DESCONTAMINAÇÃO DISTRIBUÍDA DE GRAFOS

Vanessa Carla Felipe Gonçalves

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Civil, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador(es): Felipe Maia Galvão França

Priscila Machado Vieira Lima

Rio de Janeiro

Junho de 2011

DESCONTAMINAÇÃO DISTRIBUÍDA DE GRAFOS

Vanessa Carla Felipe Gonçalves

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

_____
Prof. Felipe Maia Galvão França, Ph.D.

_____
Profª. Priscila Machado Vieira Lima, Ph.D.

_____
Prof. Valmir Carneiro Barbosa, Ph.D.

_____
Profª Luciana Salete Buriol, Dra.

RIO DE JANEIRO, RJ – BRASIL

JUNHO DE 2011

Gonçalves, Vanessa Carla Felipe

Descontaminação Distribuida de Grafos/ Vanessa Carla Felipe Gonçalves. – Rio de Janeiro: UFRJ/COPPE, 2011.

XI, 72 p.: il.; 29,7 cm.

Orientador(es): Felipe Maia Galvão França

Priscila Machado Vieira Lima

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2011.

Referências Bibliográficas: p. 61-67.

1. Descontaminação de Grafos. 2. Escalonamento por Reversão de Arestas. 3. Algoritmos em grafos. I. França, Felipe Maia Galvão *et al*. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Aos meus pais e irmão, família e amigos.

AGRADECIMENTOS

Gostaria de agradecer em primeiro lugar ao professor Felipe Maia Galvão França, pela paciência e por ter sido meu guia durante esta jornada. Seu empenho em fazer com que as coisas dessem certo me mostrou que chegar nessa parte de desfecho do caminho é uma grande conquista.

Agradeço também a Professora Priscila Machado Vieira Lima, que sempre teve a presença de espírito de me desacelerar quando as coisas saíam de seu caminho, me colocando nos eixos e na direção certa. Além de apresentar um novo horizonte para este trabalho, tornando-o ainda mais interessante para mim.

Ao professor Valmir Barbosa por ministrar as aulas da disciplina Algoritmos Distribuídos que derem início ao interesse pelo assunto.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo suporte financeiro a este trabalho.

Aos amigos da vida acadêmica e que também se tornaram amigos para toda uma vida: Gizelle Gaspar (minha eterna roomate), Isabella Casasola, Priscila Abonante, Fernanda Couto, Carla Rodrigues (paciente, amiga, incentivadora,...), Caio Souza (o menino das caronas), Luiz Tomaz, Rita Berardi e muitos outros. Sem eles eu não teria alcançado este patamar. Cada um de vocês é dono de uma parte da minha conquista.

Ao meu revisor e incentivador Robson Domingos da Rocha, que nesta fase final foi de suma importância e que esteve presente durante todo o tempo. Além de termos trabalhado juntos e de ter absorvido muito do seu conhecimento técnico e profissional e da sua..... irreverência. Obrigada!

Aos amigos COPPETEQUIANOS: Camille Furtado (doida!), Daniel Oliveira (Dêniel – pronto para ajudar em qualquer circunstância), Amanda Mattos, Felipe Leite, Marcio Duran, Matheus Wildemberg, Patrícia Curvelo e os tantos outros com os quais trabalhei direta ou indiretamente e aprendi muita coisa. Muitos deles continuam participando da minha vida e os considero como irmãos. Obrigada!

E em especial, agradeço a Deus por ter me dado a oportunidade de seguir esse caminho pelo qual me deparei com pessoas excepcionais e que farão parte da minha vida para sempre.

Resumo da Dissertação apresentada a COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M. Sc.).

DESCONTAMINAÇÃO DISTRIBUÍDA DE GRAFOS

Vanessa Carla Felipe Gonçalves

Junho/2011

Orientadore(s): Felipe Maia Galvão França
Priscila Machado Vieira Lima

Programa: Engenharia de Sistemas e Computação

Seja $G = (V,A)$ um grafo conexo considerado como inicialmente contaminado. O problema de descontaminação de G consiste em extinguir essa contaminação, considerando os critérios de recontaminação de cada situação apresentada, utilizando-se do menor número de guardas para realizar essa "limpeza". No caso mais estrito, um nó sadio é contaminado caso entre em contato com um vizinho contaminado; um nó sadio permanece sadio somente se todos os seus vizinhos forem sadios ou estiverem guardados. Na literatura, casos como a contrução de *Link Farms*, i.e., conjuntos de páginas com *links* inseridos por *webspammers* visando aumentar a visibilidade de uma página alvo *T*, configura um caso de contaminação de grafos. Desfazer tais *link farms*, ou seja, descontaminar o grafo *web* correspondente, requer o emprego de guardas, buscando minimizar tanto o número de agentes quanto o de saltos, onde estes últimos indicam o tempo de descontaminação. No trabalho pioneiro de Luccio & Pagli, o critério de recontaminação é o de ter 50% dos vizinhos contaminados, sendo que a topologia dos grafos que representam esses conjuntos de páginas é restrita a grafos circulantes. Este trabalho apresenta *Alg-D*, um algoritmo assíncrono e distribuído baseado na dinâmica de Escalonamento por Reversão de Arestas. Além de descontaminar grafos conexos de topologia arbitrária, *Alg-D* apresenta melhores resultados dos que os encontrados na literatura para o caso de grafos circulantes.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

DISTRIBUTED GRAPH DECONTAMINATION

Vanessa Carla Felipe Gonçalves

June/2011

Advisors: Felipe Maia Galvão França
Priscila Machado Vieira Lima

Department: Computer and Systems Engineering

Let $G = (N,E)$ be a contaminated connected graph. The decontamination problem consists of extinguishing the contamination of G, considering the recontamination *criterium* of each situation presented, while employing the smallest number of guards needed to accomplish this task. In the strictest case, a *clean* node is infected if it has a contaminated neighbor; a node remains clean if all its neighbours are either clean or guarded. In the literature, *link farms* are sets of pages having links inserted by webspammers in order to enhance the visibitlity of a target page T, and such constitutes a case of graph contamination. Undoing these *link farms*, i.e., decontaminating the corresponding web graph, requires the use of guards. In decontaminating a graph, it is desirable to minimise both the number of agents and the number of hops, where the latter determines the time taken for the decontamination. In the pioneer work by Luccio & Pagli, the recontamination criterium adopted means having 50% of the neighbours contaminated. Also, in that work graph topology is restricted to *circulant* graphs. This work presents *Alg-D*, an asynchronous distributed algorithm which is based on the *Sheduling by Edge Reversal* dynamics. Besides decontaminating arbitrary-topology connected graphs, *Alg-D* presents better results than the ones proposed for circulant graphs in related works.

SUMÁRIO

Índice de Figuras

1 Introdução

1.1. Motivação

A descontaminação de grafos ainda constitui um assunto pouco explorado e o aprofundamento nos estudos relacionados a este tipo de é de grande atualidade. O objetivo da descontaminação de grafos é extinguir qualquer problema que possa ser encarado como uma *contaminação* em um grafo conexo. Ademais, faz-se ainda necessário analisar soluções para descontaminação do ponto de vista da escalabilidade.

Em alguns trabalhos relacionados podemos encontrar métodos de descontaminação que utilizam inundação, onde a estrutura em questão é percorrida em largura *(Breadth-First*) como em (FLOCCHINI *et al*., 2005) ou que são projetados para uma dada topologia (LUCCIO *et al.*, 2007), mas que ainda deixam algumas lacunas a serem preenchidas. No caso da inundação, pode ser observado que um número excessivo de agentes móveis foi usado para descontaminar os grafos apresentados. Em (LUCCIO *et al.*, 2007) o embasamento da solução em uma topologia particular de grafo, associada a um sub-tipo do problema, pode não ser adequado para resolver o caso geral. O algoritmo distribuído para descontaminação de grafos que será mostrado neste trabalho é baseado na dinâmica *Escalonamento por Reversão de Arestas* (ERA) (GAFNI *et al.*, 1981, BARBOSA *et al.*, 1989, BARBOSA 1996, BARBOSA, 2000), um algoritmo distribuído para compartilhamento de recursos.

Outro ponto de atenção seria a Rede Mundial de Computadores – Internet -, que está sempre em constante modificação. Cada vez mais, as formas de ataques se intensificam e são realizadas de formas variadas. Desta forma, apenas ações paliativas são tomadas, tendo em vista que, a cada ataque, novas tecnologias e recursos são utilizados, impossibilitando uma ação imediata que resolva o problema de forma mais incisiva. Incidentes relacionados à segurança de informações valiosas disponíveis em redes (abertas ou não~~),~~ pessoais ou empresariais~~,~~) devem ser tratados objetivamente, para evitar que esses dados sejam adquiridos por terceiros e usados para fins danosos. Um exemplo recente, que exemplifica a extensão do problema a ser enfrentado, é o ataque realizado a uma rede de jogos online - a *PlayStation*® *Network* (PSN), que

possibilita que portadores do console *PlayStation*® da Sony™ interajam online. O ocorrido afetou cerca de 77 milhões de usuários e extraiu um quantidade expressiva de números de cartões de crédito dos usuários titulares de contas pagas, além de nome completo e endereço de usuários com variados perfis. Esse ataque causou o desligamento temporário da rede PSN e, conseqüentemente, acarretou em prejuízos para a mantenedora, pois a venda de jogos pela rede, provedora de uma boa parte dos lucros da empresa, foi interrompida. A invasão alarmou os assinantes da rede que tiveram o número de seus cartões de crédito seqüestrados (PSN, 2011). Uma rápida ação para contornar o problema da invasão teria evitado que as perdas monetárias e os danos à imagem da rede que sofreu o ataque fossem tão grandes. Uma abordagem para lidar com tal situação será explicitada neste trabalho.

1.2. Objetivos

O objetivo deste trabalho é apresentar um novo algoritmo que opera de forma assíncrona e distribuída para descontaminar grafos conexos que apresentem qualquer topologia. Após a introdução do método, é objetivo mostrar também que, em comparação com outros métodos previamente apresentados na literatura, o número de agentes móveis utilizados pode ser satisfatoriamente pequeno. Para tal, várias abordagens presentes na literatura serão exploradas e novos pontos de atenção serão introduzidos.

A generalização para qualquer estrutura é importante para que outros problemas com características que possam ser representadas por grafos conexos sejam reavaliados e inseridos no contexto da contaminação de grafos. Esse passo é necessário para que seja avaliada a aplicabilidade do método aqui proposto, limitando ou aumentando sua extensão a problemas encontrados em várias áreas e subáreas relacionadas à busca em grafos. Uma vez que o Escalonamento por Reversão de Arestas, algoritmo base para o procedimento adotado neste trabalho, tem se mostrado útil para resolver problemas em que há restrições de vizinhança e tem seu estado global influenciado por cada integrante do sistema em questão.

Pretendemos mostrar também que, ao tratar situações já conhecidas, mas ainda não reconhecidas como problemas de contaminação de grafos, novas linhas de investigação podem ser iniciadas. A partir desses novos estudos, o método aqui

proposto poderá ser aprimorado até que seja possível aplicar a descontaminação de grafos a problemas de tamanho real. Vale ressaltar ainda que, a detecção de situações que caracterizam uma contaminação não será abordada neste trabalho.

1.3. Organização dos capítulos

Para um melhor entendimento, a organização deste trabalho está composta da seguinte forma:

Capitulo 2: Este capítulo mostra os conceitos utilizados para o desenvolvimento da proposta aqui apresentada. Tais conceitos envolvem a definição da contaminação de grafos, da descontaminação de grafos, o Escalonamento por Reversão de Arestas e a geração de orientações acíclicas de forma distribuída.

Capitulo 3: Neste capítulo, o método de descontaminação de grafos assíncrono e distribuído é apresentado.

Capitulo 4: Este capítulo traz os resultados aferidos e a análise dos resultados obtidos.

Capitulo 5: Neste segmento, as conclusões e os trabalhos futuros são apresentados.

2 Conceitos Básicos e Conhecimentos Preliminares

Neste capítulo serão definidos alguns conceitos e apresentados alguns conhecimentos preliminares necessários para o entendimento do cenário no qual se insere o trabalho aqui proposto. Nas seções que seguem, o termo contaminação de grafos será introduzido e o Escalonamento por Reversão de Arestas, elucidado.

2.1. Métodos de busca em grafos e propriedades abordadas

Neste trabalho serão abordados temas bastante explorados na literatura relacionada a algoritmos em grafos. Métodos de busca foram desenvolvidos para atender a necessidade de extrair informações acerca de um determinado grafo; e a definição de algumas propriedades foi feita para que um agrupamento de cada tipo de grafo pudesse ser realizado. Nas seções a seguir, o método de busca em largura e a densidade serão definidos.

2.1.1 Densidade de um grafo e Busca em Largura

Densidade de um grafo

A densidade de um grafo G definida por $d(G)$ é uma propriedade que está relacionada à quantidade de arestas existentes no grafo. Com esta relação, é possível definir o grau de esparsidade de um grafo. Esta medida é definida pela fórmula abaixo para grafos não orientados:

$$d(\mathrm{G}) = \frac{2m}{n\,(n-1)} \qquad \textbf{1)}$$

Onde n é o número de vértices e m é o número de arestas. Esta medida caracteriza um grafo como denso ou esparso. Um grafo denso possui uma grande quantidade de arestas por nó, ao contrário do que se observa em um grafo esparso (SZWARCFITER, 1984).

Busca em largura

A busca em largura (B*readth-Fisrt Search* - BFS) é um algoritmo de busca em grafos e é realizada percorrendo-se o grafo com base na vizinhança de cada nó. Dado um grafo $G = (V,A)$ e um vértice inicial r, a busca em largura explora sistematicamente as arestas de G para descobrir cada vértice que é alcançável a partir de r. A distância para cada vértice alcançável a partir de r também é calculada (menor número de arestas). O algoritmo funciona em ambos os grafos dirigidos e não dirigidos. Os vértices alcançados pelo algoritmo são colocados em uma fila. O BFS produz uma "árvore de busca em largura", tendo r como raiz, que contém todos os vértices alcançáveis. Este algoritmo possui complexidade $O(m + n)$, onde n é o número de vértices e m é o número de arestas do grafo G no qual o método esta sendo aplicado.

2.2. Contaminação de Grafos

A contaminação de grafos é um termo que surgiu recentemente, mas a aplicação do termo “contaminação” como estado global de um determinado ambiente é um conceito largamente disseminado. Na literatura atual, vários problemas têm sido apontados como problemas relacionados a contaminação de grafos. Tal conclusão deriva do fato de que muitos deles possuem características que possam ser mapeadas em um grafo conexo. Ao definir um grafo conexo correspondente ao problema indicado, é possível utilizar-se de métodos já disponíveis de busca em grafos para solucionar os mesmos.

2.2.1 Problemas Relacionados

Como dito anteriormente, baseando-se na propriedade do grafo em questão, é possível detectar uma contaminação imposta a este grafo e mapeá-lo de acordo com suas especificações para que métodos de descontaminação sejam aplicados. Problemas como redes de computadores contaminadas, inserção de *links* “fantasmas” em páginas *web* e equipes de exploração em florestas ou prédios em chamas podem ser listados como problemas mapeáveis.

Equipes de exploração: Uma equipe de exploração movimentando-se em uma floresta pode ser um problema que se encaixa na definição de contaminação de grafos. Uma vez que as áreas de exploração possam ser definidas como áreas em que certas

restrições devam ser tomadas como regras para avançar no local a ser explorado, essas dependências podem ser transformadas em um grafo conexo passível, portanto, de descontaminação (KALRA *et al*., 2006). Além da simples movimentação e exploração, o combate a incêndios florestais também pode ser incluído no fator que torna o trabalho de equipes de exploração em um problema de contaminação a ser contornado. Este problema seria uma combinação da forma como o grafo em questão deve ser percorrido e a contaminação que deve ser extinta do mesmo, o fogo em si, fazendo com que a contaminação a ser combatida, neste caso, seja bem explícita.

Edifícios em chamas: Em um edifício em chamas, é necessário que haja um planejamento para evacuação do mesmo para evitar que pessoas sejam atingidas pelas chamas por estarem em locais de risco. Na literatura, abordagens na área de robótica (MURPHY, 2004, BRADSHAW, 1991) vêm sendo aplicadas com o intuito de diminuir o risco de falha humana e a perda de vidas de pessoas especializadas no combate ao fogo (QUINTIERE, 2006). Recentes acontecimentos reforçaram a necessidade de dispensar uma maior atenção a este problema, como o ataque ao *World Trade Center* (WTC), nos EUA, em setembro de 2001, no qual morreram cerca de 3000 pessoas (MURPHY, 2004).

Enxergando este problema como o de um grafo contaminado, teríamos uma relação de dependência entre as áreas atingidas e suas ligações. Essas dependências definem um grafo de restrições e, conseqüentemente, um grafo conexo no qual nós e arestas são explicitados. Com este mapeamento, seria possível acoplar ao comportamento de robôs posicionamento e movimentação corretos para realizar o combate ao incêndio.

Redes de computadores infectadas: Constitui um dos casos mais explícitos de contaminação de grafos. Em uma rede contaminada com um vírus, por exemplo, cada servidor nesta rede seria um nó do grafo e as arestas, os *links* entre estes servidores.

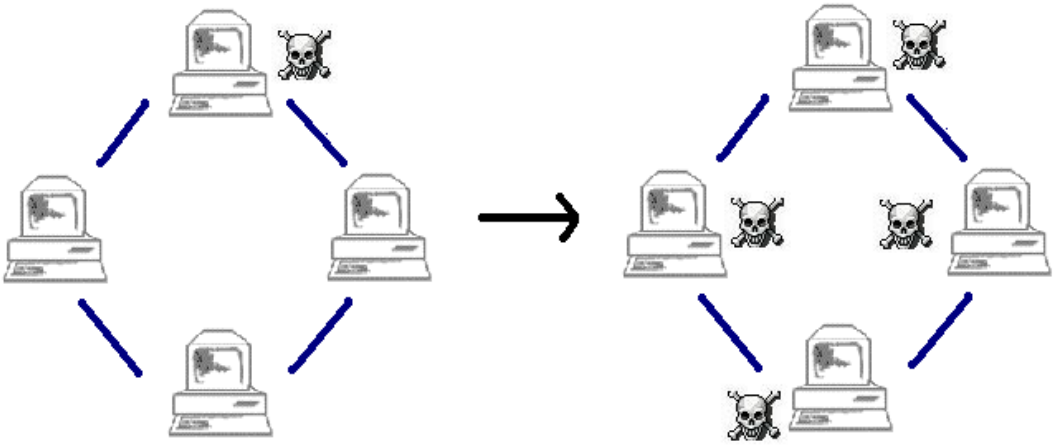

Figura 1 - Infecção em uma Rede de Computadores.

