da matriz L;*
2. *Calcular o vetor* $x = \sqrt{n}u_2$*;*
3. *Calcular a mediana m dos valores de x;*
4. *Calcular o vetor* x'*, onde* $x'_i = -1$*, se* $x_i < m$ *ou* $x'_i = +1$*,se* $x_i > m$*;*
   Se houver valores iguais a m, manter o balanceamento;
5. *Calcular o valor de* $f(x') = \frac{1}{4}\sum_{|E|}(x'_i - x'_j)^2$*;*
6. *Balancear as partições com o algoritmo de KL.*
7. *end-BISSECAO*

Os parâmetros de entrada do algoritmo $BISSECAO$ são a matriz Laplaciana L e n que corresponde ao número de vértices do grafo, e também é a ordem da matriz. Se o grafo dado tiver um número ímpar de vértices é necessário acrescentar mais um vértice para termos o mesmo número de vértices nos dois subconjuntos após realizarmos a bisseção. Neste algoritmo, se as partições não estiverem balanceadas, utiliza-se o método de Kernighan e Li (KL) [9] para balanceá-las. Neste trabalho, utilizamos grafos com número par ou ímpar de vértices. Ou seja, a condição de balanceamento não é atendida.

3.2 Exemplo de aplicação do algoritmo

Na Figura 3.1 é apresentado um grafo G com quatro vértices. Sua matriz Laplaciana é representada pela Figura 3.2:

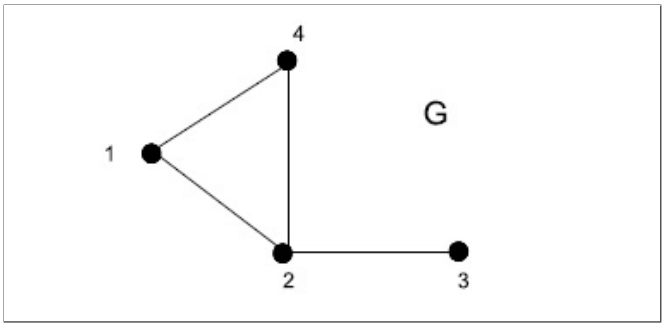


Figura 3.1: Grafo G com 4 vértices(retirado de [16])

$$L(G) = \begin{bmatrix} 2 & -1 & 0 & -1 \\ -1 & 3 & -1 & -1 \\ 0 & -1 & 1 & 0 \\ -1 & -1 & 0 & 2 \end{bmatrix}$$

Figura 3.2: Matriz Laplaciana de um grafo G com quatro vértices(retirado de [16])

Sejam v um vetor próprio e λ um valor próprio da matriz L. Pela definição de vetor próprio, podemos afirmar que $Lv{=}\lambda v$. Considere $v{=}Iv$, onde I é a matriz identidade. Então:

$$Lv - \lambda v = 0$$
$$(L - \lambda I) = 0$$

Neste caso o vetor próprio v não pode ser nulo. O polinômio característico $P(\lambda) = det(L - \lambda I) = 0$. Ao resolvermos este determinante para o exemplo dado encontramos uma equação em λ:

$$\lambda^4 - 8\lambda^3 + 19\lambda^2 - 12\lambda = 0$$

A solução desta equação nos fornece 4 valores próprios: $\lambda_1 = 0, \lambda_2 = 1, \lambda_3 = 3$ e $\lambda_4 = 4$.

Estamos interessados no segundo valor próprio, que é $\lambda_2 = 1$. Queremos encontrar o vetor próprio correspondente para este valor próprio. Ao resolvermos $Lv = \lambda v$, encontramos uma família de vetores próprios, $u_2 = (w, 0, -2w, w)$.

Se considerarmos $w = 1$, então, $u_2 = (1, 0, -2, 1)$ é um segundo vetor próprio associado ao valor próprio $\lambda_2 = 1$.

O vetor indicador x é dado por $x = \sqrt{u}u_2$, onde $n = 4$ é o número de vértices do grafo. Assim, $x = (2, 0, -4, 2)$. Para passarmos à solução do problema contínuo para uma partição discreta é necessário calcular a mediana e escrever este vetor em função de $1's$ e $-1's$.

Uma maneira utilizada para calcular a mediana consiste em ordenar os elementos do vetor x em ordem crescente $(-4, 0, 2, 2)$. Depois, a partir dos dois valores centrais, tirar a média aritmética (que é 1). Se o número de vértices for ímpar, consideramos que a mediana é o valor central. Em seguida, basta escrever o vetor x', de forma que os elementos maiores que a mediana em x tenham o valor 1 em x' e aqueles menores que a mediana tenham valor -1. Para valores iguais à mediana é necessário fazer uma análise para decidir em que conjunto estes vértices serão atribuídos de modo que os conjuntos fiquem balanceados.

Neste exemplo, $x' = (1, -1, -1, 1)$. Portanto, os vértices v_1 e v_4 estão num conjunto e os demais estão em outro conjunto.

A função $f(x) = \frac{1}{4}\sum_{|E|}(x_i - x_j)^2 = 2$ calcula o número de arestas que existe entre os conjuntos. Percebemos, então, que existem duas arestas entre os conjuntos na partição gerada, como mostra a Figura 3.3.

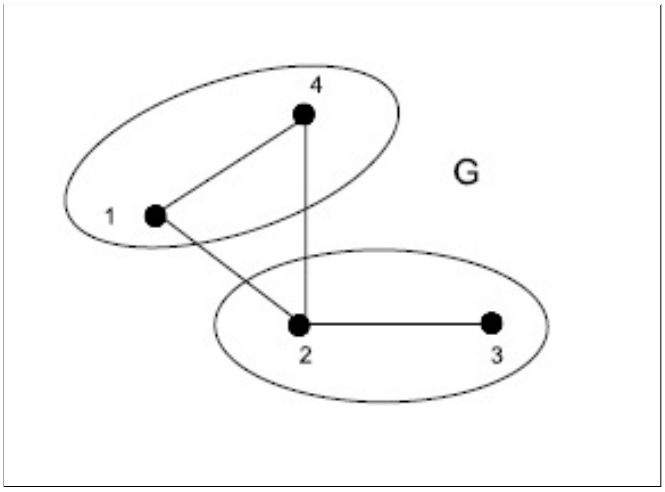


Figura 3.3: Bisseção do grafo(retirado de [16])

3.3 Implementação

Este algoritmo foi implementado na linguagem *Python* e está descriot no Algoritmo 1. O código fonte pode ser visto no Apêndice A.

Os parâmetros de entrada do algoritmo *bissecao* são a matriz Laplaciana L e o número de vértices do grafo, n, que também é a ordem da matriz. Gerar novos grafos G_1 e G_2, se o número de vértices de G_1 ou G_2 for menor ou igual a 3, imprimir os nós do grafo, caso contrário, gerar nova matriz e chamar a funcão recursivamente. O parâmetro `mapa` é utilizado para mapear os vértices do grafo original nos vértices do novo grafo já representado como a matriz laplaciana. Para os valores do vetor x' que têm valor igual a zero (vetor x tinha algum valor igual ao valor da mediana), agrupamos os vértices correspondentes num novo grafo G_3 e aplicamos a bisseção novamente a este novo grafo.

bissecao(L,n,mapa);
begin;
Calcular o segundo vetor próprio da matriz L;
Calcular o vetor $x = \sqrt{n}u_2$;
Calcular a mediana m dos valores de x;
Calcular o vetor x', onde $x'_i = -1$, se $x_i < m$ ou $x'_i = +1$,se $x_i > m$;
Se houver valores iguais a m, formar um grupo;
Conta quantos 1 e -1 e 0 estão no vetor x';
if $conta_{um}$ *ou* $conta_{menorum}$ *ou* $conta_{zero} \leq 3$ **then**
    imprimir nó do grafo
end
if $conta_{um} > 3$ **then**
    gerar novamatriz G_1 com tamanho $conta_{um}$, $bissecao(G_1, conta_{um}$,mapa)
end
if $conta_{menorum} > 3$ **then**
    gerar novamatriz G_2 com tamanho $conta_{menorum}$,
    $bissecao(G_2, conta_{menorum}$,mapa)
end
if $conta_{zero} > 3$ **then**
    gerar novamatriz G_3 com tamanho $conta_{zero}$, $bissecao(G_3, conta_{zero}$,mapa)
end
end

Algorithm 1: *bissecao*(L,n,mapa)

Capítulo 4

Materiais e Métodos

Neste capítulo descrevemos os materiais e a metodologia utilizados neste trabalho. Comparamos a nossa implementação do método de bisseção recursiva com um método de particionamento em grafos baseado na escolha de vértices com maior número de arestas, cujo algoritmo foi apresentado na Seção 2.2.3 (Max Aggregation). Comparamos também os resultados dos dois algoritmos com o grafo original. As métricas de comparação são:

- número total de vértices do grafo original e número total de vértices dos grafos particionados com os dois algoritmos, considerando cada grupo de vértices do grafo original numa partição como sendo um único vértice no grafo particionado;
- número total de arestas do grafo original e número total de arestas dos grafos particionados;
- densidade dos grafos. A densidade representa o percentual de conexão entre os nós. O cálculo de densidade é realizado dividindo-se o número de arestas e pelo número de arestas num grafo correspondente completo com n nós ($densidade = \frac{e}{[n*(n-1)]/2}$).
- grau médio dos vértices.

- número de arestas no corte do grafo.