A Figura 1 elucida como uma rede de computadores pode tornar-se contaminada por um vírus, uma vez que todos os computadores possuem ligações diretas para um ou mais computadores da mesma. Para evitar que dados de caráter sigiloso ou dados pessoais sejam expostos, essa rede deve ser descontaminada. Hoje, o procedimento mais usado é a instalação de softwares em cada máquina que monitoram o aparecimento de programas não-amigáveis na máquina. Cada um então promove uma varredura local, e destrói o programa intruso. Este procedimento se torna desgastante quando há um vírus ainda não reconhecido pelo *software* utilizado, o que acarreta em uma atualização em massa dos softwares em cada computador na rede e, em até a desconexão da mesma para reparos. Uma vez que seja utilizado um método que possa trabalhar apenas a partir de certos computadores na rede e percorrer *link* após *link* da rede limpando os computadores infectados, o trabalho de limpeza de uma rede qualquer pode ser menos oneroso (BLIN, 2006, LUCCIO *et al*., 2006).

Link Farms: As *Link Farms* ou "emaranhado de *links*" são casos que se encaixam em uma generalização da contaminação de grafos. *Link Farms*, ou apenas *Farms*, são construídas a partir de links, para uma certa página alvo *T*, adicionados a páginas *web*. Além deste *link*, outros *links* entre as páginas escolhidas para popular a *Farm* são criados para que, uma vez em uma das páginas do conjunto, o usuário seja levado à página alvo *T*. O objetivo deste "ataque" é aumentar a visibilidade desta página alvo.

A contaminação é o aumento "ilegal" da visibilidade da página, o que afeta medidas como o *rankinG* de páginas *web* (*Page Rank*). Isto interfere em mecanismos

de busca, listando páginas que podem não estar relacionadas ao assunto que se deseja pesquisar. Os *links* para a página alvo em si, também são considerados uma contaminação, uma vez que, ao retirar um desses *links* para a página alvo, outras páginas podem reconstruí-lo, impondo condições de contaminação entre os vizinhos desta *Farm* (LUCCIO *et al.*, 2007).

2.3. Descontaminação de Grafos

A descontaminação de grafos é a metodologia desenvolvida para ser aplicada em um grafo *G* contaminado com o intuito de alterar o estado deste grafo para limpo ou livre de contaminação utilizando-se o menor número possível de *agentes móveis* (MOSCARINI *et al.*, 1998). Estes agentes são entidades autônomas desenvolvidas para viajar pelas conexões (*links* físicos – cabos – ou abstratos) da rede (ou grafo) que devem interferir carregando em seu "núcleo" dispositivos necessários para atenderem ao propósito ao qual estão designados. A aplicação destes agentes, inicialmente estava direcionada aos problemas geralmente encontrados em redes de computadores, pois a execução local do agente em cada ponto da rede (nó) faz com que sua utilização traga benefícios como: redução do tráfego da rede, execução assíncrona e anônima e encapsulamento de protocolos, entre outros (LANGE *et al.*, 1999). A partir dessa abordagem inicial, a detecção e captura de intrusos em redes foi o próximo passo a ser dado, dando início ao conceito de descontaminação de redes (ASAKA *et al.*,1999, BARRIERE *et al.*, 2002) e, em seguida, de grafos (FLOCCINI, 2005, FLOCCINI, 2007).

O objetivo dos agentes móveis, no caso da descontaminação de grafos, é percorrer o mesmo, replicando-se para atacar a contaminação presente em cada nó. O procedimento a ser realizado para efetuar a descontaminação é acoplado ao comportamento de cada agente, fazendo com que, ao implantar-se em um nó, o mesmo destrua o que torna aquele nó contaminado. Uma vez destruída a contaminação, o agente deve verificar a possibilidade de mover-se ou não para o próximo vizinho contaminado, de acordo com o critério de re-contaminação a ser levado em consideração. Caso não haja a possibilidade de mover-se do nó que o esteja hospedando pelo perigo de re-contaminação, o agente deve replicar-se e enviar as cópias para seus vizinhos. A regra com que este envio é feito será dada pelo algoritmo

proposto em seção subseqüente, que determina quais nós receberão agentes e em que momento, no decorrer da execução do procedimento de descontaminação.

Atualmente, poucos métodos relacionados diretamente ao problema de descontaminação de grafos podem ser encontrados (FLOCCINI, 2005, FLOCCINI, 2007, LUCCIO *et al.*, 2007). Para alguns casos, como grafos em larga escala, metodologias que não levem em conta o tamanho do grafo a ser processado podem não ser eficazes em atender ao propósito de utilizar o menor número, ou perto disso, de recursos necessários para descontaminar o grafo.

2.3.1 Critérios para o Número de Agentes Utilizados

Ao descontaminar um grafo é necessário se ater ao fato de que, dependendo dos critérios de contaminação e da estrutura do grafo, o número de agentes necessários seja uma medida a ser observada cuidadosamente. No caso de um critério mais estrito, o número de agentes a serem utilizados será maior do que em um caso com menos restrições para que um nó seja considerado limpo. A seguir, os critérios mais comumente utilizados serão apresentados.

No caso mais comum de descontaminação, o número de agentes utilizados para descontaminar um grafo deve ser minimizado. Dada a prova de NP-completude deste problema de minimização de agentes (MOSCARINI *et al.*, 1998), é justificável utilizar heurísticas que garantam que as características iniciais para alcançar números próximos ao valor ótimo sejam utilizadas. Em outros casos, o que deve ser levado em conta seria o estado global do grafo, não importando o número de agentes utilizados para que este estado seja alcançado (uma rede mantida para auxiliar um serviço essencial, por exemplo).

A minimização no número de agentes a serem utilizados na descontaminação está diretamente relacionada à forma como os agentes serão inicialmente implantados no grafo e como os mesmos irão moverem-se entre os nós para realizarem a descontaminação. Métodos de busca em grafos, além de somente os agentes, devem ser combinados para que a descontaminação tenha êxito, pois todos os nós devem ser alcançados para que o estado global do grafo possa ser analisado. Nos casos em que apenas a vizinhança imediata é utilizada como parâmetro para disseminação dos

agentes, é possível ver que há um uso excessivo dos mesmos no processo de descontaminação (FLOCCINI, 2005). Em FLOCCINI (2007), pode-se perceber que a modificação de uma visão local para uma "fotografia" mais abrangente do grafo resultou em uma diminuição no número de agentes envolvidos no processo.

Cliques Maximais vs. Número de Agentes Utilizados

Uma Clique Maximal é um subconjunto de vértices e arestas de um grafo *G*, conhecido como subgrafo de *G*, completamente conectados entre si por arestas de tal modo que não existe mais nenhum outro vértice do grafo que esteja conectado a todos os vértices do subconjunto. Neste caso, este subconjunto é um grafo completo (PROTTI, 1997). Na Figura 2, são exibidos grafos completos K_n com $1 \leq n \leq 5$, onde *n* é o número de nós do grafo. A partir da definição de contaminação, que é baseada na vizinhança de um nó, é possível perceber que esta estrutura seria a de maior interferência no estado global de um nó tendo em vista que cada um dos vizinhos nesta configuração está diretamente conectado a ele.

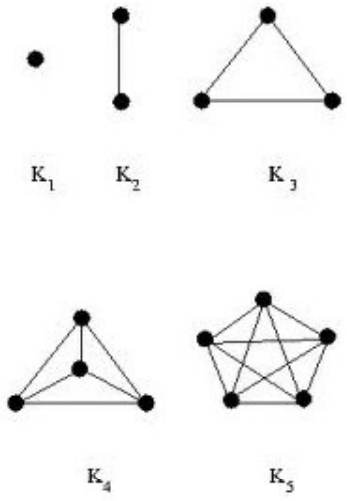

Figura 2 - Grafos K_n completos.

Como anteriormente citado (Seção 3.2.1), no caso mais estrito, uma vez que um nó limpo e não guardado é exposto a um vizinho contaminado, ele pode ser "atacado" por este vizinho e tornar-se contaminado novamente. Em um grafo completo, no caso da descontaminação, como todos os nós estão ligados entre si, existe a necessidade de

colocar um AM em cada nó pertencente a este clique, evitando assim que um nó presente no mesmo seja re-contaminado. Podemos concluir então, que a melhor solução para iniciar o processo de descontaminação, seria separar o grafo em cliques maximais e orientá-lo de tal forma que cada clique maximal "recebesse" um AM na rodada inicial, a partir deste momento, saberíamos quantos agentes seriam necessários para descontaminar o grafo por completo.

Generalizações

Como supracitado, outros casos podem priorizar outros aspectos da descontaminação como a minimização do tempo gasto para descontaminar o grafo, ou ainda, a constante manutenção do "sistema" no qual a descontaminação está sendo aplicada (tempo para descontaminação e o número agentes usados na descontaminação são contabilizados apenas para o acompanhamento dos recursos utilizados). Em alguns problemas relacionados, pode-se perceber que a minimização do quantitativo do número de agentes não é somente percebida em comparações diretas com o número de nós no grafo e suas arestas. Na abordagem das *Link Farms*, por exemplo, em que o número de vizinhos contaminados tem influência sobre a decisão de utilizar ou não mais agentes, pode ser encontrado um relaxamento quanto a medida do número de agentes utilizados. Deste modo, a minimização do número de agentes está baseada não só no número de nós do grafo, mas também em suas conexões e na forma com que a "contaminação" se comporta. Em outros casos, como os de serviços essenciais, o que precisa ser minimizado é o tempo em que a descontaminação é realizada, não importando (ou com uma menor importância), o número de agentes utilizados na descontaminação.

2.3.2 Critérios de Contaminação

Para realizar a descontaminação, o critério de contaminação deve ser levado em conta. Este critério indica de que forma um nó não protegido pode ser contaminado por seus vizinhos. Este ataque a um nó não protegido pode ser determinado por um número específico de nós em relação ao total de vizinhos, ou outros tipos de referências podem ser levadas em conta.

- **Critério Original**: Neste critério, o número de vizinhos contaminados que pode contaminar um vizinho desprotegido é igual a um (1), ou seja, basta que um vizinho de um nó não protegido esteja contaminado para que este nó, que estava antes limpo, seja contaminado novamente. Como nos vários casos citados na Seção 2.2.1, as restrições em cada abordagem fazem referência ao tipo de problema a ser solucionado. No entanto, a medida a ser tomada para que as restrições sejam atendidas é basicamente a mesma: evitar que um nó seja re-contaminado.

- **Generalizações**: Ao relaxar o critério de contaminação, outros problemas podem ser mapeados mais facilmente. A partir desse mapeamento, pode-se aferir as características da abordagem em questão com a finalidade de definir como a contaminação da mesma pode ser extinta. Em LUCCIO *et al.* (2007), por exemplo, o critério de contaminação utilizado foi baseado na quantidade de vizinhos de cada nó de acordo com o grafo em questão. Após a descontaminação, para que um nó não protegido fosse contaminado novamente, um mínimo de 50% de seus vizinhos deveria ser observado, caso contrário, o nó era considerado limpo. Este critério pode variar de acordo com a necessidade e o grau das restrições a serem atendidas.

2.4. Escalonamento por Reversão de Arestas (ERA)

O Escalonamento por Reversão de Arestas (ERA) é um algoritmo baseado na vizinhança dos vértices de um grafo. Uma vez definida uma orientação acíclica em um grafo G e conseqüentemente nós sumidouros, o ERA pode ser aplicado para se obter um conjunto de iterações em que os nós sumidouros se alternem até que uma repetição ocorra, caracterizando uma execução com sucesso do algoritmo (BARBOSA *et al.*, 1989, BARBOSA, 1996).

As operações realizadas pelo ERA tem como cerne basicamente inverter as arestas de nós sumidouros para que se possa obter a próxima orientação. Este método vem sido largamente utilizado com o objetivo de solucionar problemas relacionados ao compartilhamento de recursos, tendo em vista que um conjunto de nós com restrições de vizinhança possa operar sem que ocorra *deadlock* (bloqueio perpétuo; e.g., o

processo A, em operação, espera um recurso que o processo B está utilizando; e o processo B, também em operação, espera por um recurso que o processo A está utilizando, fazendo com que os processo A e B nunca recebam os recursos requisitados para que operem, liberando os recursos aos quais obtêm prioridade) ou *starvation* (inanição; o processo nunca obtém prioridade sobre todos os recursos necessários para que entre em operação) (BARBOSA *et al.*, 1989, BARBOSA, 1996).

2.4.1 Definição

O ERA é implementado através do envio de mensagens entre vizinhos imediatos para indicar a reversão da aresta que os liga. Em qualquer momento na evolução do algoritmo, um nó está esperando para operar ou está em operação: Um nó inoperante espera por um recurso a ser liberado; um nó que está operando utiliza os recursos que compartilha com outros nós. Ao passo que todos os nós sumidouros estão operando, os nós restantes permanecem inoperantes, garantindo a exclusão mútua para cada acesso realizado aos recursos compartilhados. Após operar, um nó reverte suas arestas incidentes, liberando os recursos utilizados. No próximo passo, os nós que obtiverem prioridade sobre os recursos irão operar (sumidouros). Então, o escalonamento pode ser considerado como a evolução no tempo de uma sequência de orientações acíclicas.

O principal objetivo no Escalonamento por Reversão de Arestas é garantir que todos os nós envolvidos no processo operem ao menos uma vez. Assim sendo, o ERA pode ser considerado como uma aplicação infinita de uma função $\omega = g(\omega)$, que denota o algoritmo guloso representado pelo ERA em G, onde $g(\omega)$ é a orientação acíclica gerada pela reversão das arestas dos sumidouros da orientação acíclica ω. Assim, na seqüência $\omega_1, \omega_2, \omega_3, \ldots$, temos $\omega_2 = g(\omega_1)$ (ARANTES Jr, 2006). Assumindo que G é finito, é fácil ver que eventualmente um repetição será encontrada (BARBOSA *et al.*, 1989, BARBOSA, 1996). Esta repetição é um período p construído por uma série de reversões durante a execução do algoritmo de escalonamento. Este período é determinado por um subconjunto contido em $\{\omega_0, \omega_1, \omega_2, \ldots, \omega_n\}$, um conjunto de orientações acíclicas obtidos pelo ERA, aonde $\omega_0 \neq \omega_1 \neq \ldots \neq \omega_{n-1} \neq \omega_n$ e ao reverter as arestas de ω_n nós obtemos ω_i, já observada em algum momento no processo. Então p é

$\{\omega_i, .., \omega_n\}$, $0 \leq i < n$, e o tamanho do período é $|p|$. Neste período p, o número de vezes que um nó torna-se sumidouro é igual para todos os nós.

Para realizar o escalonamento por reversão de arestas em um determinado problema, é necessário que o mesmo seja transformado em um grafo direcionado acíclico *G*. Em um grafo representando uma cadeia de compartilhamento de recursos, por exemplo, cada nó representa um processo a ser escalonado, e cada aresta $a_{i,j}$, um recurso compartilhado entre dois processos i e j. Desta forma, as restrições estão explicitadas e as dependências garantidas e, ao aplicar o escalonamento, cada processo irá operar quando obtiver prioridade sobre todos os recursos necessários para sua execução.

Pode ser visto intuitivamente que a estrutura do grafo *G* que será alvo do escalonamento tem uma grande influência no quantitativo de concorrência. Grafos menos densos irão garantir mais concorrência do que grafos mais densos. O quantitativo de concorrência obtido através do Escalonamento por Reversão de Arestas é altamente dependente da orientação acíclica inicial de *G*. Na Figura 3, observa-se que uma nova orientação acíclica é obtida a partir da inversão das arestas incidentes aos nós sumidouros.

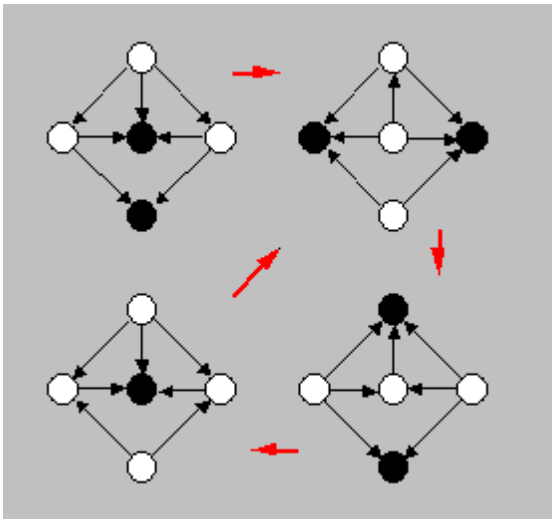

Figura 3 - Processo de reversão de arestas.

Um caso bem conhecido na literatura que pode ser solucionado com o ERA para o modo de operação em alta carga é o *Dining Philosopher Problem* (Filósofos à mesa) (DIJKSTRA, 1965). Neste problema, cinco filósofos estão sentados à mesa e

cinco garfos são dispostos para que os mesmos os usem para jantar, que neste caso seria a operação a ser realizada por cada filósofo, e os garfos, recursos necessários para operar (comer). A restrição imposta é que cada filósofo deve usar dois garfos para comer. A Figura 4 ilustra a situação citada acima.

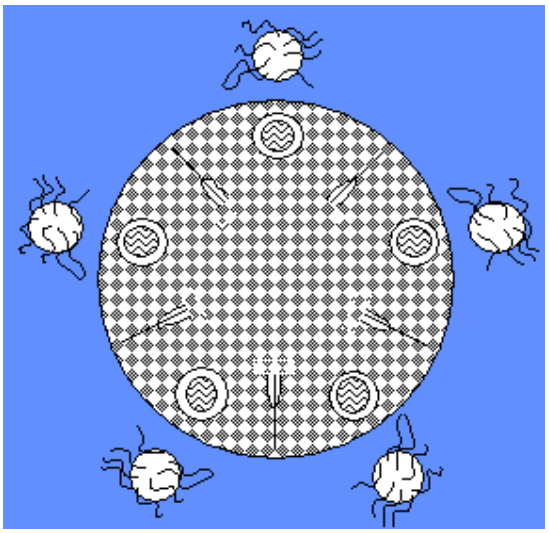

Figura 4 - ***DininG philosophers*** **(DIJKSTRA, 1965).**

Considerando um grafo $G = (V,A)$ que represente o problema, o conjunto V de vértices corresponde aos processos envolvidos no escalonamento (cada filósofo) e o conjunto A de arestas (garfo compartilhado por cada conjunto de dois filósofos subseqüentes). Para que um Filósofo consiga operar, ele deve obter dois garfos para tal, então, ao gerar uma orientação acíclica ω neste grafo G, um nó do grafo se tornará sumidouro e, portanto, poderá operar porque obtém prioridade sobre os recursos compartilhados.