A Tabela 4.1 apresenta as principais características dos grafos usados nos experimentos. Nesta tabela, os grafos estão organizados em três diferentes grupos de acordo com a sua densidade. Os grupos foram gerados utilizando um algoritmo de agrupamento baseado no algoritmo k-means com k=3. O primeiro grupo (grafos 1-5) foi considerado pouco denso. O segundo grupo foi considerado como de densidade média. O último grupo (com apenas um grafo representativo, Grafo 10) é de alta densidade (grafo totalmente conexo).

Os resultados são apresentados na forma gráfica (produzidos com a ferramenta `dot`) e na forma de tabela comparando as métricas. Três grafos são apresentados para cada experimento, o grafo original, o grafo obtido com o algoritmo Max Aggregation e a o grafo obtido com o algoritmo de bisseção espectral recursiva.

A implementação do método de Bisseção Recursiva Espectral foi feita em Python, versão 2.7. Para calcular os valores próprios e vetores próprios utilizamos a biblioteca `sympy`. Esta biblioteca tem uma vantagem sobre outras com o mesmo propósito (por exemplo, `numpy` e `scipy`), porque faz o máximo possível de cálculos de forma algébrica e somente no fim realiza cálculos numéricos, reduzindo, assim, a propagação de erros.

4.1 Comparação entre os dois algoritmos

O algoritmo de Max Aggregation tem apenas como finalidade ler da entrada um conjunto de dados numéricos correspondentes aos vértices e arestas do grafo, verificar dependências entre eles através do número de variáveis em comum, gerar o grafo (fazer ligações entre os vértices lidos da entrada), começar o processo de geração de grupos e colocá-los num ficheiro com um formato pré-definido, que pode servir de entrada a outro programa que irá fazer as operações correspondentes a cada subconjunto de vértices.

Tabela 4.1: Características dos Grafos Originais

	Densidade Baixa					Densidade Média				Densidade Alta
Grafo	1	2	3	4	5	6	7	8	9	10
Vértices	4	6	14	4	5	4	5	5	9	4
Número de arestas	3	9	55	4	7	5	9	9	33	6
Densidade	0.5	0.6	0.60	0.66	0.7	0.83	0.9	0.9	0.91	1
Grau médio dos vértices	1.5	3	7.85	2	3.2	2.5	3.4	3.6	7.3	3

O algoritmo de bisseção espectral recursivo cria conjuntos balanceados e conexos, que produzem um particionamento visualmente mais agradável, porém não necessariamente melhor. O algoritmo de bisseção espectral tende a gerar o menor número de arestas entre os subconjuntos. Utilizando o método de bisseção recursiva, quanto maior o número de partes que queremos obter de um grafo, maior é o número de vetores próprios computados. Os algoritmos de bisseção apresentam alguns problemas. Por exemplo, eles não aceitam um corte inicial menos atrativo, que produziria, mais tarde, redes com cortes melhores. Ou seja, estes algoritmos não possuem *lookahead*. Em bisseção, a tarefa de dividir o grafo em conjuntos de vértices (decomposição do problema) é separado da atribuição de vértices para um processador específico (problema de atribuição). O *overhead* de comunicação em um programa depende da decomposição e da atribuição. Consequentemente, é preferível considerar estes aspectos do problema juntos. Por exemplo, poderíamos escolher dois conjuntos com maior volume de comunicação entre eles para colocá-los topologicamente juntos numa arquitetura.

Capítulo 5

Resultados

Os métodos usados neste trabalho foram Max Aggregation e Bisseção Recursiva Espectral. Os grafos foram divididos em 3 grupos: baixa densidade, densidade média e alta densidade. Nas seções seguintes mostramos os resultados do particionamento de cada um dos métodos e comparamos. Procuramos também estabelecer se algum método é adequado para algum dos grupos de grafos.

As figuras relativas aos grafos 1 a 5 (Figuras 5.1 a 5.5) correspondem aos grafos de baixa densidade. As Figuras 5.6 a 5.9 correspondem aos grafos de densidade média. A Figura 5.10 corresponde a um grafo de densidade alta. Cada uma destas figuras mostra o grafo original e os dois grafos particionados com o método Max Aggregation e com o método de Bisseção Recursiva Espectral. O grafo original representa um CSP sintético, onde cada vértice corresponde a uma restrição e cada aresta corresponde às variáveis que conectam um par de restrições. No grafo particionado, cada subgrafo corresponde a um subconjunto de restrições. As arestas entre os subgrafos (blocos) correspondem às variáveis comuns aos conjuntos de restrições de cada bloco. O algoritmo Max Aggregation leva em consideração que o peso entre as arestas é o número de variáveis comuns entre blocos. O algoritmo de bisseção espectral assume que o peso entre os vértices é igual a um.

A Tabela 5.1 mostra um resumo das características dos grafos original e particiona-

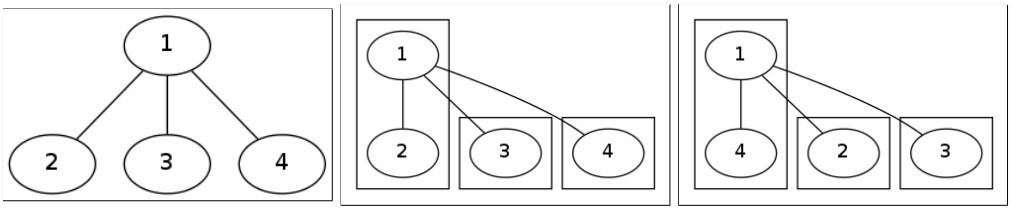
dos em relação ao número de vértices, número de arestas, densidade e grau médio de cada vértice.

Apresentamos ainda um resumo das diferenças entre os dois algoritmos para cada métrica usada na avaliação. Estes resultados são mostrados nas tabelas 5.2 a 5.5, onde um sinal de "-" indica uma redução, um sinal de "+" indica um aumento e um sinal de "=" indica que não houve alteração de particionamento de um grafo para o outro. $Original \rightarrow MA$ significa a diferença do grafo original para o grafo particionado com o método Max Aggregation. $Original \rightarrow RB$ significa a diferença do grafo original para o grafo particionado com o método de bisseção espectral recursiva. $MA \rightarrow RB$ significa a diferença do método Max Aggregation para o método da Bisseção Recursiva.

5.1 Grupo de grafos com densidade baixa

No grupo de densidade baixa, ambos os particionamentos são equivalentes para o Grafo1, apenas havendo uma troca de vértices (2 com 4 nas Figuras 5.1b e 5.1c).

O Grafo2 pôde ser particionado com o método Max Aggregation, mas o método de Bisseção Recursiva Espectral não produziu solução devido a um problema no ambiente de execução Python (a versão utilizada foi a 2.7). Uma das funções da biblioteca sympy é fazer o máximo possível de processamento algébrico, apenas fazendo cálculos numéricos no final. Para este grafo em particular, quando a biblioteca recebe uma matriz com valores algébricos para remover espaços nulos (parte do "parser" da biblioteca, método `nullspaces`), o interpretador perde-se na computação. O método para remoção de espaços nulos de uma matriz algébrica funciona bem quando não é chamado no contexto do programa. Fizemos um teste com um pequeno programa que passa como parâmetro para o método `nullspaces` uma matriz algébrica e o interpretador não retorna resultados no programa, mas retorna resultados se a mesma chamada for feita no "prompt" do interpretador. Se a matriz tiver apenas valores numéricos, a biblioteca comporta-se bem.



(a) Original (b) Max Aggregation (c) Recursive Bisection

Figura 5.1: Grafo1

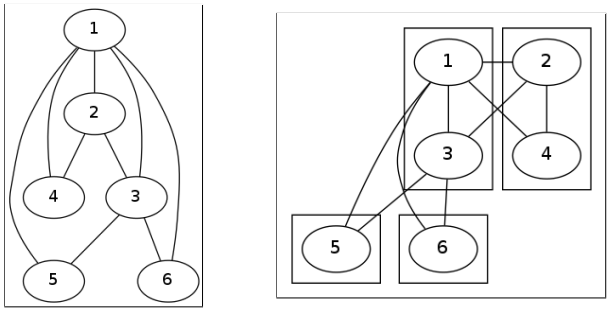
O particionamento do Grafo3 (Figura 5.3), utilizando o método de bisseção espectral recursiva, gerou um grupo para cada vértice, o que não muda as características do grafo original. Já o método de Max Aggregation conseguiu dois grupos de dois vértices, o que representa uma melhora modesta no número de arestas do grafo particionado em relação ao grafo original.

Para o Grafo4 (Figura 5.4), notamos que o Max Aggregation faz uma escolha de partições que possui um número de arestas entre grupos (número de arestas do corte) maior do que o número de arestas produzido pelo método de bisseção espectral recursiva. O número de grupos também é menor para este último método, indicando a necessidade de menor número de processadores para a execução das restrições do grafo. Na prática, pode haver a necessidade de se adicionar mais processadores, portanto este método está sendo conservador para este grafo.

Os particionamentos obtidos para o Grafo5 (Figura 5.5) são equivalentes, havendo apenas uma troca dos vértices (2,3) com (4,5) no grupo com maior número de vértices. Como não estamos levando em consideração a quantidade de processamento por vértice, estes particionamentos podem fazer diferença na prática.