É fácil perceber que sempre existirá uma coloração de nós de G associada a ω da seguinte maneira: cada nó recebe uma cor igual ao tamanho do maior caminho direcionado dele a um nó sumidouro. O *lâmbda* (λ) de um nó qualquer i no grafo, representa o maior caminho direcionado deste nó i a um nó sumidouro no grafo, isto significa que um nó sumidouro em ω recebe cor $\lambda = 0$, e esta é a decomposição por sumidouros de ω. Nesta decomposição por sumidouros, além dos nós que estão operando momentaneamente, também é possível observar quais nós serão os próximos a obter prioridade sobre os recursos (operar): os nós com $\lambda = 1$.

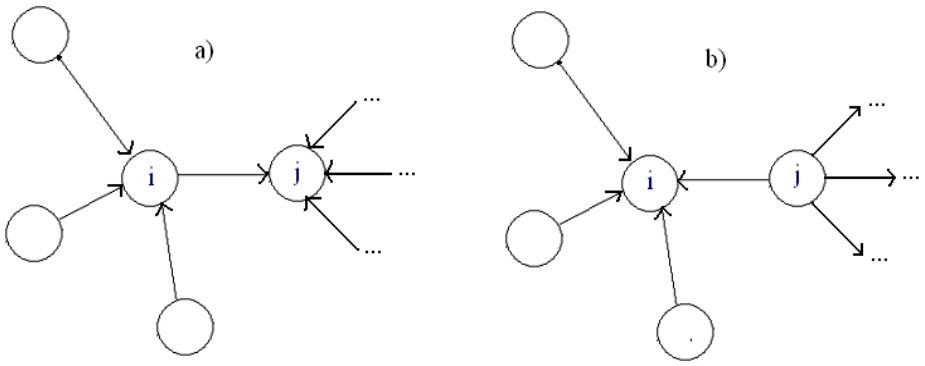

Figura 5 - Visualização local de uma execução do ERA.

Por definição, se um nó *i* possui $\lambda = 1$, o maior caminho desse nó a um nó sumidouro é apenas uma aresta e, portanto, não existe nenhum outro nó no maior caminho possível para alcançar um nó sumidouro (BARBOSA *et al.*, 1989, BARBOSA, 1996). Na Figura 5 (a), podemos ver que o nó i possui $\lambda = 1$ e o nó *j* é um sumidouro ($\lambda = 0$); Na Figura 5 (b), após a reversão de arestas de j, o nó *i* passa a ser o novo sumidouro. A partir desta observação local da execução do ERA em um grafo G, não é possível aferir qual seria o novo valor do lâmbda para o nó j. No entanto, o novo valor do lâmbda do nó i, de acordo com as definições, é igual a 0, e este nó passa a pertencer ao novo conjunto de nós sumidouros, ou seja, nós que possuem prioridades sobre os recursos. Desta forma, ao reverter suas arestas, um nó sumidouro faz com que os nós com $\lambda = 1$ tornem-se os novos sumidouros, garantindo que a dinâmica seja mantida (BARBOSA *et al.*, 1989, BARBOSA, 1996).

2.4.2 Decomposição por sumidouros

Sabemos que, em uma orientação acíclica ω de um grafo direcionado G, há pelo menos um nó que possui todas as arestas orientadas para si. Cada orientação acíclica pode ser também representada por uma, e somente uma, decomposição por sumidouros. Nesta decomposição, os nós são dispostos em camadas e cada camada representa o maior caminho de um dado nó a um nó sumidouro. A Figura 6 demonstra como uma decomposição por sumidouros pode ser representada.

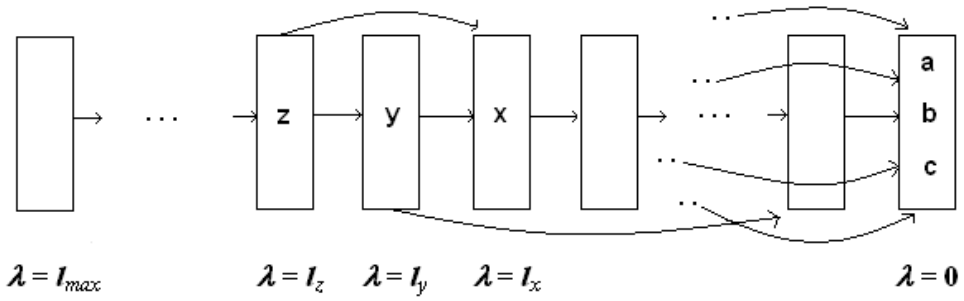

Figura 6 - Decomposição por sumidouros.

Cada camada também pode representar uma coloração para um dado nó do grafo *G.* Tendo camadas de 0 ao maior caminho a um sumidouro observado na dada orientação acíclica. Na Figura 7 abaixo, uma orientação acíclica pode ser observada:

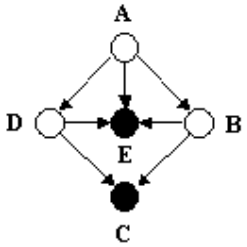

Figura 7 -Orientação acíclica ω em G.

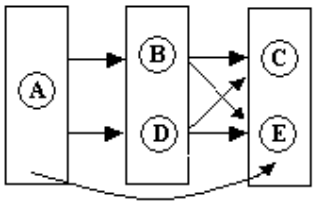

Figura 8 -Decomposição por sumidouros de ω.

A decomposição por sumidouros da orientação acíclica da Figura 7 está representada na Figura 8. Pode-se perceber que a maior camada nesta decomposição por sumidouros é 2. A camada do nó A é representada por $\lambda = 2$; Dos nós B e D, $\lambda = 1$ e dos nós C e E , $\lambda = 0$. A coloração do grafo mostrada na Figura 9 representa a decomposição por sumidouros mostrada na Figura 8.

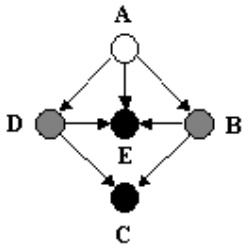

Figura 9 - Coloração em *G* para a orientação ω.

Na Figura 10, A cor branca representa o $\lambda = 2$; A cor cinza, $\lambda = 1$. A cor preta representa os nós com $\lambda = 0$, ou seja, sumidouros. Em cada coloração, o tamanho do conjunto de cores utilizado está diretamente relacionado com o valor máximo de λ no grafo direcionado. É possível perceber na Figura 10, que três (3) cores foram utilizadas para coloração, e esse quantitativo de cores corresponde ao número de camadas da decomposição por sumidouros mostrada na Figura 9.

2.4.3 Gerando Orientações Acíclicas Distribuidamente

Como dito anteriormente (Seção 2.4), para que seja iniciado o processo de Escalonamento por Reversão de Arestas, é necessário que uma orientação acíclica seja realizada no grafo *G* a ser utilizado no processo. Foi citado também que a primeira orientação acíclica em um grafo que será escalonado por reversão de arestas é um fator muito importante para o quantitativo de concorrência aferido durante o processo de reversão de arestas. Uma vez que seja gerada uma orientação acíclica que possui uma "larga" decomposição por sumidouros (em relação ao número de nós do grafo em questão), podemos assumir que o quantitativo de concorrência, ou seja, o número de nós sumidouros, será menor do que o observado em uma decomposição por sumidouros de menor comprimento no mesmo grafo. No entanto, achar um quantitativo ótimo máximo (BARBOSA *et al.*, 1989) ou mínimo (ARANTES Jr *et al.*, 2009) de concorrência são problemas um intratáveis.

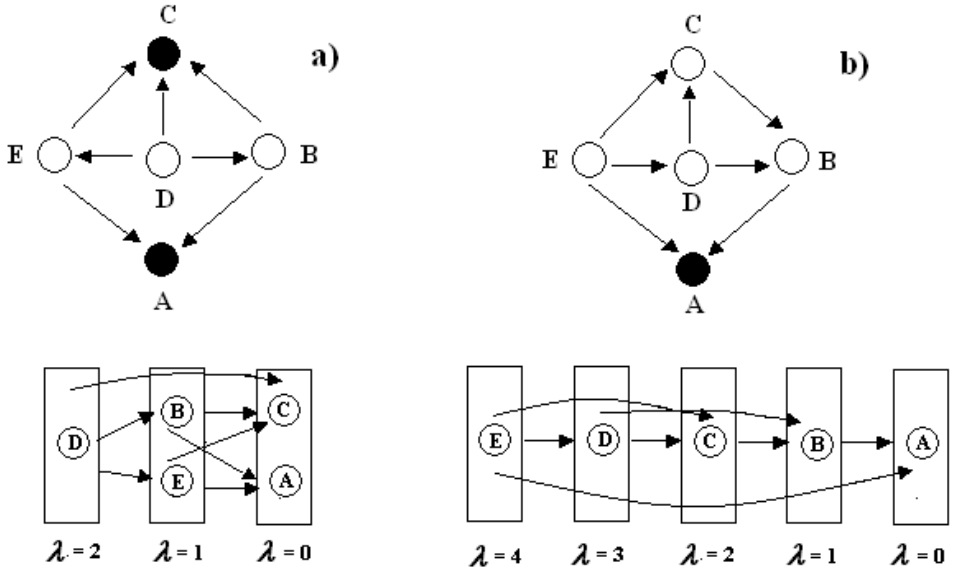

Figura 10 - Orientações ω e ω'.

Podemos ver na Figura 10 que em (a) o grafo *G* possui uma decomposição por sumidouros de tamanho três e em (b) o mesmo grafo possui decomposição de tamanho cinco. Ao "alongar" os caminhos no grafo, ocorre um aumento no número de camadas da decomposição por sumidouros. E com isso, o número de nós sumidouros passou a ser menor, o que em relação a um sistema de compartilhamento de recursos significa em uma diminuição da concorrência no grafo, ou seja, a quantidade de nós que operam ao mesmo tempo é reduzida em relação à primeira orientação.

A partir dessas observações, além de uma orientação acíclica, foi concluído que, para conseguir melhores resultados acerca do número de agentes utilizados, fosse usado um algoritmo que proporcionasse um número pequeno de nós sumidouros, i.e, que a concorrência fosse a menor possível, fazendo com que o número de agentes que executem concorrentemente também seja mínimo. Uma vez que ao ter decomposições por sumidouros longas aumenta a probabilidade de obter um menor número de nós sumidouros concorrentes, surgiu a necessidade do desenvolvimento de heurísticas capazes de realizar este tipo de resultado em um grafo. Com o objetivo de obter orientações acíclicas com longas decomposições, e que essas orientações fossem obtidas distribuidamente, pesquisou-se na literatura heurísticas que propusessem este tipo de resultado. As heurísticas randômicas *Alg-Cor*, *Alg-Viz* e *Alg-Arestas* foram introduzidas por ARANTES Jr. (2006) e a definição das mesmas é apresentada a seguir, nas Seções 2.2.3.1, 2.2.3.2 e 2.2.3.4.

2.4.3.1 Alg-Cor

Alg-Cor é um algoritmo utilizado para induzir uma coloração de nós de um grafo. Seja $G = (V,A)$ um grafo e C um conjunto de cores, uma coloração de G é uma atribuição de alguma cor de C para cada vértice de V, de tal modo que a dois vértices adjacentes sejam atribuídas cores diferentes. Assim sendo, uma coloração de G é uma função $f: V \rightarrow C$ tal que para cada par de vértices $(v,w) \in A \rightarrow f(v) \neq f(w)$.

Para atender a necessidade de um algoritmo distribuído para atribuir o valor da maior distância de cada nó a um sumidouro, neste caso, o algoritmo sofreu uma pequena alteração para colorir um grafo G conexo distribuidamente. Assuma uma operação assíncrona em um grafo G com n nós. Em cada passo do algoritmo cada nó i lança um dado de face 0 a n - 1. Considere $d_i = \propto$ onde $\propto \in \{0,1,2,..,n\text{-}1\}$. Caso um nó ganhe de todos os seus vizinhos, ou seja, $d_i > d_j$ para todo j pertencente a vizinho de i, então o nó i recebe uma coloração. Essa coloração é baseada na menor cor ainda não utilizada dentre seus vizinhos. O algoritmo termina sua execução quando todos os nós receberem uma coloração válida (ARANTES Jr, 2006).

Este algoritmo produz orientações acíclicas com caminhos curtos, ou seja, com o valor máximo de λ minimizado, aumentando o número de sumidouros obtidos. O passo que determina esse quantitativo é a determinação da próxima cor a ser utilizada (ao colorir um nó pode-se optar por utilizar uma nova cor ainda não utilizada, aumentando o tamanho do cojunto de cores – λ máximo). Ao alterar esse passo assumindo essa nova regra, caminhos maiores podem ser obtidos.

2.4.3.2 Alg-Viz

Considere que o algoritmo executa de forma síncrona. Um nó é dito probabilístico se ele ainda possui arestas incidentes não orientadas e continua tomando parte nos sorteios e, determinístico, no caso de não participar mais por já ter tido todas as arestas incidentes orientadas. Em cada passo do algoritmo todos os nós probabilísticos lançam uma moeda, obtendo 0 ou 1 (aqui representadas por $moeda_i$ para $i \in \{1,2,..,n\}$). Um nó que obtiver 1 e cujos vizinhos probabilísticos restantes tiverem obtido 0 irá orientar todas as suas arestas ainda não orientadas na sua direção. O algoritmo continua até que todas as arestas sejam orientadas (ARANTES Jr, 2006).

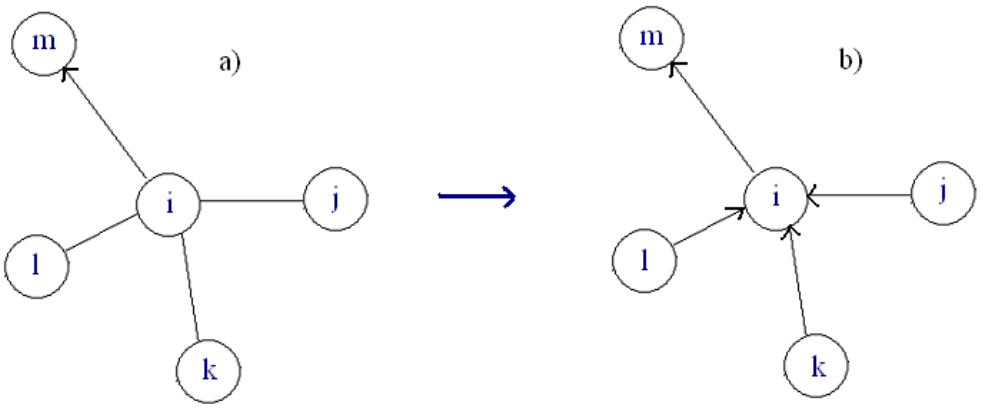

Figura 11 - Alg-Viz em execução.

Na Figura 11 (a), esta representada uma iteração *q* do algoritmo. Neste passo, as arestas ainda não orientadas que podem ser observadas indicam que os nós *i*, *j*, *k* e *l* ainda são probabilísticos. O nó *m* já não é mais probabilístico, pois já orientou sua aresta com o nó *i* para sua direção, o que significa que em uma rodada anterior $q' < q$, ele era probabilístico e obteve um ao lançar a moeda. Seu vizinho *i*, nesta iteração *q*', obteve zero ao lançar a moeda. Na Figura 11 (b), podemos perceber que o nó *i* orientou suas arestas com os vizinhos *j*, *k*, e *l* para sua direção. O que significa que ao lançar a moeda, *i* obteve um e todos os seus vizinhos probabilísticos (*j*, *k* e *l*) obtiveram zero.

Neste algoritmo, a moeda foi viciada, pois a ordem do tempo de convergência do algoritmo com uma moeda não viciada era sub-exponencial. Uma função acumulativa de probabilidade foi usada para viciar o dado baseada no número de vizinhos probabilísticos de cada nó ainda probabilístico. Esta função fez com que a ordem do tempo de convergência passasse a ser O(*n*), onde *n* é o número de nós envolvidos no processo (ARANTES Jr, 2006).

2.4.3.3 Alg-Arestas

Assuma uma operação síncrona do *Alg-Arestas* em um grafo $G = (V,A)$. Considere $d_i = \propto$ onde $\propto \in \{0,1,2,..,f\text{-}1\}$, o resultado associado a um nó $n_i \in V$, obtido após jogarmos um dado equilibrado com *f* faces. Para cada aresta $[n_i,n_j] \in E$ (ainda não orientada), se $d_i > d_j$ então orientamos $[n_i,n_j]$ na direção de n_i (aqui representada indistintamente por $n_i \leftarrow n_j$ ou $n_j \rightarrow n_i$). Se para alguma aresta tivermos $d_i = d_j$, repete-se novamente o processo até que todas as arestas tenham sido orientadas (ARANTES Jr,

2006). Na Figura 12, dois nós adjacentes *i* e *j* realizam o sorteio. Os valores obtidos são $d_i = 2$ e $d_j = 5$. Como $d_j > d_i$, a aresta $[i,j]$ agora possui o sentido de *i* para *j*.

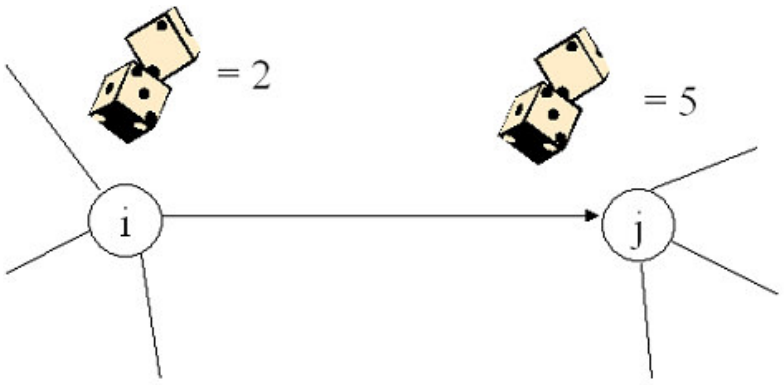

Figura 12 - Alg-Arestas.

2.4.3.4 Comparação dos comportamentos

Com o intuito de obter dados reais para comparação, os testes realizados em (ARANTES Jr *et al*., 2009) foram recriados. Para tal, os algoritmos antes construídos na linguagem C introduzidos por ARANTES Jr (2009), foram traduzidos para a linguagem Java™, a linguagem utilizada para o desenvolvimento de todos os métodos apresentados neste trabalho. Neste experimento, foram gerados grafos com 50 nós e conectividades {0.40, 0.475, 0.55, 0.625, 0.7, 0.775, 0.85, 0.925, 1} para comparação com os dados recuperados em ARANTES Jr (2009). Analisando os resultados obtidos pelos experimentos apresentados em ARANTES Jr et. al. (2009), podemos concluir que o algoritmo *Alg-Arestas* retorna um resultado compatível com a do nosso objetivo, que é o de aumentar os caminhos direcionados do grafo. O gráfico da Figura 13 explicita os resultados obtidos pela aplicação dos algoritmos propostos em (ARANTES Jr *et al*., 2009) nos grafos de características indicadas acima.