5.2 Grupo de grafos com densidade média

No grupo de densidade média, o Grafo6 (Figura 5.6) teve diferentes particionamentos com Max Aggregation e com bisseção espectral recursiva. Se levarmos em conside-



(a) Original (b) Max Aggregation

Figura 5.2: Grafo2

ração o número de arestas do grafo particionado, Max Aggregation fez um melhor particionamento, onde apenas dois processadores seriam necessários com uma troca de mensagens menos pesada (3 arestas) do que a troca de mensagens requerida pelos três subgrafos gerados pelo método de bisseção recursiva espectral.

No Grafo7 (Figura 5.7), apesar dos grupos de vértices obtidos por Max Aggregation estarem mais bem balanceados do que os grupos gerados pela bisseção espectral, o número de arestas entre os grupos é maior do que no grafo obtido com a bisseção espectral. Na prática isto poderia indicar um menor número de comunicações entre os grupos.

O Grafo8 (Figura 5.8) teve particionamentos equivalentes usando ambos os métodos, apenas com uma troca dos vértices 2 e 4 pelos grupos.

O Grafo9 (Figura 5.9) foi particionado em quatro subgrupos com Max Aggregation e em seis subgrupos com bisseção recursiva. Embora o método de bisseção recursiva tenha um número de arestas maior cruzando os grupos, este particionamento, na prática, pode ser vantajoso do ponto de vista da execução paralela, visto que trocas de mensagens através de arestas em grupos não relacionados podem ser efetuadas simultaneamente. Além disto, esta solução é mais flexível porque permite a utilização de um maior número de processadores do que a solução produzida pelo Max Aggregation.

5.3 Grafo de densidade alta

O único representante dos grafos de densidade alta, Grafo10 (Figura 5.7), foi particionado em dois grupos com Max Aggregation e em três grupos com bisseção espectral recursiva. Apesar do número de arestas de corte de Max Aggregation ser melhor do que para bisseção espectral, o facto do vértice 2 poder fazer comunicações simultâneas com dois processadores distintos (grupo que faz o processamento das restrições do vértice 1 e grupo que faz o processamento dos vértices 3 e 4), pode ser uma vantagem, na prática.

5.4 Discussão

Podemos observar um panorama geral de comportamento dos métodos de particionamento através da Tabela 5.1 e das Tabelas 5.2 a 5.5.

Em relação ao número de vértices do grafo particionado, o método Max Aggregation é capaz de reduzir este número para todos os grafos estudados em relação ao grafo original, de acordo com a Tabela 5.2. O método de Bisseção Recursiva Espectral mantém o número original de vértices para um dos grafos (Grafo3) e não obtém solução para um dos outros grafos (Grafo2). O mesmo comportamento se verifica para o número de arestas (Tabela 5.3).

Não se verifica uma mudança de comportamento entre os dois métodos em relação ao número de vértices e número de arestas (linha $MA \rightarrow RB$ das Tabelas 5.2 e 5.3), com exceção do Grafo7, onde o número de vértices manteve-se o mesmo para ambos os particionamentos (Tabela 5.2) e o número de arestas reduziu de Max Aggregation para o método de Bisseção (Tabela 5.3).

O método de Max Aggregation aumentou a densidade de todos os grafos originais após o particionamento (Tabela 5.4) enquanto o método de Bisseção manteve a densidade do grafo original, para o Grafo3, ao fazer um particionamento onde cada vértice

ficou num subgrupo individual (Figura 5.3).

A característica que foi mais diferenciada entre os dois particionamentos foi o grau médio dos vértices nos grafos particicionados. A Tabela 5.5 mostra estas diferenças. O método de Max Aggregation conseguiu reduzir o grau médio dos vértices em apenas um caso, para o Grafo1, enquanto o método de Bisseção Recursiva conseguiu reduzir o grau médio dos vértices em dois casos: Grafo1 e Grafo6.

Em termos de comportamento por grupo de grafos (densidades baixa, média ou alta) não há uma tendência clara. Se considerarmos apenas como métrica de avaliação o número de arestas do grafo particionado (tamanho do corte dos grafos), o algoritmo Max Aggregation parece o método mais indicado, o que contraria de certa forma a literatura, pois o "gold standard" para particionamento de grafos para execução paralela é o de Bisseção Recursiva Espectral, porque é o método na literatura que produz melhores cortes em grafos. Por outro lado, se considerarmos o novo número de vértices dos grafos particionados, observamos que o método de bisseção espectral recursiva, em geral, resulta em um número maior de vértices do que o método de Max Aggregation, o que pode favorecer trocas de mensagens simultâneas entre vértices de diferentes grupos.

Tabela 5.1: Características dos Grafos

		Densidade Baixa					Densidade Média				Densidade Alta
	Grafo	1	2	3	4	5	6	7	8	9	10
Grafo Original	Vértices	4	6	14	4	5	4	5	5	9	4
	Número de arestas	3	9	55	4	8	5	9	9	33	6
	Densidade	0.5	0.6	0.6	0.66	0.8	0.83	0.9	0.9	0.91	1
	Grau médio dos vértices	1.5	3	7.85	2	3.2	2.5	3.4	3.6	7.3	3
Max Aggregation	Vértices	3	4	12	3	3	2	3	3	4	2
	Número de arestas	2	7	45	3	6	3	7	6	28	3
	Densidade	0.66	1.16	0.68	1	2	3	2.33	2	4.66	3
	Grau médio dos vértices	1.33	3.5	8.83	2	4	3	4.33	4	14.25	3
Recursive Bisection	Vértices	3		14	2	3	3	3	3	6	3
	Número de arestas	2		55	2	6	4	6	6	30	5
	Densidade	0.66		0.6	2	2	1.33	2	2	1.5	1.66
	Grau médio dos vértices	1.33		7.85	2	4	2.66	4	4	10.66	3.33

Tabela 5.2: Número de Vértices dos Grafos

		Densidade Baixa					Densidade Média				Densidade Alta
	Grafo	1	2	3	4	5	6	7	8	9	10
Vértices	Grafo original	4	6	14	4	5	4	5	5	9	4
	Max Aggregation(MA)	3	4	12	3	3	2	3	3	4	2
	Recursive Bisection(RB)	3		14	2	3	3	3	3	6	3
Comparação	Original → MA	-	-	-	-	-	-	-	-	-	-
	Original → RB	-		=	-	-	-	-	-	-	-
	MA → RB	=		+	-	=	+	=	=	+	+

Tabela 5.3: Número de arestas dos Grafos

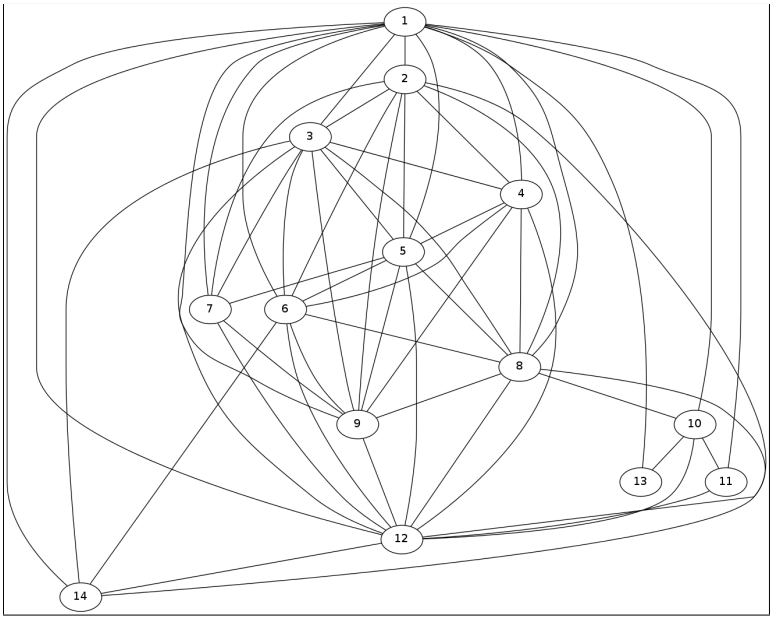
		Densidade Baixa					Densidade Média				Densidade Alta
	Grafo	1	2	3	4	5	6	7	8	9	10
Nº arestas	Grafo original	3	9	55	4	8	5	9	9	33	6
	Max Aggregation(MA)	2	7	45	3	6	3	7	6	28	3
	Recursive Bisection(RB)	2		55	2	6	4	6	6	30	5
Comparação	Original → MA	-	-	-	-	-	-	-	-	-	-
	Original → RB	-		=	-	-	-	-	-	-	-
	MA → RB	=		+	-	=	+	-	=	+	+

Tabela 5.4: Densidade dos Grafos

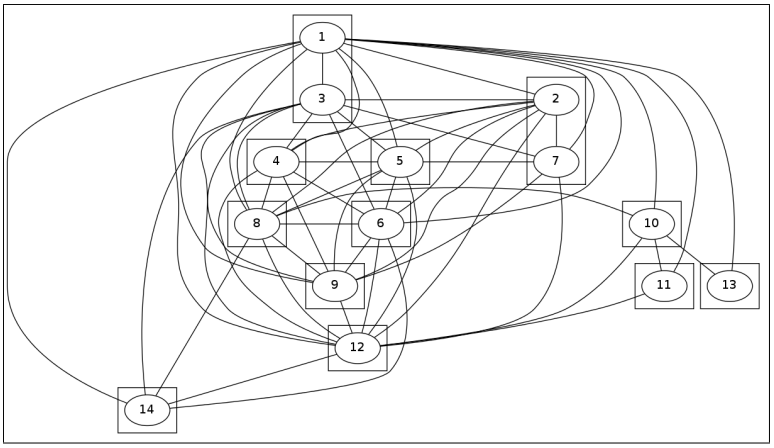
		Densidade Baixa					Densidade Média				Densidade Alta
	Grafo	1	2	3	4	5	6	7	8	9	10
Densidade	Grafo original	0.5	0.6	0.6	0.66	0.8	0.83	0.9	0.9	0.91	1
	Max Aggregation(MA)	0.66	1.16	0.68	1	2	3	2.33	2	4.66	3
	Recursive Bisection(RB)	0.66		0.6	2	2	1.33	2	2	1.5	1.66
Comparação	Original → MA	+	+	+	+	+	+	+	+	+	+
	Original → RB	+		=	+	+	+	+	+	+	+
	MA → RB	=		-	+	=	-	-	=	-	-