Com base nos resultados obtidos pelo *Alg-Arestas*, é aqui introduzida uma nova proposta para maximização dos caminhos (aumento das decomposições por sumidouros). Nas pesquisas realizadas, o resultado finalmente proposto foi o *Alg-Stretcher,* que será apresentado em seção subseqüente. Este algoritmo é aplicado ao resultado final do *Alg-Arestas* proporcionando um aumento nos caminhos produzidos por este.

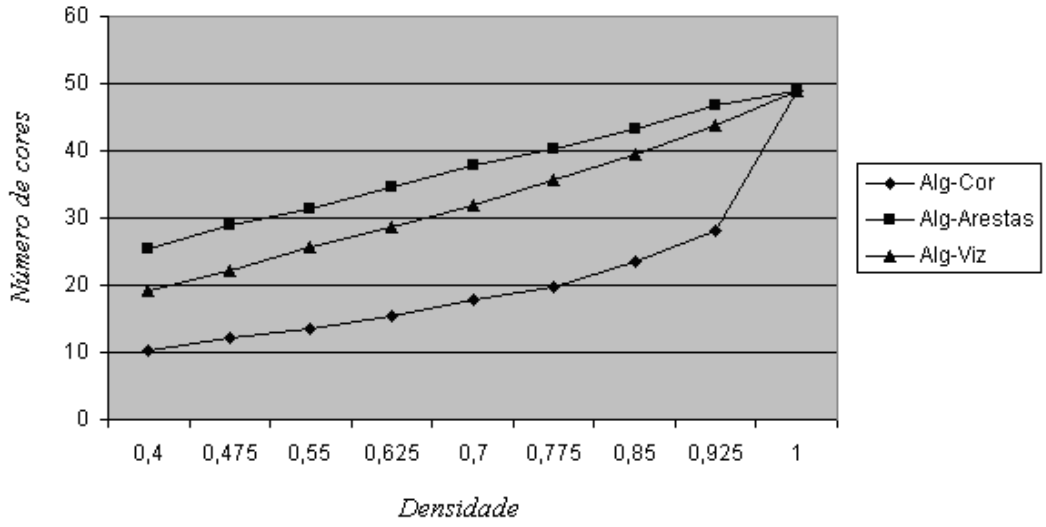

Figura 13 - Comparação dos dados obtidos pelas heurísticas.

2.4.4 Aplicações

Como antes citado, o ERA pode ser aplicado em problemas que podem ser reduzidos a grafos conexos, e esses indicam a principal característica dos mesmos: o conjunto de restrições que rege o problema central. Uma vez que um grafo conexo é definido para cada caso a ser escalonado, os dados podem ser aferidos facilmente a cada iteração do método ERA. A seguir, algumas aplicações encontradas em trabalhos relacionados serão especificadas.

O compartilhamento de recursos é um assunto bastante explorado na literatura e solucioná-lo requer uma ação não-trivial devido a sua complexidade. O ERA foi desenvolvido para priorizar execuções baseando-se nas restrições (vizinhança) de cada nó no problema ao qual está sendo aplicado. Um grafo $G = (V,A)$ pode representar um sistema de compartilhamento de recursos desta forma: cada nó representa um processo e, uma aresta existe entre dois nós se, e somente se, os dois processos correspondentes dividem pelo menos um recurso atômico (FRANÇA *et al.*, 1995). Em sistemas extremamente onerados com requisições e muitas dependências entre os processos (BARBOSA *et al.*, 1989), o ERA pode ser aplicado para coordenar a execução dos mesmos, sem que haja *deadlock* ou *starvation*. Em (PINHO et. al., 2009), o ERA foi aplicado no desenvolvimento de protocolos MAC livres de colisão com o propósito de diminuir a energia consumida pelas redes de sensores sem fio (PINHO *et al.*, 2009).

Um outro exemplo de aplicação do ERA pode ser observado em CASSIA *et al.* (2009) e FRANÇA *et al*. (2007), em que o método ERA foi utilizado no desenvolvimento de circuitos digitais assíncronos (*clockless*). O procedimento é baseado em uma temporização realizada pelo escalonamento por reversão de arestas assíncrono (*Asynchronous Scheduling by Edge Reversal Timing* - ASERT), um algoritmo totalmente descentralizado de sincronização e temporização, que funciona com atrasos combinados, relógios locais ou qualquer forma equivalente de determinar, estática ou dinamicamente, o tempo de operação de cada unidade funcional. Em LENGERKE *et al*. (2008a) foi utilizado para o escalonamento integrado de *Job Shop* e em LENGERKE *et al*. (2008b), a proposta seria implementar o ERA para prover um sistema automático de sinalização provendo um controle de tráfego para os AGV's (*Automated Guided Vehicles*) nas áreas do sistema flexível de manufatura. Em YAN*G et al.* (2003) e BRAGA *et al*. (2008), geradores rítmicos biologicamente plausíveis, como os CPGs (*Central Pattern Generators*) foram implementados baseando-se nas dinâmicas do ERA. Nesse projeto, o ERA foi utilizado para reproduzir ou prever o comportamento dos CPGs.

Com base nesses exemplos, é possível perceber a extensa aplicabilidade das dinâmicas do ERA. E, a partir desta proposta, outras dinâmicas também foram elaboradas, como o Escalonamento por Reversão Múltipla de Arestas (*Scheduling by Multiple Edge Reversal* – SMER (FRANÇA, 1993)) , que pode ser aplicado em grafos que possuam mais de uma aresta direcionada entre dois nós. A simples dinâmica proposta pelo ERA incorpora a particularidade de uma boa parte dos problemas encontrados na literatura relacionados a grafos, fazendo com que sua utilização no processo de exploração dos mesmos seja feita de forma eficiente.

2.5. Grafos da *Web*

A *World Wide Web* (Rede de alcance mundial, em português) ou simplesmente *Web* (WEB, 2010) foi idealizada primeiramente em 1980 por Tim Berners Lee e desde então obteve um crescimento exponencial. Após alguns estudos, mecanismos de rastreamento para colher informações a partir das construções de conglomerados de páginas *web* foram desenvolvidos. Estes mecanismos são chamado de *crawlers* (rastreadores) e os mesmos partem de um conjunto inicial de páginas, extraindo os

links e armazenando as informações obtidas para que as mesmas sejam analisadas posteriormente.

Diferentes formas de armazenagem têm sido estudadas para facilitar o acesso a esses dados recuperados pelos *crawlers*. Apesar do avanço obtido com relação ao espaço em memória física alcançado ao longo dos anos, é necessário compactar os dados obtidos por serem muito extensos. Algumas propostas foram analisadas, como o WebBase (HIRAI *et al*.,2000), o LINK *database* apresentado em RANDALL *et al*. (2001) e a plataforma WebGraph (BOLDI *et al.*, 2003), que serviu como base para os experimentos realizados neste trabalho e será explicitada em seção subseqüente.

As análises destas informações armazenadas mostraram que a forma com que são construídas as ligações entre as páginas da *web* fazem com que padrões possam ser reconhecidos. Esses padrões definem como um conjunto de páginas pode ser encaixado em uma estrutura. Algumas métricas como a extensão dos conjuntos de páginas que se interconectam, aferida por estudos anteriores, levaram a um estudo mais aprofundado acerca da estrutura da *web* em si. A seguir, uma abordagem realizada por dois desses estudos será apresentada.

2.5.1 Estrutura dos Grafos da *Web*

Gravata Borboleta

Os primeiros estudos em larga escala dos grafos da *web* (não confundir com grafos *web* – teia, estudados pelos teóricos (KOH *et al.*, 1980, GALLIAN, 2007)) foram feitos por Broder *et al*. (2000) revelando uma "imagem" bem definida da *web*. Como resultado desses estudos foi constatado que um grafo da *web* contém um componente de grande extensão que consiste em três componentes distintos de quase mesmo tamanho: (i) a parte principal chamada de CORE, representada por um componente único e fortemente conexo (também conhecida como componente gigante fortemente conexa); (ii) o conjunto de entrada (*IN set* em inglês), composto por nós que podem alcançar a parte principal, mas não podem ser alcançados por ela e (iii) o conjunto de saída (*OUT set*, em inglês), que são os nós que são alcançados pela parte principal mas não podem alcançá-la.

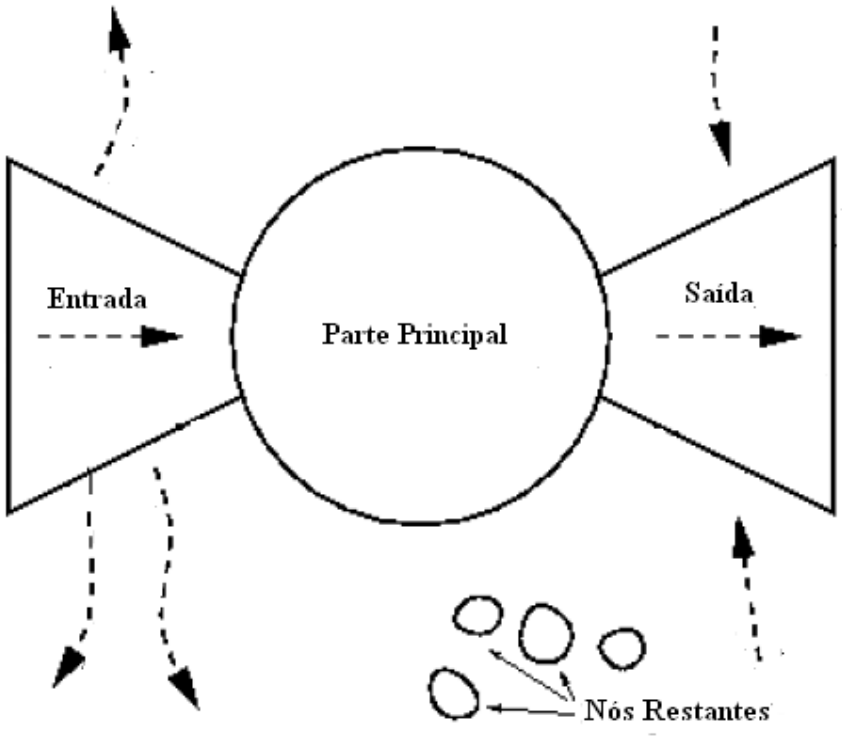

Figura 14 - Estrutura Gravata-borboleta (BRODER *et al*., 2000).

Esses três componentes formam a estrutura básica de grafos da *web* chamada de Gravata Borboleta, mostrada na Figura 14. Outros componentes secundários podem ser encontrados: Os ramos (*tendrils*, em inglês) compostos de nós não pertencentes a parte principal mas que são alcançáveis pelos nós do conjunto de entrada ou que podem alcançar os nós do conjunto de saída; E o desconexo (*disconnected*, em inglês) constituído pelos nós restantes (BRODER *et al*., 2000). Em DONATO *et al*. (2005) foi introduzido um conceito que defende que essa abstração macroscópica da *web* é pouco informativa acerca de detalhes da interconexões presentes na *web*. Após alguns experimentos, a estrutura em forma de Margarida foi apresentada.

Margarida

Nesta estrutura, os conjuntos de entrada e saída são fragmentados em vários grupos menores de pequenas e finas *pétalas* presas a densa parte central. Esta estrutura foi definida a partir da informação de que os conjuntos de entrada e saída são altamente fragmentados, enquanto a parte central é fortemente interconectada (DONATO *et al*., 2005). Após a apresentação desse conceito, foi concluído que ainda

não há como ter a comprovação de que estejamos lidando com uma estrutura neste formato, por ainda não ter em mãos mecanismos de rastreamento que possam corroborar esta informação. A Figura 15 demonstra a estrutura supracitada.

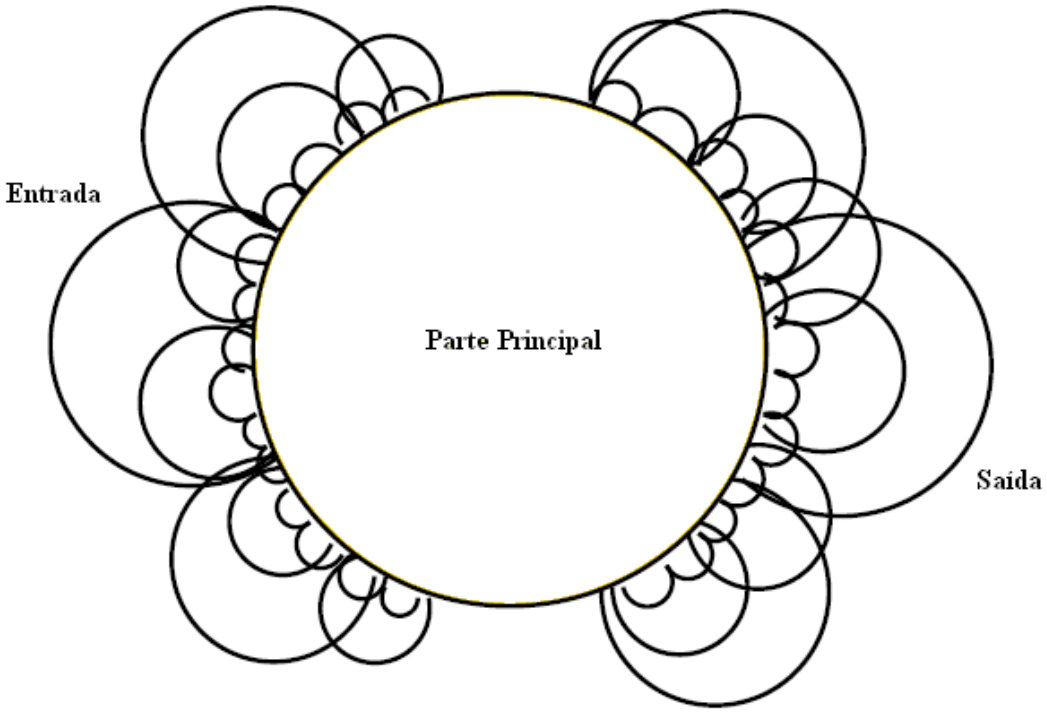

Figura 15 – Margarida (DONATO *et al.* , 2005).

2.5.2 Plataforma WebGraph

O grafo *web* relacionado a um certo conjunto de URLs é um grafo direcionado tendo essas URLs como nós e um arco de *x* para *y* cada vez que *x* possui um link para *y*. Como sabemos, o grafo da *web* é um objeto muito grande fazendo tornando não-trivial a tarefa de recuperar informações uma vez que não é possível armazená-las na memória principal (WEB, 2010). A Figura 16 elucida o processo de armazenagem das páginas da *web* e suas referências.

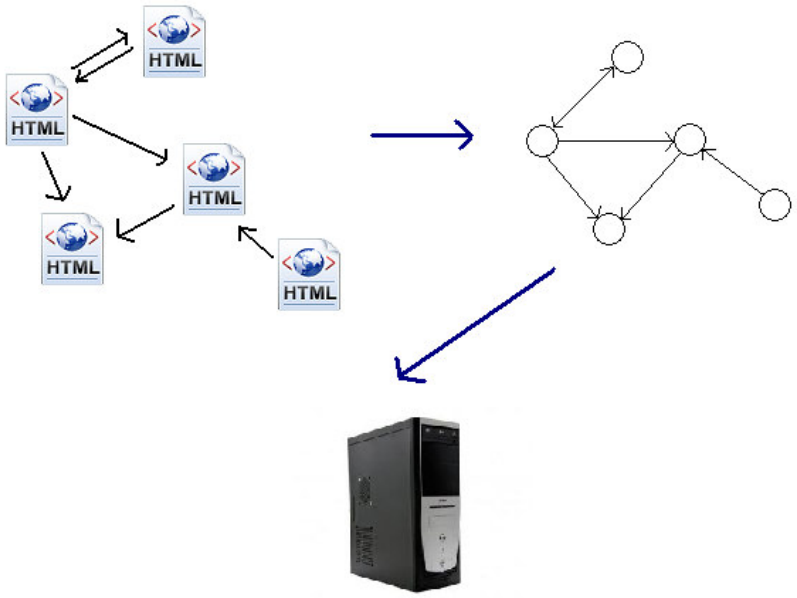

Figura 16 - Armazenamento de páginas web.

Métodos de armazenamento estão sendo desenvolvidos com o intuito de encontrar uma forma de minimizar o espaço ocupado pelos grafos recuperados da *web*. Nos grafos recuperados, vários tipos de arquivos Web devem ser analisados. Com este propósito, a plataforma *WebGraph* foi desenvolvida pelo *Laboratory for Web Algorithmics* (LAW), da Universidade de Milão. Esta plataforma foi desenvolvida em Java™ (BOLDI *et al.*, 2003a).

2.5.2.1 Descrição

Uma vez recuperadas as URLs e os *links* da *Web*, uma função elaborada pelos desenvolvedores é aplicada aos arquivos para conversão dos bits. Nesta função, um conjunto de inteiros em uma certa potência são utilizados para realizar a alteração necessária para que um conjunto de arquivos tenha seu tamanho reduzido, fazendo com que seu espaço de armazenamento diminua sensivelmente (BOLDI *et al.*, 2003a).

A plataforma foi construída para que os nós e seus links sejam acessados sem que seja necessário descomprimir o grafo já compactado pela metodologia anteriormente aplicada. Para esta plataforma, foram desenvolvidos navegadores - *Iterators* na linguagem de desenvolvimento – que, atendendo ao formato de

compressão utilizado, nos possibilita navegar pelos nós e seus sucessores de forma dinâmica (BOLDI *et al.*, 2004).

Depois de recuperadas e armazenadas na estrutura, cada página recebe um identificador. Na estrutura da plataforma, estão apenas armazenados os "*outlinks*" (*links* de saída) de cada página, ou seja, os *links* que a página que está sendo acessada contém. Portanto, em um grafo representando o grafo da web armazenado, para encontrar todos os vizinhos de cada nó, deve-se navegar por toda a estrutura, com o intuito de encontrar nós que possuam uma aresta para a página que está sendo acessada, mas não são alcançados pela mesma. Outras formas de compressão estão sendo estudadas e apresentadas na literatura atual. Neste projeto, vamos nos ater à plataforma criada por BOLDI e VIGNA.

2.5.2.2 Recolhimento dos dados e Armazenamento

Para acessar os grafos é preciso obter um dos arquivos compactados disponíveis no repositório de dados dos grafos retirados da web, os *datasets* da plataforma. Em cada um destes *datasets*, existe um conjunto de arquivos que definem as propriedades de cada grafo aferido da *web*. Dados como taxa de compressão dos links ou URLs. Cada *dataset* possui um nome especifico denominado *basename*, e este *basename* é usado para identificar a que grupo de propriedades cada arquivo pertence (WEBGRAPH, 2010).