Tabela 5.5: Grau médio dos vértices

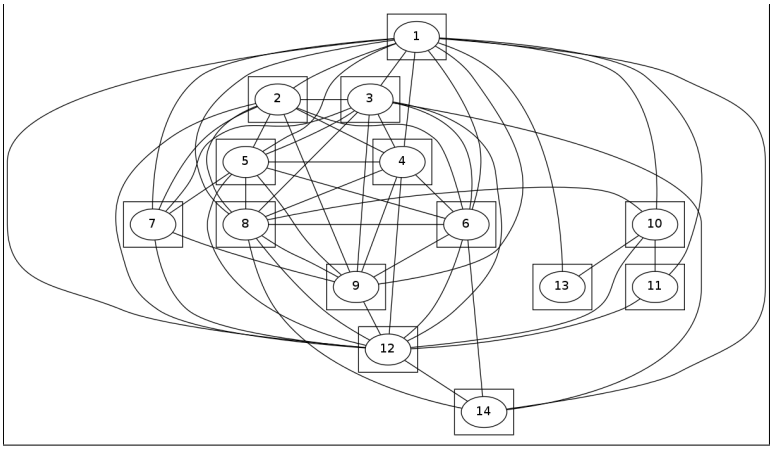
		Densidade Baixa					Densidade Média				Densidade Alta
	Grafo	1	2	3	4	5	6	7	8	9	10
Grau médio dos vértices	Grafo original	1.5	3	7.85	2	3.2	2.5	3.4	3.6	7.33	3
	Max Aggregation(MA)	1.33	3.5	8.83	2	4	3	4.33	4	14.25	3
	Recursive Bisection(RB)	1.33		7.85	2	4	2.33	4	4	10.66	3.33
Comparação	Original → MA	-	+	+	=	+	+	+	+	+	=
	Original → RB	-		=	=	+	-	+	+	+	+
	MA → RB	=		-	=	=	-	-	=	-	+



(a) original

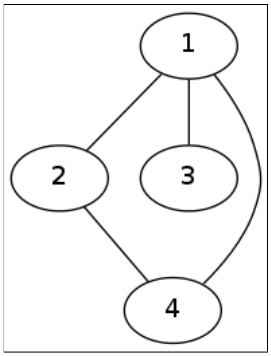


(b) Max Aggregation

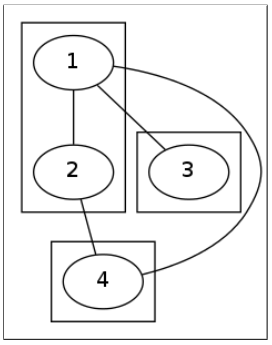


(c) Recursive Bisection

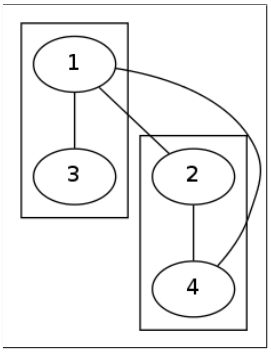
Figura 5.3: Grafo3



(a) Original

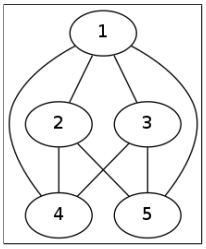


(b) Max Aggregation

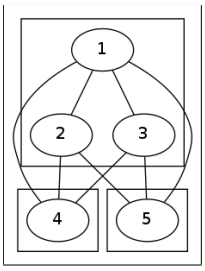


(c) Recursive Bisection

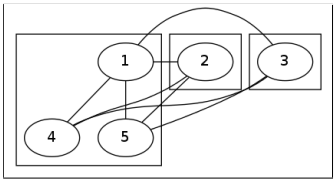
Figura 5.4: Grafo4



(a) Original

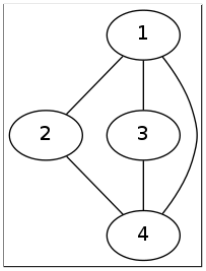


(b) Max Aggregation

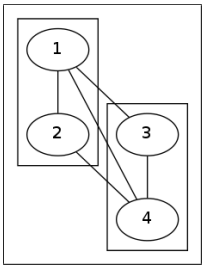


(c) Recursive Bisection

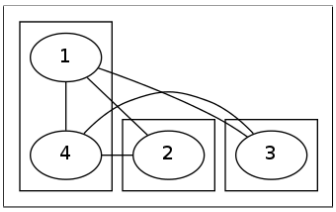
Figura 5.5: Grafo5



(a) Original

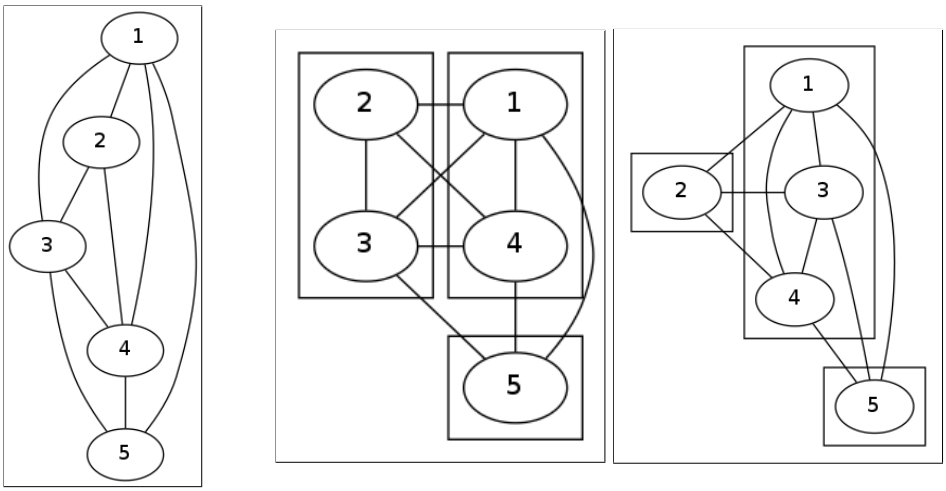


(b) Max Aggregation



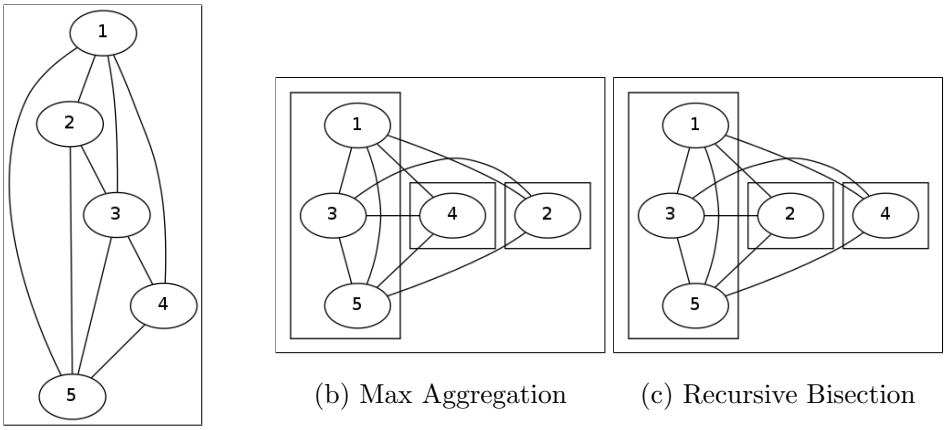
(c) Recursive Bisection

Figura 5.6: Grafo6



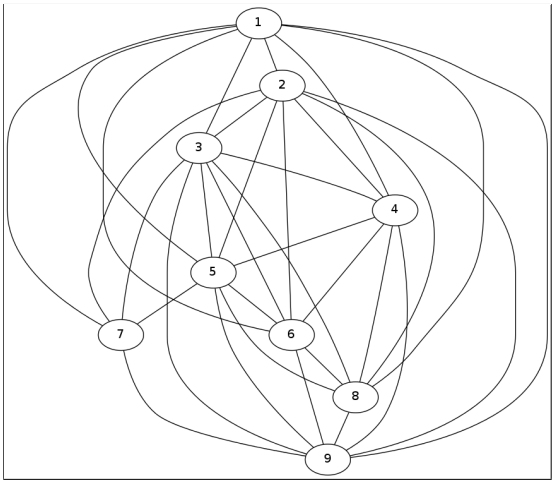
(a) Original (b) Max Aggregation (c) Recursive Bisection

Figura 5.7: Grafo7

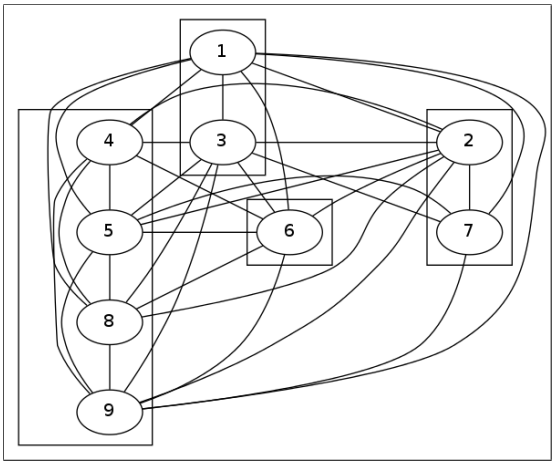


(a) Original (b) Max Aggregation (c) Recursive Bisection

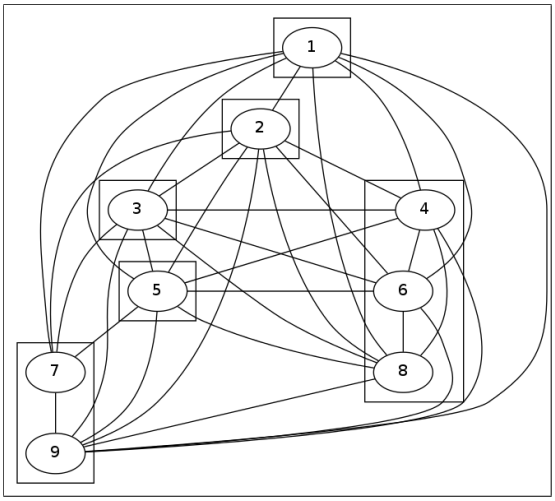
Figura 5.8: Grafo8



(a) original

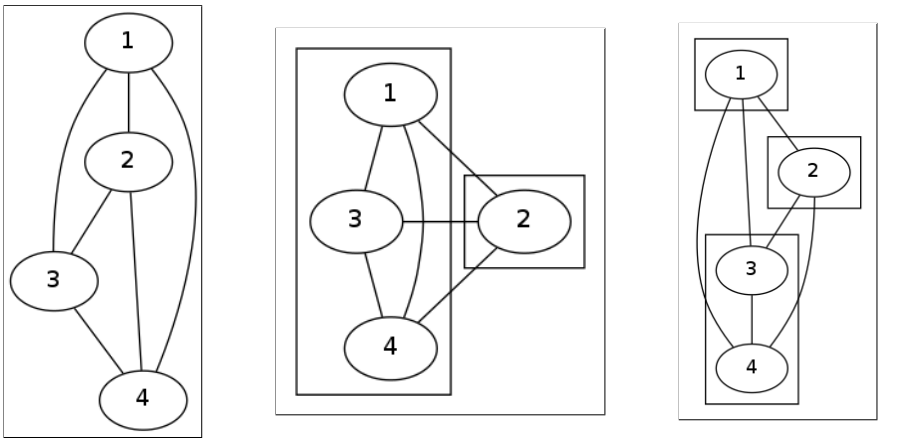


(b) Max Aggregation



(c) Recursive Bisection

Figura 5.9: Grafo9



(a) Original (b) Max Aggregation (c) Recursive Bisection

Figura 5.10: Grafo10

Capítulo 6

Conclusões e Trabalhos Futuros

Neste trabalho implementamos um algoritmo de particionamento de grafos baseado em bisseção espectral recursiva com o objetivo de dividir um CSP em subproblemas tão independentes quanto possível para posteriormente executar os diferentes subproblemas em diferentes processadores. Dos vários métodos disponíveis para particionamento de grafos, os métodos espectrais são os mais vantajosos porque levam em consideração a conectividade dos grafos, o que é importante para execução paralela, onde subgrupos de vértices representam computações e arestas representam comunicações.

Testamos 10 grafos com características diferentes e comparamos os resultados da bisseção espectral recursiva com um método baseado em corte mínimo em grafos ("min-cut"), chamado Max Aggregation. Resultados foram avaliados quanto ao número de vértices, número de arestas, densidade e grau médio dos vértices.

O algoritmo de Max Aggregation tende a ser mais conservador do que o algoritmo de bisseção espectral. Este tem a tendência de ser mais agressivo no particionamento gerando um número maior de grupos no grafo particionado do que o número de grupos gerado pelo Max Aggregation. Na prática, este número maior de grupos pode ser benéfico já que poderá haver um grande número de processadores disponíveis.

Neste trabalho, o método de bisseção espectral não leva em consideração que o grafo

pode ter pesos nas arestas (quantidade de comunicação). Seria interessante modelar este tipo de grafo na matriz laplaciana e verificar os resultados de particionamento. Também seria interessante produzir particionamentos para grafos maiores (centenas ou milhares de vértices) e testar estes particionamentos num sistema paralelo de execução de restrições. Tal sistema existe, mas no contexto deste trabalho, não foi possível fazer estas experiências.