Ao escolher um *dataset* para ser lido pela plataforma, é necessário obter pelo menos dois arquivos disponíveis no arquivo compactado. Os arquivos de extensão *.graph* e *.properties*. Com estes dois arquivos, pode-se ler o grafo a partir deoconstrutor da classe ImmutableGraph disponibilizada na plataforma para leitura dos grafos. Para isso, basta passar como parâmetro o nome da base escolhida, e posicionar os arquivos no diretório de trabalho (WEBGRAPH-DATA, 2010) (Apêndice B).

Após a leitura para memória, é necessário utilizar-se dos mecanismos disponibilizados na plataforma pra extrair a informação que se deseja. Neste trabalho, as informações importantes a serem recuperadas são acerca da vizinhança dos nós

presentes no grafo. E para isso, métodos para navegação e recuperação das informações sobre as adjacências de cada nó foram implementados.

2.6. Comentários do Capítulo

Neste capítulo foram apresentados os conceitos nos quais o método que será apresentado neste trabalho foi embasado: (i) contaminação de grafos, que é o problema a ser solucionado; (ii) o Escalonamento por Reversão de Arestas, metodologia na qual a rotina é fundamentada; (iii) a geração de orientações acíclicas, que possui grande influência sobre os números obtidos pelo procedimento e (iv) as estruturas nas quais deseja-se aplicar o método, trabalhos correlatos e novas abordagens, como a exploração da plataforma *WebGraph* para recuperação e descontaminação de grafos *web*. No próximo capítulo, serão apresentados o termo descontaminação de grafos e o algoritmo de descontaminação de grafos proposto.

3 ERA para Descontaminação de Grafos

Nesta seção será apresentado o método proposto nesta dissertação para descontaminação de grafos baseado no Escalonamento por Reversão de Arestas. Além disso, os procedimentos auxiliares para o algoritmo proposto serão especificados. Os dados apresentados foram obtidos de forma empírica.

3.1. Alg-D: Descontaminando com Reversão de Arestas

A descontaminação baseada em Reversão de Arestas funciona da seguinte forma: Agentes móveis (AMs) são associados a nós sumidouros; Uma vez que a descontaminação é feita em um nó sumidouro, novos AMs são replicados e enviados somente a nós imediatos que tornar-se-ão sumidouros após o término da execução do processo de descontaminação do agente, isto é, por reversão de arestas. Esta abordagem baseada em ERA associa implicitamente o quantitativo de concorrência dado pela dinâmica do ERA ao número de agentes concorrentemente operantes e o número total de passos realizados para a descontaminação, ou seja, um passo é contado toda vez que um nó recebe uma ou mais cópias de um AM ao tornar-se um sumidouro. O método proposto é capaz de funcionar em qualquer topologia arbitrária e sob regras de contaminação mais ou menos estritas, o que sugere a sua adequação em lidar com qualquer tipo de ataque em grafos conexos.

A partir de qualquer orientação acíclica ω de um grafo G, é possível iniciar a dinâmica de descontaminação baseada no comportamento do ERA. Com este objetivo, Agentes Móveis (AMs), que são entidades autônomas que podem ter seu comportamento alterado de acordo com o problema a ser resolvido, foram empenhados na resolução deste problema. Ao chegar a um nó, o agente executa a "limpeza" do mesmo e o mantém livre de ataques, caso necessário.

Começando por colocar os mesmos nos nós sumidouros (*home bases*), é fácil ver que nós que possuem cor $\lambda = 1$ serão os próximos a receberam a cor $\lambda = 0$ (sumidouros) após a reversão de arestas realizadas pelos nós sumidouros. Em geral, no caso da descontaminação de grafos, uma vez que um nó é exposto a um vizinho

contaminado, se ele não está protegido, pode ser contaminado novamente. No caso mais estrito, o número de vizinhos contaminados que pode re-contaminar um nó não guardado é igual a um (1). No entanto, uma generalização aceitável consiste em considerar que a maioridade de vizinhos contaminados pode contaminar um nó já limpo. Neste método, o critério de contaminação é um parâmetro que pode ser alterado.

3.1.1 Passo-a-passo do Alg-D

Os nós de um grafo *G* acíclico direcionado estão nos seguintes três estados locais: **contaminado**, **limpo** e **guardado**.

Tabela 1 - Possíveis estados e suas definições.

Estado	Descrição
Contaminado	Nó que ainda não recebeu um agente e/ou foi atacado novamente por um vizinho contaminado;
Guardado	O nó hospeda um agente móvel em execução;
Limpo	O nó não pode ser re-contaminado. Nenhum de seus vizinhos é capaz de atacá-lo.

São realizados os seguintes passos iniciais:

a) AMs são colocados nos *home bases* (nós sumidouros, $\lambda = 0$);

b) Nós sumidouros recebem o estado **Guardado** e todos os outros nós, **contaminado** (nós tem visibilidade um (1), isto é, um AM pode visualizar somente o nó no qual está executando e os seus vizinhos imediatos);

Enquanto houver um nó contaminado, cada nó verifica o valor do seu λ. Se $\lambda = 0$ e o nó está contaminado, então o nó verifica se recebeu um AM. o AM em execução no nó em questão limpa o nó e faz uma decisão: concluir a execução e mover-se para outro nó (um nó que irá tornar-se sumidouro) ou manter a execução e replicar-se, enviando as cópias para seus vizinhos que se tornarão sumidouros, isto é, mandar cópias para seus vizinhos em $\lambda = 1$. O AM irá terminar quando o nó atingir a condição necessária para a não re-contaminação.

Depois de mandar as cópias, um AM ativo manda mensagens a todos os vizinhos imediatos revertendo todas as arestas incidentes, produzindo uma nova orientação acíclica em *G* (um novo cálculo dos lâmbdas de cada nó é realizado de distribuidamente - Anexo A). Ao final da reversão de arestas de todos os nós sumidouros (em $\lambda = 0$), nós que estavam em $\lambda = 1$ passam a estar em $\lambda = 0$, recebendo uma ou mais cópias do AM. Deste momento em diante, o processo se repete até que não existam mais nós contaminados no grafo.

3.1.2 Corretude

Nesta seção será apresentada a corretude do método proposto apresentado na seção anterior.

Teorema 1. *Alg-D descontamina qualquer grafo conexo G=(V,A).*

Prova: Considere o grafo *G* = (V,A) e uma orientação acíclica ω_0 gerada neste grafo *G.*

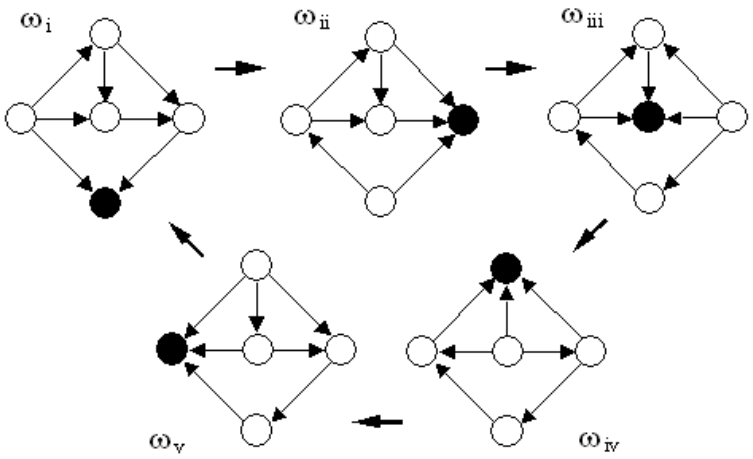

Figura 17 - Orientações acíclicas obtidas pelo ERA.

Na Figura 17, é apresentado um conjunto de orientações reproduzidas pelas dinâmicas do ERA. Pode-se perceber, que após algumas execuções, uma orientação acíclica gerada anteriormente é novamente alcançada, fazendo com que um ciclo de orientações acíclicas seja encontrado e esse ciclo recebe o nome de período. Esse período *p* é o tamanho do conjunto $\{\omega_i, \omega_{ii}, ..., \omega_v\}$, onde, ao reverter ω_v obtemos a mesma orientação percebida em ω_i. Pode ser observado também que no período *p*

ilustrado na Figura 17 todos os nós tornaram-se sumidouros e um nó que foi sumidouro na orientação ω_i deixou de ser sumidouro na orientação ω_{i+1}. Um sumidouro em ω_{i+1} possui ao menos um vizinho que é sumidouro em ω_i e em cada orientação ω_i um número m de nós foi sumidouro (BARBOSA *et al.*, 1989).

No *Alg-D*, uma vez que um nó é sumidouro, na primeira iteração, o mesmo recebe uma cópia do agente. A partir deste momento, os nós que tornar-se-ão sumidouros recebem cópias dos agentes enviados pelos vizinhos que eram sumidouros na orientação anterior e os mesmos, ao chegar no nó de destino realizam uma decisão acerca de que agente executará, fazendo com que apenas um agente esteja ativo no nó. Pela definição, todos os nós receberão agentes, uma vez que todos irão torna-se sumidouros eventualmente (BARBOSA *et al.*, 1989).

Outra preocupação é o critério de contaminação, que determina como um nó pode ser recontaminado caso não esteja guardado, ou seja, não possui um AM. A condição necessária para evitar que um nó seja recontaminado é baseado neste critério e o mesmo é o número de vizinhos contaminados de um determinado nó. No caso mais estrito, este número é igual a um e a condição necessária seria que todos os vizinhos de um nó limpo sejam limpos ou guardados. Nesta proposta, a recontaminação é prevenida ao manter a execução do agente nos nós que não alcançarem a condição necessária para evitar ataques de vizinhos contaminados. □

3.2. Alg-Arestas e Alg-Stretcher

No método proposto neste trabalho, para descontaminar um grafo é necessário que uma orientação acíclica seja imposta ao mesmo fazendo com que nós sumidouros sejam criados e, deste modo, agentes possam ser colocados no grafo. Após analisar heurísticas já presentes na literatura, e concluir após estudos que grafos com caminhos longos tendem a utilizar um menor número de agentes, foi elaborada uma heurística para ser aplicada ao *Alg-Arestas* que apresentou resultados consistentes com o objetivo de minimizar a concorrência nos grafos, ou seja, diminuir o número de nós sumidouros,

Para construir a heurística *Alg-Stretcher* (*Stretcher* significa "esticador" em português), foi usado como base um algoritmo de maximização de concorrência apresentado em (FRANÇA *et al*., 2002). Neste algoritmo, o objetivo é diminuir os caminhos construindo orientações a partir de uma orientação inicial e verificando se a nova orientação encontrada diminui o λ máximo do grafo. Esta medida indica o maior caminho direcionado de qualquer nó do grafo a um nó sumidouro.

O objetivo do *Alg-Stretcher* é produzir o efeito inverso, ou seja, aumentar o λ máximo em um grafo. Para isso, o princípio utilizado foi reverter arestas alterando a orientação acíclica gerada pelo *Alg-Arestas* e recalcular os λ's do grafo, verificando se o maior λ encontrado no grafo teve seu valor diminuído em relação ao λ máximo calculado antes das reversões. O *Alg-Stretcher* possui complexidade de tempo O(n) e complexidade de comunicação O(m).

```
N ← Número total de nós
Auxiliar ← λmax

PARA as camadas de N a 0
  PARA cada nó
      SE no.lambida = L

        Obtem ω' Invertendo Arestas
        λmax' ← Maior lambida em ω'

        SE λmax' > Auxiliar
          ω ← ω'
          Auxiliar ← λmax'

        SENÃO
          Mantém orientação anterior

        FIM SE

      FIM SE

  FIM PARA

FIM PARA
```

Figura 18 – Pseudo-Código para o Alg-Stretcher.

Definição: Seja $\lambda = l_{max}$ a maior camada da decomposição por sumidouros de uma orientação acíclica alvo ω. Das camadas $\lambda = (l_{max} - 1)$ até $\lambda = 0$, cada nó v em uma

camada $\lambda = l_v$, $0 \leq l_v < l_{max}$ é testado acerca de sua mudança para uma camada maior $\lambda > l_v$. Se com esta mudança houver um aumento no número de camadas da decomposição por sumidouros, a orientação acíclica ω' é obtida por manter todas as arestas de v orientadas na direção da decomposição por sumidouros do grafo, caso não haja nenhuma mudança , a orientação anterior é mantida.

O pseudo-código para o protocolo definido acima é apresentado na Figura 18. Após a aplicação do protocolo demonstrado na Figura 18, um novo grafo orientado acíclicamente pode ser observado. A entrada para este método é ω, uma orientação acíclica gerada pelo *Alg-Arestas*.

3.2.1 Corretude

O método apresentado na Seção 3.2.1 altera as propriedades do grafo gerado pelo *Alg-Arestas*. Em ARANTES Jr *et al*. (2009) foi mostrado que o *Alg-Arestas* não produz ciclos no grafo ao qual é aplicado. A partir deste conceito, deve ser mostrado que as medidas tomadas pelo *Alg-Stretcher* para gerar uma nova orientação também não produzem ciclos no grafo ao qual são aplicadas.

Lema 1: *O Alg-Stretcher não induz ciclos direcionados em G.*

Prova: Seja ω uma orientação acíclica gerada pelo Alg-Arestas. Figura 19 ilustra a decomposição por sumidouros de ω onde as camadas vão de $\lambda = l_{max}$ a $\lambda = 0$.

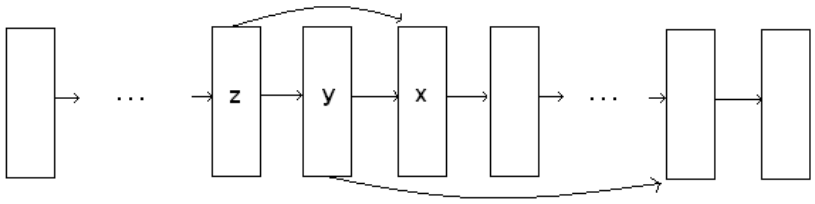

Figura 19 – Decomposição por sumidouros de ω.

A proposta do Alg-Stretcher é reverter a orientação das arestas incidentes a um nó v, pertencente a uma camada $\lambda = l_v$, que partem de nós em camadas maiores que a camada do nó v ($\lambda > l_v$). É conhecido que um ciclo é basicamente definido desta forma: um nó possui um caminho direcionado para ele mesmo, e uma orientação acíclica evita

que isto aconteça. A decomposição por sumidouros demonstra que não existe caminho direcionado de uma camada menor para uma camada maior. Na Figura 19 pode-se notar que o nó ***x*** faz parte do caminho de ***z*** e ***y*** para os nós sumidouros. Ao testar ***x*** para suas mudança para uma camada maior, será obtida a orientação ω' ilustrada na Figura 20.

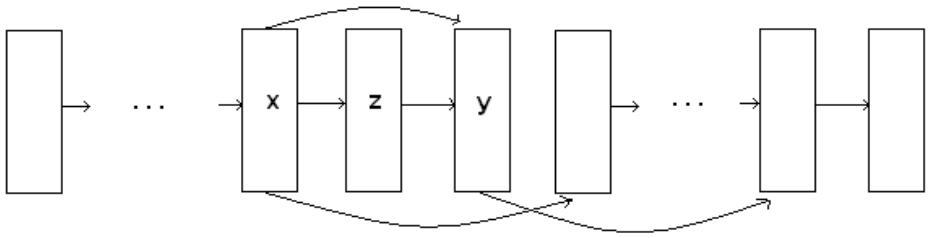

Figura 20 - Decomposição por sumidouros de ω'.

Ao reverter somente as arestas provenientes de camadas maiores que a do nó a ser testado, a criação de um ciclo no caminho para os sumidouros a partir do nó eleito *v* na decomposição não acontece, considerando que nessa orientação acíclica ω, um caminho direcionado do nó *v* para ele mesmo não existe. Na orientação anterior ω, ***z*** e ***y*** não possuem caminhos direcionados para eles mesmos assim como ***x***. Então, a única ação que poderia criar um ciclo seria a mudança de somente uma aresta direcionada de ***z*** ou ***y*** para ***x***.

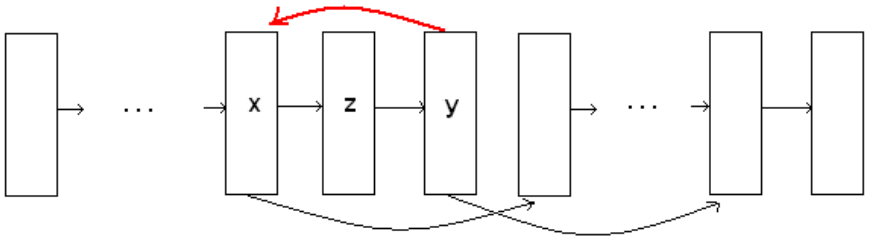

Figura 21 – Reversão causando ciclicidade.

Figura 21 demonstra que, ao não reverter a orientação da aresta $\{y,x\}$, um ciclo é criado. Como todas as arestas que poderiam criar ciclo ao não serem revertidas tem sua orientação, isto é, mantendo-se a direção da decomposição por sumidouros, é preservada a aciclicidade. □

3.2.2 Comparação dos comportamentos

Resultados experimentais da aplicação do *Alg-Stretcher* nas orientações acíclicas geradas pelo *Alg-Arestas* são representados na Figura 22. Cada ponto representa a média de 500 execuções dos algoritmos em grafos conexos randômicos com 50 nós e as conectividades indicadas. Comparado a outras duas heurísticas distribuídas (ARANTES Jr *et al*., 2009), um aumento expressivo no número de cores produzidas pode ser observado.

É esperado que o número de agentes a serem associados aos nós sumidouros resultantes será próximo do mínimo, tendo em vista que cada agente em execução representa um sumidouro no grafo em questão. É válido notificar que encontrar uma orientação acíclica associada ao (i) número mínimo de cores (BARBOSA, 1996), e ao (ii) número máximo de cores, são ambos problemas NP-completos (ARANTES Jr, 2006). Na Seção 3.2.4 será apresentado como a definição da NP-completude para concorrência mínima foi alcançada.

Outra característica que pode ser aferida a partir dos dados obtidos e representados na Figura 22 é que quanto maior o conjunto de vizinhos de cada nó no grafo (representado pela conectividade), maiores os caminhos encontrados e esse limite é dado pelo tamanho do grafo. A partir desta observação, pode-se concluir que a partir de um certo limite, tanto o *Alg-Arestas* quanto o resultado obtido pela aplicação da heurística desenvolvida retornarão o maior caminho possível no grafo em questão. Isso mostra que, em certos casos, assim como no aumento dos caminhos, não existe a possibilidade de otimização da concorrência.