Apêndice A

Código fonte

```
import sympy as sp
import math
# o numero de vertices do grafo
n=6
#matriz_original = [2,-1,0,-1,-1,3,-1,-1,0,-1,1,0,-1,-1,0,2]
#input_simples1
"""
matriz_original = [
   3, -1, -1, -1,
  -1,  1,  0,  0,
  -1,  0,  1,  0,
  -1,  0,  0,  1]
"""
#input_simple2

matriz_original = [
   5, -1, -1, -1, -1, -1,
  -1,  3, -1, -1,  0,  0,
  -1, -1,  4,  0, -1, -1,
  -1, -1,  0,  2,  0,  0,
  -1,  0, -1,  0,  2,  0,
  -1,  0, -1,  0,  0,  2 ]
"""
matriz_original = [
  -5,  1,  1,  1,  1,  1,
   1, -3,  1,  1,  0,  0,
   1,  1, -4,  0,  1,  1,
   1,  1,  0, -2,  0,  0,
   1,  0,  1,  0, -2,  0,
   1,  0,  1,  0,  0, -2 ]
"""
#input_simples3
"""
matriz_original=[
#  1   2   3   4   5   6   7   8   9  10  11  12  13  14
  13, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
  -1,  9, -1, -1, -1, -1, -1, -1, -1,  0,  0, -1,  0,  0,
```

```
  -1, -1, 10, -1, -1, -1, -1, -1, -1,  0,  0, -1,  0, -1,
  -1, -1, -1,  8, -1, -1,  0, -1, -1,  0,  0, -1,  0,  0,
  -1, -1, -1, -1,  9, -1, -1, -1, -1,  0,  0, -1,  0,  0,
  -1, -1, -1, -1, -1,  9,  0, -1, -1,  0,  0, -1,  0, -1,
  -1, -1, -1,  0, -1,  0,  6,  0, -1,  0,  0, -1,  0,  0,
  -1, -1, -1, -1, -1, -1,  0, 10, -1, -1,  0, -1,  0, -1,
  -1, -1, -1, -1, -1, -1, -1, -1,  9,  0,  0, -1,  0,  0,
  -1,  0,  0,  0,  0,  0,  0, -1,  0,  5, -1, -1, -1,  0,
  -1,  0,  0,  0,  0,  0,  0,  0,  0, -1,  3, -1,  0,  0,
  -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 12,  0, -1,
  -1,  0,  0,  0,  0,  0,  0,  0,  0, -1,  0,  0,  2,  0,
  -1,  0, -1,  0,  0, -1,  0, -1,  0,  0,  0, -1,  0,  5]
"""
"""
matriz_original=[
#  1   2   3   4   5   6   7   8   9  10  11  12  13  14
 -13,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
   1, -9,  1,  1,  1,  1,  1,  1,  1,  0,  0,  1,  0,  0,
   1,  1,-10,  1,  1,  1,  1,  1,  1,  0,  0,  1,  0,  1,
   1,  1,  1, -8,  1,  1,  0,  1,  1,  0,  0,  1,  0,  0,
   1,  1,  1,  1, -9,  1,  1,  1,  1,  0,  0,  1,  0,  0,
   1,  1,  1,  1,  1, -9,  0,  1,  1,  0,  0,  1,  0,  1,
   1,  1,  1,  0,  1,  0, -6,  0,  1,  0,  0,  1,  0,  0,
   1,  1,  1,  1,  1,  1,  0,-10,  1,  1,  0,  1,  0,  1,
   1,  1,  1,  1,  1,  1,  1,  1, -9,  0,  0,  1,  0,  0,
   1,  0,  0,  0,  0,  0,  0,  1,  0, -5,  1,  1,  1,  0,
   1,  0,  0,  0,  0,  0,  0,  0,  0,  1, -3,  1,  0,  0,
   1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,-12,  0,  1,
   1,  0,  0,  0,  0,  0,  0,  0,  0,  1,  0,  0, -2,  0,
   1,  0,  1,  0,  0,  1,  0,  1,  0,  0,  0,  1,  0, -5]
"""

#input_simples4
"""
matriz_original = [
   3, -1, -1, -1,
  -1,  2,  0, -1,
  -1,  0,  1,  0,
  -1, -1,  0,  2]
"""

#input_simple5
"""
matriz_original=[
   4, -1, -1, -1, -1,
  -1,  3,  0, -1, -1,
  -1,  0,  3, -1, -1,
  -1, -1, -1,  3,  0,
  -1, -1, -1,  0,  3]
"""

#input_simple 6
"""
matriz_original=[
   3, -1, -1, -1,
  -1,  2,  0, -1,
  -1,  0,  2, -1,
  -1, -1, -1,  3 ]
"""

#input_simples7
"""
matriz_original = [
   4, -1, -1, -1, -1,
  -1,  3, -1, -1,  0,
```

```
  -1, -1,  4, -1, -1,
  -1, -1, -1,  4, -1,
  -1,  0, -1, -1,  3]
"""
#input_simples8
"""
matriz_original=[
   4, -1, -1, -1, -1,
  -1,  3, -1,  0, -1,
  -1, -1,  4, -1, -1,
  -1,  0, -1,  3, -1,
  -1, -1, -1, -1,  4]
"""

#input_simples9
"""
matriz_original=[
   8, -1, -1, -1, -1, -1, -1, -1, -1,
  -1,  8, -1, -1, -1, -1, -1, -1, -1,
  -1, -1,  8, -1, -1, -1, -1, -1, -1,
  -1, -1, -1,  7, -1, -1,  0, -1, -1,
  -1, -1, -1, -1,  8, -1, -1, -1, -1,
  -1, -1, -1, -1, -1,  7,  0, -1, -1,
  -1, -1, -1,  0, -1,  0,  5,  0, -1,
  -1, -1, -1, -1, -1, -1,  0,  7, -1,
  -1, -1, -1, -1, -1, -1, -1, -1,  8]
"""

#input_simple 10
"""
matriz_original=[
   3, -1, -1, -1,
  -1,  3, -1, -1,
  -1, -1,  3, -1,
  -1, -1, -1,  3]
"""


#Outro Exemplo e solucao
"""
matriz_original=[
   3, -1, -1,  0, -1,  0,  0,  0,
  -1,  5, -1,  0, -1, -1, -1,  0,
  -1, -1,  4, -1,  0,  0, -1,  0,
   0,  0, -1,  2,  0,  0,  0, -1,
  -1, -1,  0,  0,  3, -1,  0,  0,
   0, -1,  0,  0, -1,  3, -1,  0,
   0, -1, -1,  0,  0, -1,  3,  0,
   0,  0,  0, -1,  0,  0,  0,  1]
"""
#Solucoes :  [-1, -1, 1, -1, -1, 1, 1, -1]
#media = 0
#Solucoes :  [0, 0, -1, 0, 1]
"""
matriz_original=[
   3,  1,  1,  0,  1,  0,  0,  0,
   1,  5,  1,  0,  1,  1,  1,  0,
   1,  1,  4,  1,  0,  0,  1,  0,
   0,  0,  1,  2,  0,  0,  0,  1,
   1,  1,  0,  0,  3,  1,  0,  0,
   0,  1,  0,  0,  1,  3,  1,  0,
   0,  1,  1,  0,  0,  1,  3,  0,
   0,  0,  0,  1,  0,  0,  0,  1]
```

```
"""
#Solucoes :  [-1, 1, -1, 1, 1, -1, 1, -1]
"""
matriz_original=[
#  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
   6, -1, -1,  0,  0, -1,  0,  0,  0, -1,  0,  0,  0, -1, -1,
  -1,  6, -1,  0,  0,  0, -1,  0,  0,  0, -1, -1,  0, -1,  0,
  -1, -1,  5,  0,  0,  0,  0, -1,  0,  0,  0,  0, -1,  0, -1,
   0,  0,  0,  7, -1, -1, -1, -1, -1, -1, -1,  0,  0,  0,  0,
   0,  0,  0, -1,  7, -1, -1, -1, -1,  0,  0, -1, -1,  0,  0,
  -1,  0,  0, -1, -1,  8, -1, -1,  0, -1,  0,  0,  0, -1, -1,
   0, -1,  0, -1, -1, -1,  8, -1,  0,  0, -1, -1,  0, -1,  0,
   0,  0, -1, -1, -1, -1, -1,  7,  0,  0,  0,  0, -1,  0, -1,
   0,  0,  0, -1, -1,  0,  0,  0,  6, -1, -1, -1, -1,  0,  0,
  -1,  0,  0, -1,  0, -1,  0,  0, -1,  7, -1,  0,  0, -1, -1,
   0, -1,  0, -1,  0,  0, -1,  0, -1, -1,  7, -1,  0, -1,  0,
   0, -1,  0,  0, -1,  0, -1,  0, -1,  0, -1,  7, -1, -1,  0,
   0,  0, -1,  0, -1,  0,  0, -1, -1,  0,  0, -1,  6,  0, -1,
  -1, -1,  0,  0,  0, -1, -1,  0,  0, -1, -1, -1,  0,  8, -1,
  -1,  0, -1,  0,  0, -1,  0, -1,  0, -1,  0,  0, -1, -1,  7]
"""
# media = 0; solucoes : [0, 0, 0, 1, -1, 0, 0, 0, 0, 0, -1, 1, 0, 0, 0]

"""
matriz_original=[
   6,  1,  1,  0,  0,  1,  0,  0,  0,  1,  0,  0,  0,  1,  1,
   1,  6,  1,  0,  0,  0,  1,  0,  0,  0,  1,  1,  0,  1,  0,
   1,  1,  5,  0,  0,  0,  0,  1,  0,  0,  0,  0,  1,  0,  1,
   0,  0,  0,  7,  1,  1,  1,  1,  1,  1,  1,  0,  0,  0,  0,
   0,  0,  0,  1,  7,  1,  1,  1,  1,  0,  0,  1,  1,  0,  0,
   1,  0,  0,  1,  1,  8,  1,  1,  0,  1,  0,  0,  0,  1,  1,
   0,  1,  0,  1,  1,  1,  8,  1,  0,  0,  1,  1,  0,  1,  0,
   0,  0,  1,  1,  1,  1,  1,  7,  0,  0,  0,  0,  1,  0,  1,
   0,  0,  0,  1,  1,  0,  0,  0,  6,  1,  1,  1,  1,  0,  0,
   1,  0,  0,  1,  0,  1,  0,  0,  1,  7,  1,  0,  0,  1,  1,
   0,  1,  0,  1,  0,  0,  1,  0,  1,  1,  7,  1,  0,  1,  0,
   0,  1,  0,  0,  1,  0,  1,  0,  1,  0,  1,  7,  1,  1,  0,
   0,  0,  1,  0,  1,  0,  0,  1,  1,  0,  0,  1,  6,  0,  1,
   1,  1,  0,  0,  0,  1,  1,  0,  0,  1,  1,  1,  0,  8,  1,
   1,  0,  1,  0,  0,  1,  0,  1,  0,  1,  0,  0,  1,  1,  7]
"""
#media = 0 Solucoes :  [0, 0, 0, 1, -1, 0, 0, 0, 0, 0, -1, 1, 0, 0, 0]
#media = 0
#Solucoes :  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#media = 0
#Solucoes :  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#media = 0
#Solucoes :  [1, 0, 0, 0, 0, 0, 0, 0, 0]
#media = 0
#Solucoes :  [1, 0, 0, 0, 0, 0, 0, 0]
#...
#media = 0
#Solucoes :  [1, 0, 0, 0]


"""
matriz_original=[
#  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
   3,  0,  0,  0,  0,  0,  0,  0,  0, -1,  0,  0,  0, -1, -1,
   0,  3, -1,  0,  0,  0, -1,  0,  0,  0,  0,  0,  0, -1,  0,
   0, -1,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  3, -1,  0, -1,  0, -1,  0,  0,  0,  0,  0,  0,
   0,  0,  0, -1,  4, -1,  0, -1,  0,  0,  0, -1,  0,  0,  0,
```

```
   0,  0,  0,  0, -1,  6, -1, -1,  0, -1,  0,  0,  0, -1, -1,