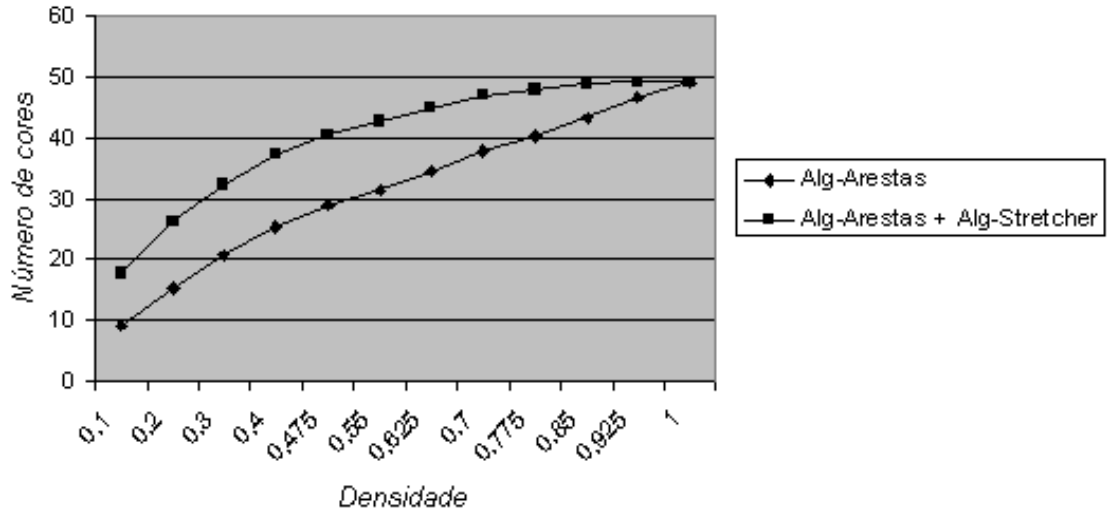

Figura 22 – Alg-Arestas vs Alg-Stretcher: número de cores gerada vs densidade.

3.2.3 NP-Completude da Concorrência Mínima

Como observado anteriormente (Seção 2.4.3), deve-se minimizar a concorrência no grafo para obter a minimização do número de agentes utilizados na descontaminação de um grafo. O algoritmo proposto para aumentar os caminhos do grafo e, conseqüentemente, diminuir a concorrência (*Alg-Stretcher*) possui certas limitações principalmente quanto à aproximação da concorrência mínima ótima para o problema apresentado. Em ARANTES Jr (2006) é mostrado que obter a concorrência mínima é um problema NP-completo, e isso é realizado através da redução de problema NP-completo para cúbicos planares introduzido em GAREY e JONHSON (1976).

Neste último, foi comprovado que obter um grafo Hamiltoniano cúbico planar é um problema NP-Completo. Seja um G = (V, A) - G grafo cúbico planar - uma instância de um ciclo Hamiltoniano (caminho em um grafo que passa por todos os nós sem que ocorra uma repetição) e n = |V|. Assumindo $\alpha = 1/n$ e $\gamma(G, \omega) - \omega$ uma orientação acíclica qualquer de G - a concorrência mínima ótima a ser obtida, deve-se provar que G possui um ciclo Hamiltoniano se, e somente se, G admite uma orientação acíclica com concorrência menor ou igual a α. A mínima concorrência no grafo deve-se a um ciclo simples, ou seja, um ciclo $k = |n|$. Ao assumir k como o ciclo Hamiltoniano do grafo G composto, em ordem de vizinhança $(v_0,...,v_n)$ e orientando cada aresta na direção do nó de menor índice para o nó de maior índice, teremos uma

orientação acíclica ω e o ciclo k formado pelas arestas (v_0, v_1), (v_1, v_2), ..., (v_{n-1}, v_n), (v_n, v_0) obtendo $\gamma(G, \omega) = 1/n$, provando a NP-completude da concorrência mínima (ARANTES Jr, 2006). Como dito na Seção 3.2.2, a concorrência máxima também é um problema NP-Completo (BARBOSA e GAFNI, 1989), mostrando que tanto a minimização de agentes para o caso da descontaminação quanto a maximização da concorrência para processos em um sistema de compartilhamento de recursos são problemas não-triviais.

Portanto, a solução ótima, que seria a menor concorrência possível possibilitando conseqüentemente o uso do número mínimo de agentes, não é uma solução fácil de ser encontrada. Uma vez tendo em mãos algoritmos que produzam orientações acíclicas com caminhos longos, o próximo passo é obter heurísticas que possam alongar ainda mais esses caminhos. O desenvolvimento da heurística do algoritmo aplicado ao Alg-Arestas foi baseado na mesma heurística que visa diminuir os caminhos desenvolvida para casos em que a concorrência máxima deseja ser alcançada (FRANÇA *et al.*, 2002).

3.3. Modelo para descontaminação de Grafos *web*

Os grafos *web*, por sua grande extensão, podem ser considerados bons pontos de partida para medir o comportamento do *Alg-D* em casos nos quais a contaminação a ser dissipada está presente em grafos de larga escala. Para reproduzir esses grafos, a plataforma *WebGraph* (seção 2.3.2) foi utilizada após estudos realizados acerca das propostas disponíveis relacionadas. A estrutura proposta pela plataforma não nos possibilita inserir qualquer informação adicional ao grafo recuperado após a compressão, fazendo com que uma estrutura auxiliar fosse desenvolvida para receber os dados necessários para realizar a descontaminação. Essa estrutura consiste em armazenar os estados de cada nó para posterior recuperação ao decorrer do algoritmo de descontaminação.

Uma vez que a estrutura gerada pela plataforma foi analisada e os experimentos iniciais foram realizados com a combinação da estrutura proposta e da estrutura auxiliar, concluiu-se que a melhor forma de utilizar as informações seria paralelizar a extração dos dados e descontaminação dos grafos obtidos a partir da base disponível (WEBGRAPH-DATA, 2010). Em máquinas com recursos reduzidos, os resultados

levaram um número excessivo de horas para serem recuperados, o que, em um caso de contaminação, pode ser tarde demais. A paralelização do algoritmo aqui proposto tem o propósito de tornar possível que o mesmo possa operar em várias máquinas e/ou processadores aumentando a sua performance.

A seguir, na Seção 3.3.1, os procedimentos realizados pelo *Alg-D* modificado para um ambiente de processamento paralelo serão demonstrados. Além desses procedimentos, a tecnologia utilizada para desenvolvê-los será apresentada. Por fim, algumas dificuldades encontradas ao decorrer da análise e implementação da solução sugerida serão apontadas.

3.3.1 Implementação dos Algoritmos

3.3.1.1 Tecnologia Utilizada

O algoritmo aqui proposto assim como as heurísticas necessárias para gerar os dados de tese, foram desenvolvidas com a tecnologia Java™. Uma linguagem orientada a objetos que nos permite recuperar as propriedades dos grafos armazenados com mais facilidade. E promover também a difusão do método aqui proposto em uma linguagem de maior acessibilidade, uma vez que a distribuição da mesma é livre.

Para paralelizar o algoritmo de descontaminação de grafos, foi utilizada uma biblioteca de passagem de mensagem entre processos chamada MPI (*Message Passing Interface*). Esta interface permite que processos disparados de um mesmo "programa", que rodem em diferentes processadores e/ou máquinas, possam trocar mensagens entre si. Várias implementações desta interface podem sem encontradas na literatura. Neste projeto, a interface utilizada foi a JMPI (Apêndice D).

A idéia seria dividir a extração dos dados obtidos através da plataforma *WebGraph* em vários processadores e máquinas, evitando a sobrecarga e acelerando o processamento. Uma vez recuperadas as informações, os nós são divididos em conjuntos e cada conjunto é processado em um processador distinto. Cada atualização realizada em um dos processos é passada por mensagem para os outros processos para que um estado global do processamento seja garantido e essas mensagens são estruturadas para que cada processo possa reconhecer que tipo de requisição ou

informação está recebendo, realizando a ação necessária (enviando uma resposta ou atualizando uma variável local).

3.3.1.2 Descrição da Metodologia Aplicada

Seja p o número de processos designados para a descontaminação. A partir do número n de nós presentes no grafo a ser descontaminado, uma divisão é feita para que cada processo "enxergue" apenas uma parte do grafo. Cada processo p é guardião de um conjunto de nós com tamanho e intervalo de identificadores a ser definido pelo número total de processos e número total de nós no grafo (Anexo C). Os conjuntos são definidos da seguinte forma:

a) Divide-se o número total n de nós pelo número p de processos obtendo-se o valor $l = \lfloor n/p \rfloor$. Caso a divisão seja um número inteiro, processo r, $0<r<p$, será obtido um intervalo de tamanho l.

b) Caso exista, o resto da divisão de n por p é incluído no processo $r=p$-1. (O tamanho do intervalo do processo p-1 será l+(n *mod* p).

Na Figura 23 abaixo, encontra-se uma elucidação de como a divisão dos nós entre os processos é realizada. Após a divisão dos intervalos, cada partição faz uma verificação contínua do estado de seus nós percorrendo o grafo, requisitando através de mensagem para o devido processo o estado de algum nó que não esteja visível em seu escopo. Na plataforma *WebGraph*, cada nó possui um identificador único e a orientação acíclica inicial , neste caso, é dada por este indicador, ou seja, cada nó de identificador i tem suas arestas apontados para os nós com identificadores $j<i$, $\{j, i\} \in$ A, fazendo com que na primeira iteração, pelo menos o nó de identificador zero seja o nó sumidouro.

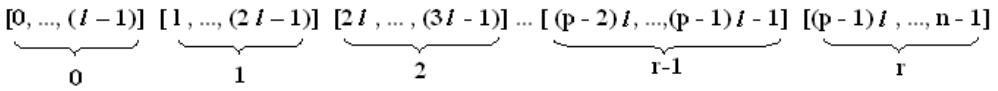

Figura 23 - Representação dos intervalos atribuídos a cada processo.

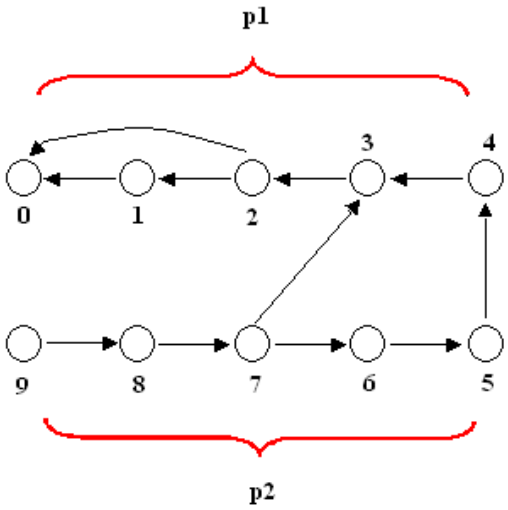

Figura 24 - Grafo G=(V,A) com |V| = 10 e 2 processos.

Na Figura 24 uma ilustração de como um grafo G = (V,A) seria dividido em conjuntos caso o número de processos designados para realizar a operação fosse igual a dois para um total de 10 nós. Neste exemplo, para que o nó de identificador 4 obtenha o estado do nó de identificador 5, e vice-versa, mensagens devem ser trocadas pelos processos. A mesma coisa acontece quando o nó de identificador 7 precisa recuperar as informações do nó de identificador 3. Na Figura 25 (a), mostra-se a estrutura da mensagem enviado pelo processo p_2 ao processo p_1 solicitando o estado do nó de identificador 4 e na Figura 25 (b), a resposta dada pelo processo p_1 ao processo p_2 (O "x" representa o valor inteiro correspondente ao estado do nó em questão). Os nós pertencentes ao mesmo conjunto realizam consultas localmente sem que seja necessário que mensagens sejam enviadas.

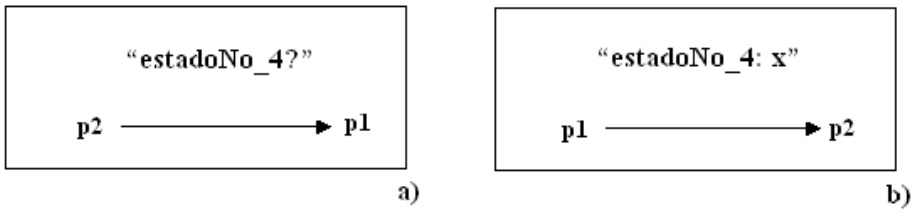

Figura 25 - Estrutura das mensagens de requisição e resposta acerca do estado de um nó.

A partir desta orientação inicial, cada nó vira sumidouro quando todos os vizinhos menores que ele já tenham sido alcançados. A partir deste momento, o algoritmo segue as especificações do *Alg-D*, tomando como guardados os nós

alcançados, contaminados os nós não alcançados e limpos os nós alcançados que possuam o número de vizinhos contaminados correspondente ao critério de recontaminação utilizado, que neste experimento foi de 50% dos vizinhos contaminados. O algoritmo é finalizado quando o número de nós contaminados chega a zero e cada processo realiza esta verificação a partir de um arquivo de texto que possui acesso sincronizado. Quando esse estado global é alcançado, cada processo é finalizado terminando a operação. Na Figura 26, a estrutura de mensagem que cada processo envia aos demais contendo a informação sobre o número de nós contaminados sobre sua guarda é apresentada. O processo p_i envia a informação a todos os outros processos e não somente a um outro processo como no caso da requisição de estado de um nó com identificador específico (o "*y*" na Figura 26 representa o valor inteiro que representa o quantitativo de nós contaminados).

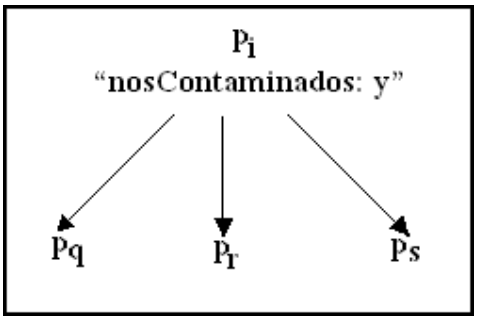

Figura 26 - Estrutura da mensagem enviada contendo a informação sobre a quantidade de nós contaminados no processo p_i.

Lema 2: *A orientação inicial ω_0 induzida no grafo a partir dos identificadores de cada nó no grafo G =(V,A) é acíclica.*

Seja $G = (V,A)$ um grafo conexo, com $n = |V|$ e v_i um nó pertencente a G com $0 \leq i < n$ como ilustrado na Figura 27. Para iniciar a descontaminação, uma orientação acíclica inicial ω_0 é induzida no grafo G onde para cada nó v_j , $\{v_i, v_j\} \in A$, $j>i$, a aresta $\{v_i, v_j\}$ é orientada na direção de v_j para v_i. Pela definição da estrutura disponibilizada pela plataforma *WebGraph*, todos os nós são armazenados com identificadores inteiros positivos e seqüenciais e não existem dois nós com um mesmo valor de identificador. Com base nesta afirmação, pode-se assegurar, que uma vez que as arestas sejam orientadas no sentido do menor identificador, não haverá ciclos na orientação inicial, tendo em vista que, para haver ciclo seria necessário que um nó de

identificador $p>q$, com $\{p,q\} \in$ A, tivesse a aresta orientada no sentido de q para p, causando um clico no grafo.

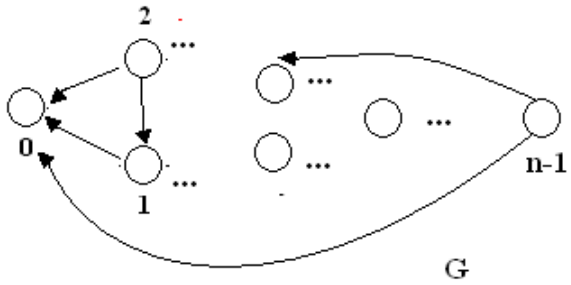

Figura 27 - Orientação inicial ω_0 em G.

Teorema 2: *O Alg-D paralelo descontamina um grafo qualquer conexo a partir de uma orientação inicial acíclica ω_0 induzida pelos valores dos identificadores de cada nó.*

A partir da orientação inicial acíclica ω_0, é possível perceber pelo Lema 1 que o único nó a ser sumidouro em ω_0 é o nó de identificador zero (0), pois não existe nenhum nó com identificador $i<0$. Então para este algoritmo, como na definição do *Alg-D*, os nós sumidouros recebem agentes na primeira iteração. Neste caso, o nó zero (0) que é o único sumidouro, recebe um agente mudando do estado **contaminado** para **guardado**. Após esse passo, cada nó vizinho de 0 no grafo é testado para ser sumidouro, baseando no estado de seus vizinhos e nos identificadores, ou seja, o próximo nó a tornar-se sumidouro será o nó que possuir vizinhos **guardados** ou **limpos** com identificadores menores que o dele, pois seus vizinhos com identificador maior ainda não foram alcançados e, portanto, apontam suas arestas na direção dele. Esse passo indica como o processo de reversão de arestas é realizado neste caso e garante que, um nó com identificador menor tenha sido alcançado antes, evitando que uma ciclicidade seja induzida no grafo.

Para verificação dos estados de cada nó, cada processo p possui um vetor que armazena os estados dos nós aos quais ele detém a guarda, e esses estados são representados por valores inteiros que são atualizados sempre que há uma mudança de estado decorrente de alguma operação realizada pelo método. Ao necessitar da verificação do estado de um nó vizinho no qual o processo não possui acesso direto

(nós fora do seu intervalo de percepção do grafo), ele solicita ao processo detentor o estado do nó em questão por meio de mensagem. Enquanto o estado do nó solicitado não é recebido, também por mensagem, este nó é considerado contaminado, evitando que haja recontaminação de nós, uma vez que, ao assumir que o nó esteja limpo ou guardado sem uma confirmação, ocasionaria uma possível recontaminação do grafo. Para que um nó seja considerado limpo, o mesmo deve possuir uma porcentagem de vizinhos guardados ou limpos relação ao número total de vizinhos, como no *Alg-D* (comumente igual a 100% ou 50%). A cada iteração, após a verificação da ocorrência de uma reversão de arestas, são verificados novamente os estados dos vizinhos de cada nó guardado (localmente ou por meio de mensagem), e são considerados limpos os nós nos quais a porcentagem de vizinhos limpos atinge a porcentagem previamente estabelecida. Deste modo, uma vez que eventualmente todos os nós serão alcançados no grafo, o *Alg-D* paralelo descontaminará o Grafo.

3.3.1.3 Dificuldades Encontradas

Para desenvolver o método descrito na seção acima, um esforço adicional foi empenhado para analisar e apurar as novas características da plataforma em questão. Além da análise das peculiaridades envolvidas, a forma com que as mesmas seriam inseridas no contexto do ERA foi elaborada. Com esta finalidade, foram desenvolvidos procedimentos para inserir os dados obtidos no contexto indicado, levando em consideração o cenário apresentado, como a forma de identificar nós sumidouros no grafo.

Após definir como a dinâmica do ERA funcionaria neste caso, algumas limitações (que incluem algumas peculiaridades em relação ao envio e recebimento de mensagens - travamento) quanto ao desempenho e documentação da tecnologia aplicada fizeram com que algumas medidas antes inseridas baseando-se na análise inicial fossem alteradas, como a forma como os processos se comunicam. Essas limitações existem pelo fato de que a implementação da interface utilizada (*Message Passing Interface*) em Java ainda estar em testes e constantes alterações, além de possuir poucas referências.

Além disso, a forma com que decorrem-se as operações tanto na plataforma *WebGraph* quanto na interface de passagem de mensagem não são rastreáveis

(procedimentos encapsulados), fazendo com que o desenvolvimento fosse baseado nos resultados finais, aumentando a incerteza. Este tipo de arquitetura impede que os problemas encontrados possam ser resolvidos com rapidez, retardando o processo de aprendizagem. Além disso, a extensão dos grafos utilizados e a forma com que cada nó e seus vizinhos devem ser acessados dentro da plataforma fizeram com que a utilização de memória dos recursos tivesse que ser otimizada.

3.4. Comentários do Capítulo

Neste capítulo, a definição da metodologia de descontaminação de grafos foi apresentada. Foi mostrado também que outros métodos foram implantados para que os resultados obtidos fossem satisfatórios, observando as restrições a serem levadas em consideração para cada tipo de problema. Os problemas encontrados ao aplicar este método em grafos de grande extensão foram expostos. No próximo capítulo, os resultados obtidos pela aplicação do método demonstrado neste capítulo serão exibidos.

4 Análise dos Resultados

Neste capítulo, serão mostradas as avaliações experimentais realizadas com o novo algoritmo em problemas já abordados na literatura.

4.1. Comparação com Trabalhos Correlatos

Nesta seção, serão apresentados os resultados obtidos pela aplicação do método aqui proposto nos cenários propostos em FLOCCHINI *et al*. (2005), FLOCCHINI *et al*. (2007) e LUCCIO *et al*. (2007). O objetivo desta comparação é demonstrar que os resultados obtidos pela proposta aqui apresentada estão satisfatórios. Nestes trabalhos, agentes móveis foram utilizados.

Em FLOCCHINI *et al*. (2005) é usada uma técnica baseada em algoritmos genéticos para colocar nos grafos produzidos os primeiros agentes de descontaminação. Estes nós que inicialmente recebem estes agentes são chamados de *home bases.* Uma vez que estas *home bases* são determinadas, é iniciado um processo de inundação de agentes no grafo em questão. Para tal, um algoritmo baseado em BFS (Seção 2.1.1) é utilizado e funciona da seguinte forma: cada agente descontamina o nó no qual está executando, replica-se e envia estas replicações a todos os vizinhos imediatos. O processo é finalizado quando todos os nós do grafo recebem os agentes. É fácil perceber que, uma vez que não há distinção entre os vizinhos que receberão agentes, haverá um número excessivo de agentes enviados.

Uma melhoria deste algoritmo pode ser observada em FLOCCHINI *et al*. (2007), pois um novo parâmetro é inserido. Este parâmetro é a visibilidade de cada nó e pode ser descrita como o alcance de cada nó a seus vizinhos no grafo. Na Visibilidade 2, todos os nós enxergam seus estados, os estados de seus vizinhos imediatos , e os estados dos vizinhos imediatos a seus vizinhos (Em FLOCCHINI *et al*. (2005), os nós possuíam visibilidade 1). Com esta nova solução, foi alcançada uma diminuição no número de agentes utilizados para a descontaminação.

Em LUCCIO *et al*. (2007) são abordadas as *Link Farms*. Essas *link farms* contêm dispositivos para evitar que as conexões inseridas de forma maliciosa sejam

rompidas. Vários métodos para proteger essas *farms* de serem extintas têm sido desenvolvidos (GYONGYI *et al*., 2005). Procedimentos que visam a destruição dessas *farms* foram propostos na literatura (FLOCCHINI *et al*., 2005, LUCCIO *et al*., 2007, FLOCCHINI *et al*., 2007).

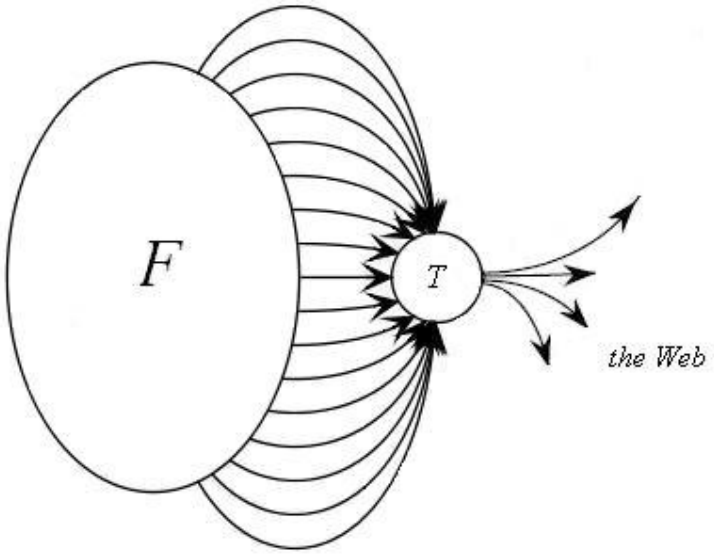

Figura 28 - Link Farm.

Foi assumido que uma *link farm F* (ver Figura 28) tem como estrutura principal um grafo circulante – ou grafo algébrico - (LUCCIO *et al*., 2007) e o trabalho proposto por eles é fortemente baseado nas propriedades da topologia deste tipo de grafo. Foi mostrado que em um grafo circulante $C_{i,n}(L)$, com L sendo uma lista de inteiros $\{1, 2..., k\}$ e k sendo o tamanho desta lista, $k + 2$ AMs são necessários para extinguir uma *link farm*. Na Figura 29 (a) $k = 3$ $k = |L|$, L sendo $\{1,2,4\}$), e na Figura 32 (b) $k = 2$ ($k = |L|$, L sendo $\{1,4\}$).

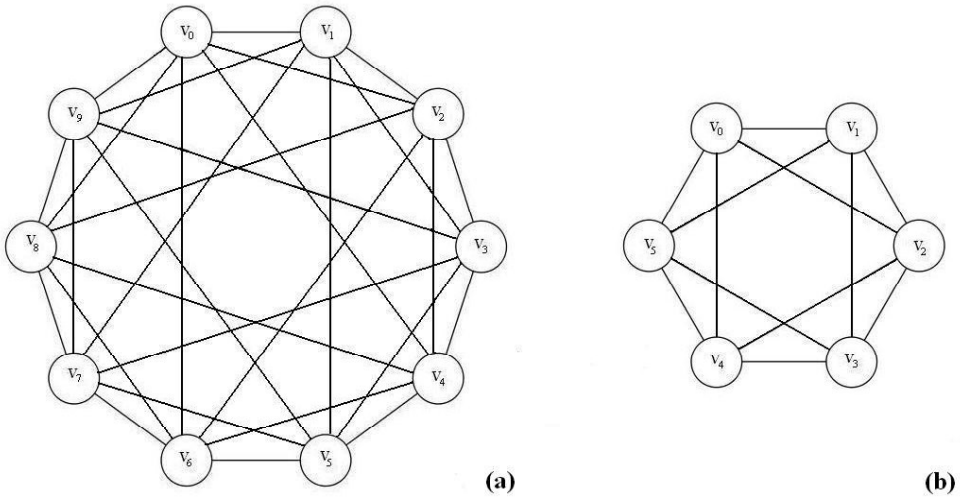

Figura 29 – *Circulant Graphs* (Grafos Circulantes).

Este tipo de infecção também vem sendo abordado na comunidade como um problema de busca em grafos e mais especificamente, descontaminação de grafos. Neste contexto pode-se dizer que as *link farms* reproduzem um grafo contaminado: As páginas pertencentes a esta são os nós do grafo e os *links* para as páginas do conjunto, as arestas. Além disso, o *link* para a página *T* pode ser considerado como a contaminação do grafo a ser extinta. Neste caso, o objetivo dos agentes é percorrer nó por nó (ou página por página), destruindo o *link* para a página *T* em questão.

Baseados na definição de que as *link farms* possuem como forma principal grafos circulantes, em LUCCIO *et al*. (2007) foi proposto um algoritmo distribuído, trabalhando síncrona e assíncronamente, para ser acoplado a agentes autônomos, chamados de *Web Marchals*, com o objetivo de desmantelar estas *link farms*. O número de agentes e o número de passos utilizados na descontaminação foram apresentados.

4.1.1 BFS e *home bases* aleatórias

Nesta seção, os dados obtidos Em FLOCCHINI *et al*. (2005) e FLOCCHINI *et al*. (2007) serão comparados aos obtidos pelo método proposto neste trabalho. Para demonstração dos resultados, nos dois trabalhos citados anteriormente, foi usado o seguinte cenário: grafos de [512, 768, 1024, 1576, 2048] nós com densidades [4, 8, 16, 32]. As Figuras 30, 31 e 32 demonstram os resultados obtidos em FLOCCHINI *et al*.

(2007). O eixo vertical determina a média do número de agentes utilizados para descontaminação realizada pelo método proposto e o eixo horizontal, representa as *home bases* utilizadas, que pode ser traduzida como o número de nós que recebem agentes na primeira iteração.

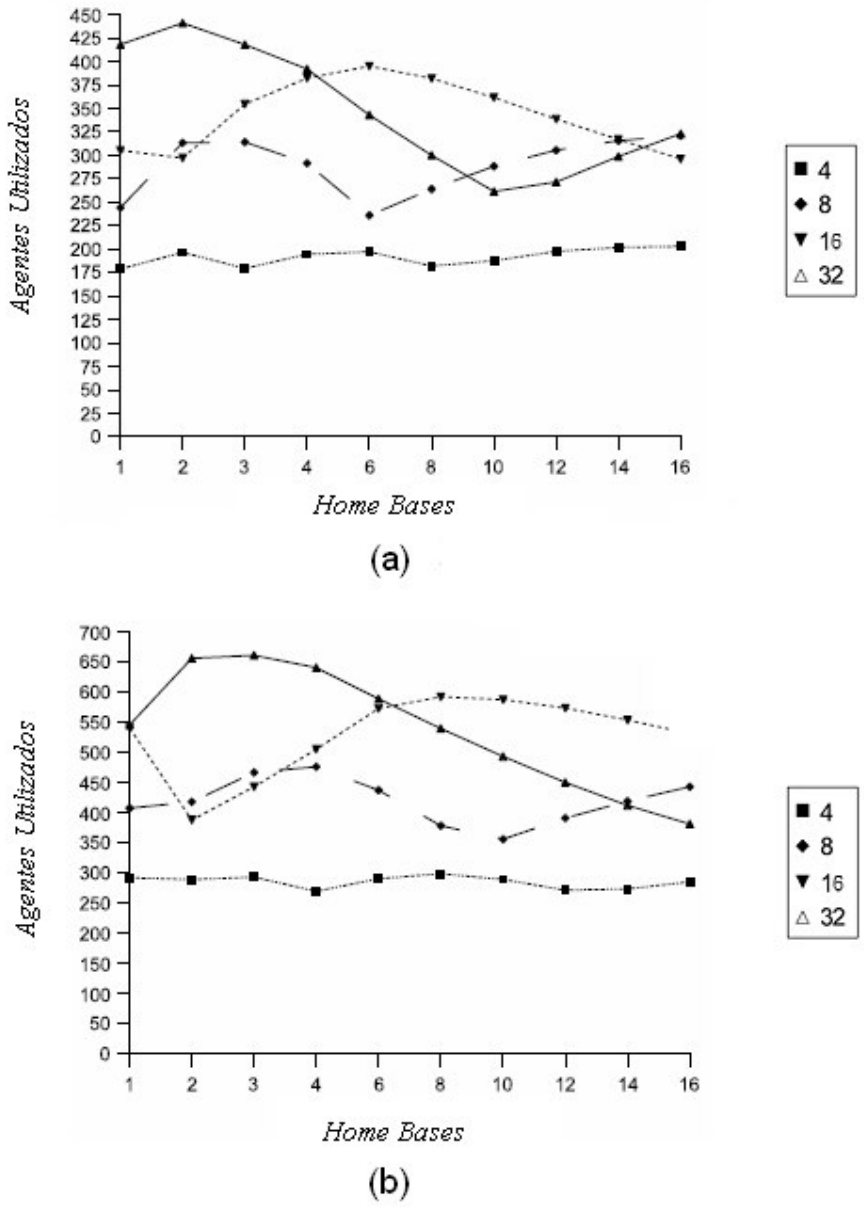

Figura 30 - Dados representados para 512 (a) e 768 (b) nós respectivamente nós (FLOCCHINI *et al.*, 2007).

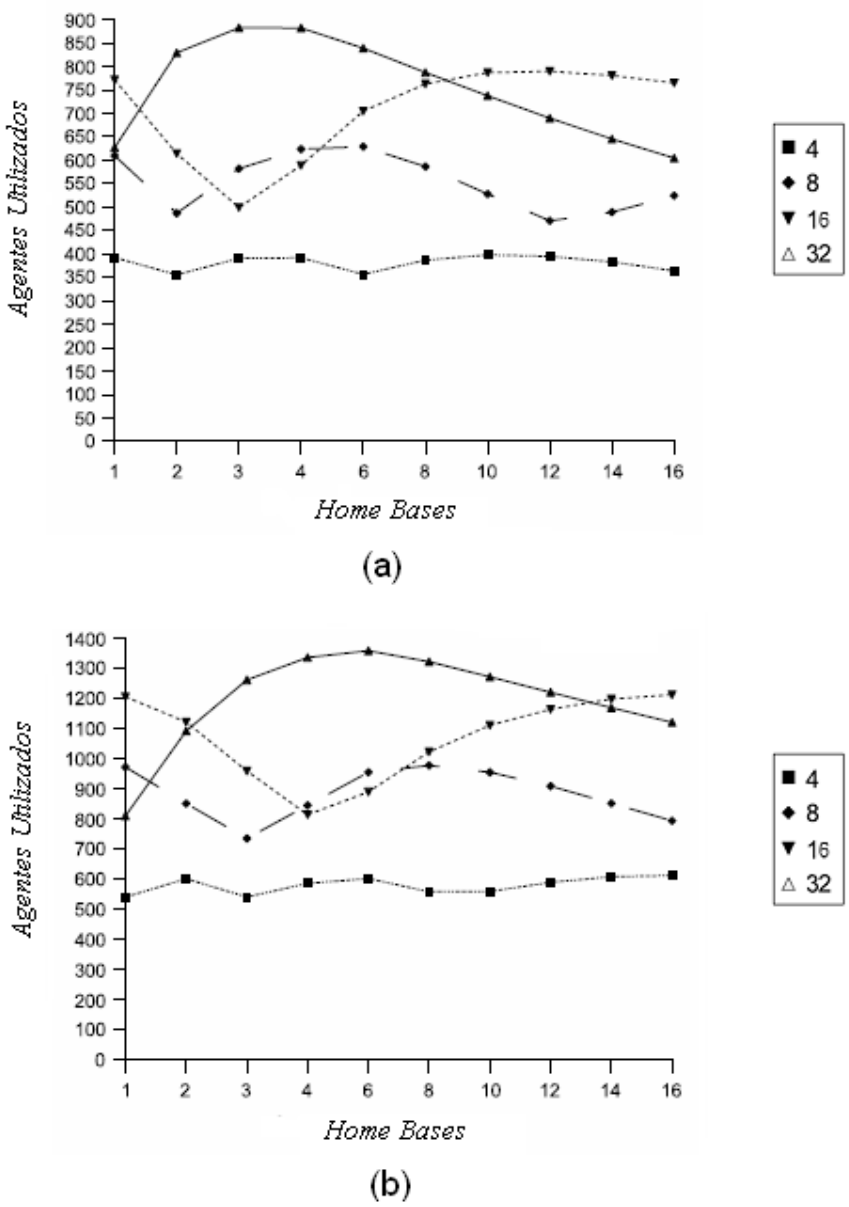

Figura 31 - Dados representados para 1024 (a) e 1576 (b) nós respectivamente nós (FLOCCHINI *et al*., 2007).

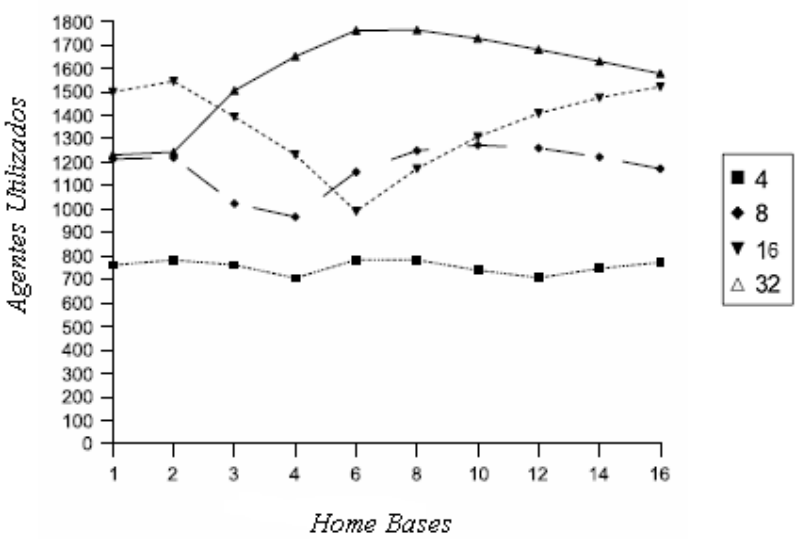

Figura 32 - Dados representados para 2048 nós (FLOCCHINI *et al.*, 2007).

Os dados da Figura 30 (a) representam os grafos com 512 nós e a Figura 30(b), 768. Na Figura 31 (a) estão representados os dados dos grafos com 1024 nós e na Figura 31(b), os dos grafos com 1576 nós. A Figura 32, apresenta os dados de grafos com 2048 nós.

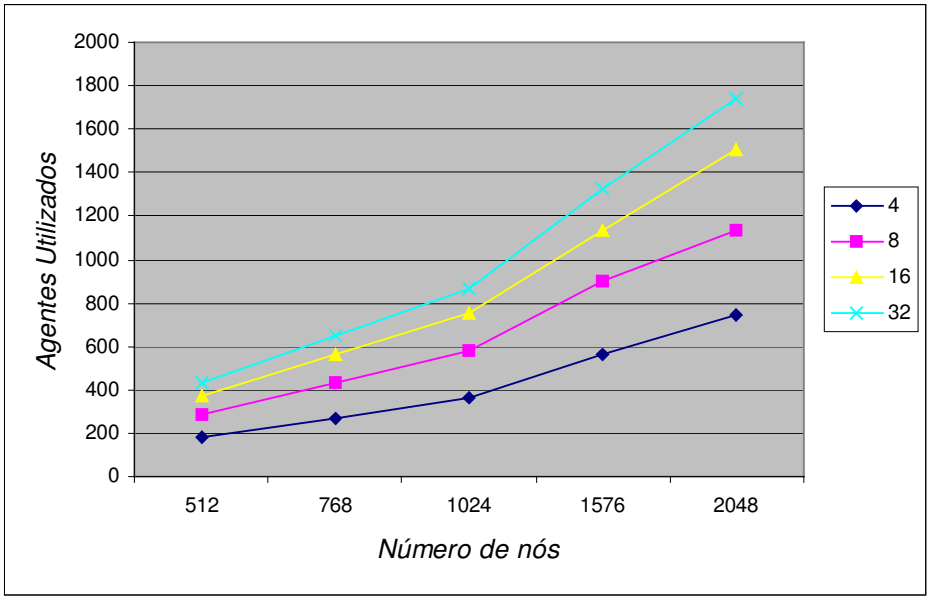

Figura 33 – Resultados obtidos pelo *Alg-D*.