   0, -1,  0, -1,  0, -1,  7, -1,  0,  0, -1, -1,  0, -1,  0,
   0,  0,  0,  0, -1, -1, -1,  5,  0,  0,  0,  0, -1,  0, -1,
   0,  0,  0, -1,  0,  0,  0,  0,  5, -1, -1, -1, -1,  0,  0,
  -1,  0,  0,  0,  0, -1,  0,  0, -1,  6, -1,  0,  0, -1, -1,
   0,  0,  0,  0,  0,  0, -1,  0, -1, -1,  5, -1,  0, -1,  0,
   0,  0,  0,  0, -1,  0, -1,  0, -1,  0, -1,  6, -1, -1,  0,
   0,  0,  0,  0,  0,  0,  0, -1, -1,  0,  0, -1,  4,  0, -1,
  -1, -1,  0,  0,  0, -1, -1,  0,  0, -1, -1, -1,  0,  8, -1,
  -1,  0,  0,  0,  0, -1,  0, -1,  0, -1,  0,  0, -1, -1,  6]
"""

#AutoVector =  [[1][1][1][1][1][1][1][1][1][1][1][1][1][1][1]]
#o vector indicador: [3.87298334620742, 3.87298334620742,\
 3.87298334620742, 3.87298334620742, 3.87298334620742,   \
 3.87298334620742, 3.87298334620742, 3.87298334620742,   \
 3.87298334620742, 3.87298334620742, 3.87298334620742,   \
 3.87298334620742, 3.87298334620742, 3.87298334620742,   \
 3.87298334620742]
#media = 3.87298334620742
#Solucoes :  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]


"""
matriz_original=[
   3,  0,  0,  0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  1,  1,
   0,  3,  1,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  1,  0,
   0,  1,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  3,  1,  0,  1,  0,  1,  0,  0,  0,  0,  0,  0,
   0,  0,  0,  1,  4,  1,  0,  1,  0,  0,  0,  1,  0,  0,  0,
   0,  0,  0,  0,  1,  6,  1,  1,  0,  1,  0,  0,  0,  1,  1,
   0,  1,  0,  1,  0,  1,  7,  1,  0,  0,  1,  1,  0,  1,  0,
   0,  0,  0,  0,  1,  1,  1,  5,  0,  0,  0,  0,  1,  0,  1,
   0,  0,  0,  1,  0,  0,  0,  0,  5,  1,  1,  1,  1,  0,  0,
   1,  0,  0,  0,  0,  1,  0,  0,  1,  6,  1,  0,  0,  1,  1,
   0,  0,  0,  0,  0,  0,  1,  0,  1,  1,  5,  1,  0,  1,  0,
   0,  0,  0,  0,  1,  0,  1,  0,  1,  0,  1,  6,  1,  1,  0,
   0,  0,  0,  0,  0,  0,  0,  1,  1,  0,  0,  1,  4,  0,  1,
   1,  1,  0,  0,  0,  1,  1,  0,  0,  1,  1,  1,  0,  8,  1,
   1,  0,  0,  0,  0,  1,  0,  1,  0,  1,  0,  0,  1,  1,  6]
"""
#media = 0
#Solucoes :  [-1, 0, 0, -1, -1, 1, 0, -1, 1, -1, 0, 0, 1, 0, 0]
#media = 0
#Solucoes :  [1, 0, 0, 0, 0]

#calcular a mediana
def calcular_mediana(x):

    vm=sorted(x)
    nl=len(vm)

    if nl%2 == 0 :
        media = (vm[nl/2-1]+vm[(nl+2)/2-1])/2
    else:
        media = vm[(1+nl)/2]
    return media

def cont_um(v,n):
    cont=0
    for i in range(0,n):
        if(v[i]==1):
            cont+=1
    return cont
```

```
def cont_menosum(v,n):
    cont=0
    for i in range(0,n):
        if(v[i]==-1):
            cont+=1
    return cont

def cont_zero(v,n):
    cont=0
    for i in range(0,n):
        if(v[i]==0):
            cont+=1
    return cont

def geraNovoMatriz(mat,tamanho,conj,mapa):
    novomatriz=[0]*tamanho*tamanho
    n_novomatriz=len(conj)
    x=0
    nlinha=0

    for i in conj:
        cont=0
        cont_linha=0
        for k in range(0,len(mapa)):
            if mapa[k]==i: break
        pos_i = k
        for j in conj:
            for k in range(0,len(mapa)):
                if mapa[k]==j: break
            pos_j = k

            if i!=j:

                ligar=mat[pos_i*len(mapa)+pos_j]

                if ligar==-1:
                    cont+=1
                    novomatriz[x+cont_linha]=ligar
                elif ligar == 0:
                    novomatriz[x+cont_linha]=ligar

            cont_linha+=1
        novomatriz[n_novomatriz*nlinha+nlinha]=cont
        x+=n_novomatriz
        nlinha+=1
    return novomatriz

def conj_menosum(v,mapa):
    a=[]
    n=len(v)
    for i in range(0,n):
        if v[i]==-1:
            a.append(mapa[i])
    return a

def conj_um(v,mapa):
    a=[]
    n=len(v)
    for i in range(0,n):
        if v[i]==1:
            a.append(mapa[i])
    return a

def conj_zero(v,mapa):
```

```
    a=[]
    n=len(v)
    for i in range(0,n):
        if v[i]==0:
            a.append(mapa[i])
    return a

def func_bissecao(matriz_original,n,mapa):
    a = sp.Matrix(n,n,matriz_original)
    autoValues = sp.Matrix(a.eigenvals().keys())
    #Devolve uma lista com: autovalores, multiplicidade e base
    #autoVector = [sp.Matrix(n,1,a.eigenvects()[0][2][0]),\
                   sp.Matrix(n,1,a.eigenvects()[1][2][0])]
    Informacao = a.eigenvects()
    autoVector = []
    for i in Informacao: # para cada autovalor
        for j in range(i[1]): # multiplicidade do autovalor
            autoVector.append(i[2][j])
    x=[0]*n
    #calcula vector indicador
    for i in range(0,n):
        x[i]+=(math.sqrt(n)*autoVector[0][i])
    print "o vector indicador:",x
    v=[0]*n
    media=calcular_mediana(x)
    print "media =",media
    for i in range(0,n):
        if x[i]>media:
            v[i]=1
        if x[i]<media :
            v[i]=-1
    print "Solucoes : ",v
    cont1=cont_um(v,n)
    cont_1=cont_menosum(v,n)
    conta_zero=cont_zero(v,n)
    print cont1,cont_1
    if cont1<=3:
        for i in range(0,n):
            if v[i]==1:
                print "Grupo de um:",mapa[i]

    if cont_1<=3:
        for i in range(0,n):
            if v[i]==-1:
                print "Grupo de menos um:",mapa[i]
    if conta_zero<=3:
        for i in range(0,n):
            if v[i]==0:
                print "Grupo de zero",mapa[i]

    if conta_zero>3:
        conj=[0]*conta_zero
        conj=conj_zero(v,mapa)
        n_novomatriz=len(conj)
```

```
        novomatriz=[0]*conta_zero*conta_zero
        novomatriz=geraNovoMatriz(a.mat,conta_zero,conj,mapa)
        print_novomatriz(novomatriz)
        func_bissecao(novomatriz,n_novomatriz,conj)
    if cont1>3:
        conj=[0]*cont1
        conj=conj_um(v,mapa)
        n_novomatriz=len(conj)
        novomatriz=[0]*cont1*cont1
        novomatriz=geraNovoMatriz(a.mat,cont1,conj,mapa)
        print_novomatriz(novomatriz)
        func_bissecao(novomatriz,n_novomatriz,conj)
    if cont_1>3:
        conj=[0]*cont_1
        conj=conj_menosum(v,mapa)
        n_novomatriz=len(conj)
        novomatriz=[0]*cont_1*cont_1
        novomatriz=geraNovoMatriz(a.mat,cont_1,conj,mapa)
        print_novomatriz(novomatriz)
        func_bissecao(novomatriz,n_novomatriz,conj)

def print_novomatriz(novomatriz):
    l=int(math.sqrt(len(novomatriz)))
    i=0
    while i < len(novomatriz):
            for j in range(0,l):
                print '{0} '.format(novomatriz[i]),
                i+=1
            print "\n"


def func_arestas(matriz_original,n,matriz):
    a = sp.Matrix(n,n,matriz_original)
    autoValues = sp.Matrix(a.eigenvals().keys()) # Correto
    #Devolve uma lista com: autovalores, multiplicidade e base
    Informacao = a.eigenvects()
    autoVector = []
    for i in Informacao: # para cada autovalor
for j in range(i[1]): # multiplicidade do autovalor
            autoVector.append(i[2][j])
    x=[0]*n
    for i in range(0,n):
        x[i]+=(math.sqrt(n)*autoVector[0][i])
    v=[0]*n
    media=calcular_mediana(x)
    for i in range(0,n):
        if x[i]>media:
            v[i]+=1
        if x[i]<media :
            v[i]+=-1
    print "Solucoes : ",v
    soma=0
    for i in range(0,n):
```

```
        for j in range(0,n):
            if(matriz[i][j]==1):
                soma+=(v[i]-v[j])**2
    f=soma/n
    print "O numero de arestas que existe entre os conjuntos e:",f

mapa=[0]*(n)
for i in range(0,n):
    mapa[i]=i+1

func_bissecao(matriz_original,n,mapa)
#func_arestas(matriz_original,n,matriz)
```

Apêndice B

Resultados de Execução

B.0.1 Grafo 1

```
matriz_original = [
   3, -1, -1, -1,
  -1,  1,  0,  0,
  -1,  0,  1,  0,
  -1,  0,  0,  1]
o vector indicador: [0, -2.00000000000000, 2.00000000000000, 0]
media = 0
Solucoes :  [0, -1, 1, 0]
```

B.0.2 Grafo 2

```
matriz_original = [
   5, -1, -1, -1, -1, -1,
  -1,  3, -1, -1,  0,  0,
  -1, -1,  4,  0, -1, -1,
  -1, -1,  0,  2,  0,  0,
  -1,  0, -1,  0,  2,  0,
  -1,  0, -1,  0,  0,  2 ]
Não foi possivel testar, eata entra um ciclo finito.
```

B.0.3 Grafo 3

```
matriz_original=[
#  1   2   3   4   5   6   7   8   9  10  11  12  13  14
 -13,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
   1, -9,  1,  1,  1,  1,  1,  1,  1,  0,  0,  1,  0,  0,
   1,  1,-10,  1,  1,  1,  1,  1,  1,  0,  0,  1,  0,  1,
   1,  1,  1, -8,  1,  1,  0,  1,  1,  0,  0,  1,  0,  0,
```

```
   1,  1,  1,  1, -9,  1,  1,  1,  1,  0,  0,  1,  0,  0,
   1,  1,  1,  1,  1, -9,  0,  1,  1,  0,  0,  1,  0,  1,
   1,  1,  1,  0,  1,  0, -6,  0,  1,  0,  0,  1,  0,  0,
   1,  1,  1,  1,  1,  1,  0,-10,  1,  1,  0,  1,  0,  1,
   1,  1,  1,  1,  1,  1,  1,  1, -9,  0,  0,  1,  0,  0,
   1,  0,  0,  0,  0,  0,  0,  1,  0, -5,  1,  1,  1,  0,
   1,  0,  0,  0,  0,  0,  0,  0,  0,  1, -3,  1,  0,  0,
   1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,-12,  0,  1,
   1,  0,  0,  0,  0,  0,  0,  0,  0,  1,  0,  0, -2,  0,
   1,  0,  1,  0,  0,  1,  0,  1,  0,  0,  0,  1,  0, -5]
o vector indicador: [-48.6415460280612, 3.74165738677394, \
3.74165738677394, 3.74165738677394, 3.74165738677394,     \
3.74165738677394, 3.74165738677394, 3.74165738677394,     \
3.74165738677394, 3.74165738677394, 3.74165738677394,     \
3.74165738677394, 3.74165738677394, 3.74165738677394]
media = 3.74165738677394
Solucoes :  [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

o vector indicador: [3.60555127546399, 0, 0, 0, 0, 0, 0,  \
                     0, 0, 0, 0, 0, 0]
media = 0
Solucoes :  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

o vector indicador: [3.46410161513775, 0, 0, 0, 0, 0, 0,  \
                     0, 0, 0, 0, 0, 0]
media = 0
Solucoes :  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

o vector indicador: [3.31662479035540, 0, 0, 0, 0, 0, 0,  \
                     0, 0, 0, 0, 0, 0]
media = 0
Solucoes :  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

o vector indicador: [3.16227766016838, 0, 0, 0, 0, 0, 0,  \
                     0, 0, 0, 0, 0, 0]
media = 0
Solucoes :  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

o vector indicador: [3.00000000000000, 0, 0, 0, 0, 0, 0,  \
                     0, 0, 0, 0, 0, 0]
media = 0
Solucoes :  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

o vector indicador: [2.82842712474619, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
media = 0
Solucoes :  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

o vector indicador: [2.64575131106459, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
media = 0
Solucoes :  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

o vector indicador: [2.44948974278318, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
media = 0
Solucoes :  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

o vector indicador: [2.23606797749979, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
media = 0
Solucoes :  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
o vector indicador: [2.00000000000000, 0, 0, 0]
media = 0
Solucoes :  [1, 0, 0, 0]
```

B.0.4 Grafo 4

```
matriz_original = [
   3, -1, -1, -1,
  -1,  2,  0, -1,
  -1,  0,  1,  0,
  -1, -1,  0,  2]
o vector indicador: [0, 2.00000000000000, -4.00000000000000, \
                     2.00000000000000]
media = 1.00000000000000
Solucoes :  [-1, 1, -1, 1]
```

B.0.5 Grafo 5

```
matriz_original=[
   4, -1, -1, -1, -1,
  -1,  3,  0, -1, -1,
  -1,  0,  3, -1, -1,
  -1, -1, -1,  3,  0,
  -1, -1, -1,  0,  3]
o vector indicador: [0, -2.23606797749979, 2.23606797749979, \
                     0, 0]
media = 0
Solucoes :  [0, -1, 1, 0, 0]
```

B.0.6 Grafo 6

```
matriz_original=[
   3, -1, -1, -1,
  -1,  2,  0, -1,
  -1,  0,  2, -1,
  -1, -1, -1,  3 ]
o vector indicador: [0, -2.00000000000000, 2.00000000000000, 0]
media = 0
Solucoes :  [0, -1, 1, 0]
```

B.0.7 Grafo 7

```
matriz_original = [
   4, -1, -1, -1, -1,
  -1,  3, -1, -1,  0,
```

```
  -1, -1,  4, -1, -1,
  -1, -1, -1,  4, -1,
  -1,  0, -1, -1,  3]
o vector indicador: [0, -2.23606797749979, 0, 0, 2.23606797749979]
media = 0
Solucoes :  [0, -1, 0, 0, 1]
```

B.0.8 Grafo 8

```
matriz_original=[
   4, -1, -1, -1, -1,
  -1,  3, -1,  0, -1,
  -1, -1,  4, -1, -1,
  -1,  0, -1,  3, -1,
  -1, -1, -1, -1,  4]
o vector indicador: [0, -2.23606797749979, 0, 2.23606797749979, 0]
media = 0
Solucoes :  [0, -1, 0, 1, 0]
```

B.0.9 Grafo 9

```
matriz_original=[
   8, -1, -1, -1, -1, -1, -1, -1, -1,
  -1,  8, -1, -1, -1, -1, -1, -1, -1,
  -1, -1,  8, -1, -1, -1, -1, -1, -1,
  -1, -1, -1,  7, -1, -1,  0, -1, -1,
  -1, -1, -1, -1,  8, -1, -1, -1, -1,
  -1, -1, -1, -1, -1,  7,  0, -1, -1,
  -1, -1, -1,  0, -1,  0,  5,  0, -1,
  -1, -1, -1, -1, -1, -1,  0,  7, -1,
  -1, -1, -1, -1, -1, -1, -1, -1,  8]
o vector indicador: [-3.00000000000000, 3.00000000000000, 0, \
                     0, 0, 0, 0, 0, 0]
media = 0
Solucoes :  [-1, 1, 0, 0, 0, 0, 0, 0, 0]
o vector indicador: [-2.64575131106459, 0, 2.64575131106459, \
                     0, 0, 0, 0]
media = 0
Solucoes :  [-1, 0, 1, 0, 0, 0, 0]
o vector indicador: [2.23606797749979, 2.23606797749979,     \
                    -6.70820393249937, 2.23606797749979, 0]
media = 2.23606797749979
Solucoes :  [0, 0, -1, 0, -1]
```

B.0.10 Grafo 10

```
matriz_original=[
   3, -1, -1, -1,
  -1,  3, -1, -1,
  -1, -1,  3, -1,
  -1, -1, -1,  3]
o vector indicador: [-2.00000000000000, 2.00000000000000, 0, 0]
media = 0
Solucoes :  [-1, 1, 0, 0]
```

Referências

[1] Sanjeev Arora, Satish Rao, and Umesh Vazirani. Geometry, flows, and graph-partitioning algorithms. *Commun. ACM*, 51(10):96–105, October 2008.

[2] C. Bessiere. Arc-consistency and Arc-consistency Again. *Artificial Intelligence*, 65:179–190, 1994.

[3] C. Bessiere and E. C. Freuder. Using Constraint Metaknowledge to Reduce Arc-consistency Computation. *Artificial Intelligence*, 107(1):125–148, January 1999.

[4] Per-Olof Fjällström. Algorithms for graph partitioning: a survey. *Linköping Electronic Articles in Computer and Information Science*, 3(10), 1998.

[5] R. Gomory and T. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

[6] B. Hendrickson and R. Leland. Multidimensional Spectral Load Balancing. In *Proceeding of 6th SIAM Conf. Parallel Proc. Sci. Comput., Sandia National Laboratories*, pages 953–961, January 1993.

[7] P. V. Hentenryck, Y. Deville, and C. Teng. A Generic Arc-consistency Algorithm and its Specializations. *Artificial Intelligence*, 57:291–321, 1992.

[8] Karin Hogstedt, Doug Kimelman, V T Rajan, Tova Roth, and Mark Wegman. Graph cutting algorithms for distributed applications partitioning. *SIGMETRICS Perform. Eval. Rev.*, 28(4):27–29, March 2001.

[9] B.W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell Systems Technical Journal*, 49(2), 1970.

[10] V. Kumar. Algorithms for Constraint Satisfaction Problems: A Survey. *Artificial Intelligence Magazine*, 13(1):32–44, 1992.

[11] A. K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[12] K. Marriot and P. J. Stuckey. *Programming with constraints: An Introduction*. MIT Press, 1998.

[13] R. Mohr and T. C. Henderson. Arc and Path Consistency Revisited. *Artificial Intelligence*, 28(2):225–233, 1986.

[14] D. Padua. *Encyclopedia of Parallel Computing*, volume 4 of *Springer reference*. Springer, 2011.

[15] M. R. Pereira. Paralelização de Algoritmos de Consistência de Arcos em um Cluster de PCś. Dissertação de mestrado, COPPE - Engenharia de Sistemas e Computação - Universidade Federal do Rio de Janeiro, Rio de Janeiro, Agosto 2001.

[16] Marluce Rodrigues Pereira. *Particionamento Automático de Restrições*. PhD thesis, Department of Systems and Computer Engineering, Federal University of Rio de Janeiro, Março 2006.

[17] Alex Pothen, Horst D. Simon, and Kan-Pu Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, May 1990.

[18] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, Maio 2003.

[19] A. Ruiz-Andino, L. Araujo, F. Sáenz, and J. Ruz. Parallel Execution Models for Constraint Programming over Finite Domains. In Gopalan Nadathur, editor, *Principles and Practice of Declarative Programming, Intl. Conf. PPDP, Paris, France*, volume 1702 of *Lecture Notes in Computer Science*, pages 134–151. Springer, September 29–October 1 1999.

[20] Christian Schulz. *High Quality Graph Partitioning*. PhD thesis, Faculty of Computer Science, Karlsruhe Institute of Technology, Jul 2013.

[21] H.D. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2(2–3):135 – 148, 1991. ¡ce:title¿Parallel Methods on Large-scale Structural Analysis and Physics Applications¡/ce:title¿.

[22] G. R. Spendley, G. R. Hext, and F. R. Himsworth. Sequential Application of Simplex Designs in Optimization and Evolutionary Operation. *Technometrics*, 4:441–461, 1962.

[23] Fábio Tavares. Particionamento e execução paralela de redes de restrições. Technical report, University of Porto, Setembro 2008.