O gráfico da Figura 33 apresenta o resultado obtido pela aplicação da solução aqui proposta. Para comparar os dados obtidos pelas propostas realizadas em FLOCCHINI (2005) e FLOCCHINI (2007), o menor número de agentes utilizado para

cada grau correspondente foi extraído de cada resultado. As Figuras 34 e 35 a seguir demonstram os valores extraídos de cada experimento.

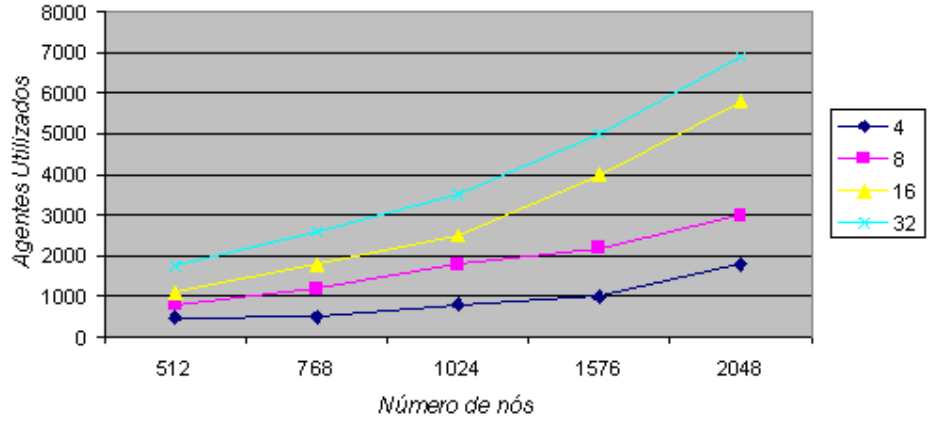

Figura 34 - Dados obtidos a partir dos resultados apresentados em FLOCCHINI (2005).

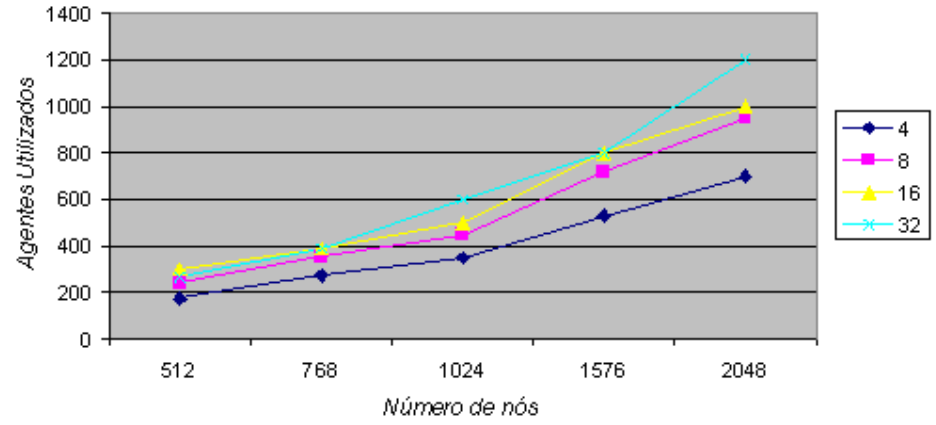

Figura 35 - Dados obtidos a partir dos resultados apresentados em FLOCCHINI (2007).

Podemos perceber, pela Figura 36, que os resultados obtidos pelo método aqui proposto encontram-se na mesma escala dos resultados obtidos pelo segundo experimento apresentado em FLOCCHINI *et al*. (2007). Neste trabalho, o método proposto é exatamente utilizar esta distância entre dois nós no grafo fazendo com que os mesmos sejam alcançados sem que haja um uso excessivo de agentes. Nos testes realizados com o *Alg-D*, a aplicação da heurística que aumenta mais os caminhos (*Alg-Stretcher*) foi realizada, e foi observado que, ao não aplicá-la à orientação obtida pelo *Alg-Arestas*, ocorreu uma utilização de, em média, 2% mais agentes. Pode-se notar uma evolução na forma de encarar a descontaminação de grafos, uma vez que o algoritmo aqui proposto depende de propriedades "reais", preocupação não observada na literatura utilizada.

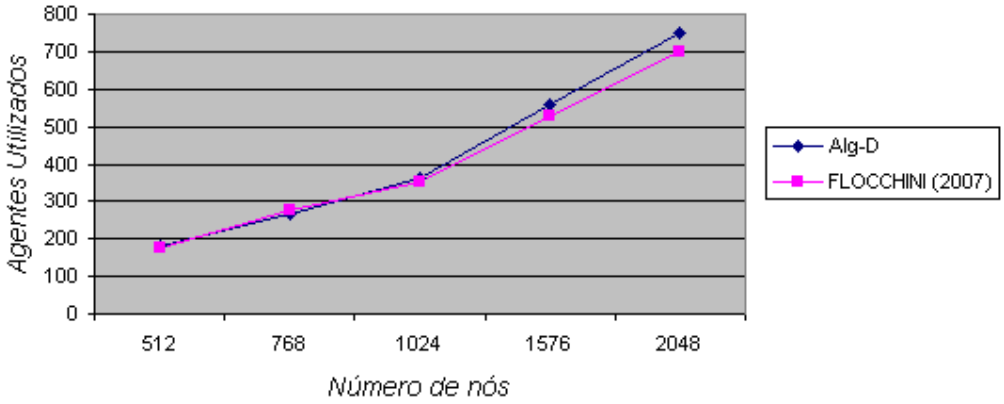

Figura 36 - Comparação dos dados obtidos para o *Alg-D* com grafos de grau 4.

4.1.2 Link Farms

Com o objetivo de demonstrar uma comparação quantitativa entre o algoritmo assíncrono proposto em LUCCIO *et al*. (2007) e o *Alg-D*, os dois algoritmos foram aplicados ao mesmo conjunto de grafos. A orientação acíclica inicial escolhida como ponto inicial para o *Alg-D* tenta reproduzir o mesmo cenário produzido pelo algoritmo em LUCCIO *et al*. (2007), isto é, AMs irão mover-se somente em *links* do círculo principal. Na Figura 37, é realizada uma demonstração de como o método proposto em LUCCIO *et al*. (2007) se comporta em um grafo circulante (neste caso o grafo circulante $C_{i,6}(L=\{1,2\})$.

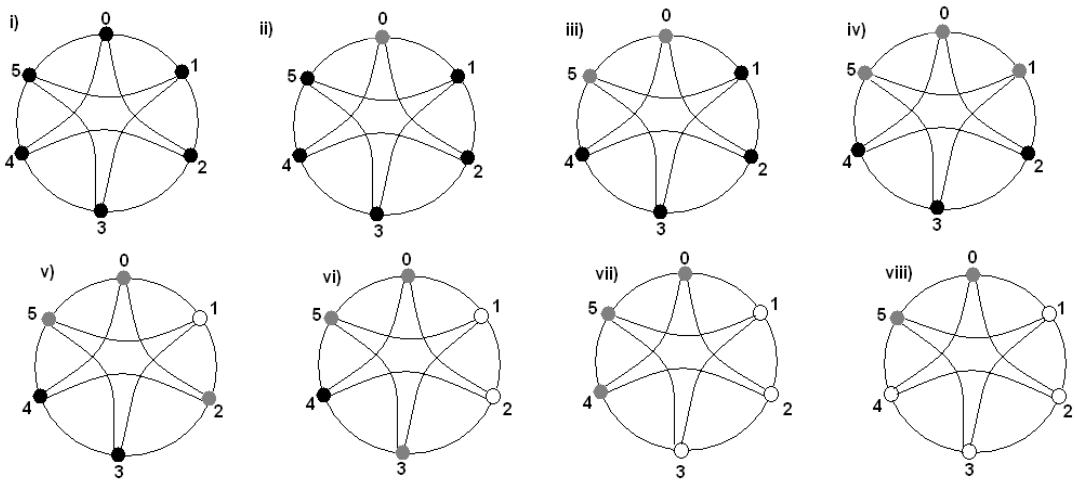

Figura 37 - Passo-a-passo do método de descontaminação proposto em LUCCIO *et al*. (2007).

Grafos circulantes $C_{i,n}(L)$ com $k = 2$ ($L = \{1,2\}$) e $k = 3$ ($L = \{1,2,3\}$) foram considerados. No caso de $k = 2$, *Alg-D* usou três AMs para realizar a descontaminação, enquanto o algoritmo em (LUCCIO *et al*., 2007) usou quatro AMs. No caso de $k = 3$, *Alg-D* usou quatro AMs e o algoritmo em (LUCCIO *et al*., 2007) cinco AMs (todos os testes foram feitos com $10 \leq n \leq 10{,}000$).

Em LUCCIO *et al*. (2007) foi concluído que o número de "saltos" dados pelos AMs podem ser contados como o tempo que a descontaminação leva para terminar. Então uma comparação do número de "saltos" necessários para a descontaminação foi feita. O número de saltos em LUCCIO *et al*. (2007) é $n - c + h$, onde n é o número de nós como visto anteriormente, c igual a $\lfloor (k + 1)/2 \rfloor$ e h o número de saltos necessários para colocar os primeiros AMs, e é representado como a seguir:

$$h = \begin{cases} 3(k^2/4 - k/2) & \text{para } k \text{ par} \\ 3(k^2/4 - k/2 + 1/4) & \text{para } k \text{ ímpar} \end{cases} \qquad \textbf{2)}$$

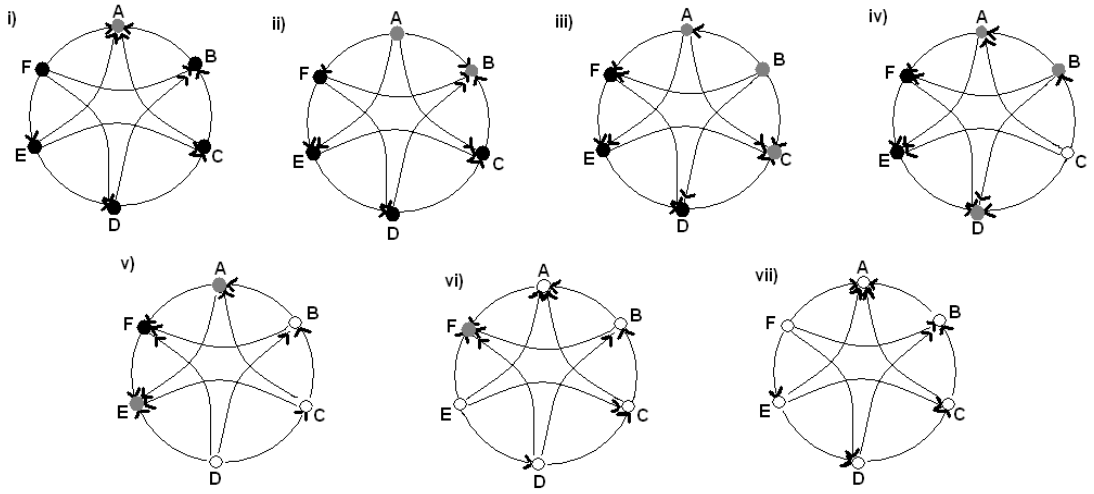

Figura 38 - Descontaminação de um grafo circulante $C_{i,6}(L=\{1,2\})$.

Na Figura 38, é mostrado o passo-a-passo do algoritmo *Alg-D* para um grafo circulante *C*i,6(L={1,2}). Observando o número de saltos, no caso $k = 2$, *Alg-D* e o algoritmo em LUCCIO *et al*. (2007) executaram o mesmo número de saltos, o que para o *Alg-D* foi constante em k, utilizando sempre $n - 1$ saltos. A partir de $k = 3$, *Alg-D* usou n - 1 para descontaminar enquanto o algoritmo de LUCCIO e PAGLI, os agentes

utilizaram $n + 1$ saltos. Isso se dá pelo fato de o algoritmo aqui proposto não precisar dos saltos para verificar a posição dos agentes como em LUCCIO *et al*. (2007).

4.2. Comentários do Capítulo

Neste capítulo os dados obtidos pela aplicação do algoritmo proposto neste trabalho nos diversos cenários foram apresentados. No caso das *link farms*, algumas alterações no algoritmo foram feitas para contemplar o formato dos grafos alvo, os Grafos Circulantes. Essas alterações não modificaram o comportamento principal do *Alg-D*, mostrando sua generalidade. No próximo capítulo, a análise final dos resultados alcançados neste trabalho será feita. Além disso, novos experimentos a serem realizados serão propostos.

5 Conclusão

5.1. Considerações Finais

Neste trabalho, a descontaminação de grafos foi associada ao compartilhamento de recursos através do mapeamento de problemas e suas restrições em grafos conexos. Partindo desta associação, um método assíncrono e distribuído para descontaminação desses grafos foi proposto. Esta nova abordagem foi introduzida com o objetivo de complementar propostas relacionadas e preencher as lacunas deixadas pelos métodos apresentados anteriormente na literatura para solucionar o problema da descontaminação de grafos. É importante salientar que a possibilidade do uso desta abstração para vários problemas já apresentados na literatura pode proporcionar um novo caminho para pesquisas. Foi argumentado que se deve encontrar uma aproximação para o número de agentes utilizados em uma descontaminação, combinando o conjunto de cliques maximais do grafo a ser descontaminado com o critério de contaminação para cada problema.

Neste trabalho também foi investigada a paralelização do algoritmo inicial com o objetivo de aplicá-lo a grafos de larga escala em um ambiente em que o grafo tenha sido capturado e esteja armazenado. Para aplicar esta variação do procedimento proposto de descontaminação, seria necessária identificar os nós do grafo G a serem limpos. Após a aplicação de uma metodologia distribuída para realizar a identificação dos nós, uma orientação acíclica pode ser induzida a partir desta identificação e descontaminação paralela pode ser aplicada uma vez tendo em mãos os recursos necessários para a execução paralela da descontaminação. Concluiu-se que, em tais casos, é necessário o emprego de algoritmos de memória externa em conjunto com a nossa metodologia.

Pode ser citado também que, neste trabalho, métodos de detecção de contaminação não são apresentados. Esses métodos dependem do problema a ser solucionado e, como indicado, cada agente deve conter a solução a ser aplicada localmente em cada nó ou componente do todo. A generalização aqui feita não leva em

consideração o tempo para realizar cada operação de um agente num determinado nó, uma vez que essas operações serão definidas pelo problema a ser solucionado.

5.2. Trabalhos Futuros

Um próximo passo a ser dado para dar continuidade a este trabalho seria uma melhoria no algoritmo que determina as orientações acíclicas iniciais, o *Alg-Stretcher*. Este método, como mostrado anteriormente, tem o objetivo de "alargar" os caminhos do grafo direcionados. Foi explicitado também, que a minimização da concorrência é um problema NP-Completo, o que nos indica que será necessário considerável esforço para encontrar uma melhor metodologia com o propósito de minimizar a concorrência em um grafo e, conseqüentemente, a utilização de menos agentes no processo de descontaminação.

Como mencionado neste trabalho (Seção 2.1.2), outros problemas podem ser posteriormente estudados a fim de extrair propriedades que remetam ao conceito de contaminação de grafos. Um breve estudo foi feito acerca das redes sociais, como *orkut* e *facebook*, que têm como característica principal as conexões entre pessoas (GRANOVETTER, 1973, ROGER *et al*., 1981, HALPIN *et al*., 2010). A partir dessas conexões, um grafo conexo pode também ser aferido, possibilitando a aplicação da abordagem proposta aqui.

Estudos mais elaborados devem ser realizados no intuito de traduzir a estrutura de cada uma dessas redes sociais e a forma com que uma contaminação, uma vez detectada, possa ser solucionada. Uma contaminação a ser citada seria o método de *spam* de mensagem de *marketing* ou difamação nessas redes realizado, na maioria das vezes, por perfis falsos, o que impossibilita uma medida de proteção.

Outra abordagem prática seria a implementação do método de descontaminação baseada em posicionamento global. Esta implementação poderia ser utilizada pelas equipes de exploração mencionadas na Seção 2.1.2.1 para comunicação e decisões sobre movimentação baseadas no posicionamento de cada componente da equipe. O método seria acoplado a aparelhos que possuíssem o receptor de sinal de GPS e faria uso dos atuais serviços abertos de GPS e de um servidor externo ou local.

Considerações acerca da dinamicidade da Internet também devem ser feitas, uma vez que a mesma está em constante alteração e novas páginas e dados são inseridos ou excluídos constantemente. Com isso, um grafo da web recentemente "detectado" pode estar sendo alterado e ao mesmo tempo descontaminado, permitindo que o processo de descontaminação, hoje estático, possa sofrer interferência (ALVES *et al*., 2011).

Referências Bibliográficas

ADLER, M. , MITZENMACHER, M., 2001, "Towards compressing Web graphs. In Proc. of the IEEE Data Compression Conference, pages 203.212.

ALVES, D., GONÇALVES, V. C. F., LIMA, P. V. M., MACULAN, N., FRANÇA, F. M. G., 2011, "G-DI: a Graph Decontamination Iterator for the Web". Extended abstract, ACM WebScience 2011(WebSci11).

ARANTES Jr., G. M., 2006, "Trilhas, Otimização de Concorrência e Inicialização Probabilística em Sistemas sob Reversão de Arestas". Tese (Doutorado em Engenharia de Sistemas e Computação) - Universidade Federal do Rio de Janeiro.

ARANTES Jr, G. M., FRANÇA, FELIPE M.G., MARTINHON, CARLOS A., 2009, "Randomized generation of acyclic orientations upon anonymous distributed systems". Journal of Parallel and Distributed Computin*G,* Vol.69, p. 239-246.

ASAKA, M., OKAZAWA, S., TAGUCHI A., GOTO, S., 1999, "A method of tracing intruders by use of mobile agents". In 9th Annual Conference of the Internet Society (INET'99).

BHARAT, KRISHNA, BRODER ANDREI, HENZINGER, MONIKA, KUMAR, PUNEET, VENKATASUBRAMANIAN, SURESH, 1998, "The Connectivity Server: Fast access to linkage information on the web". In Proceedings of the Seventh International World Wide Web Conference, pages 469.477, Brisbane, Australia.

BAKER, M., CARPENTER, B., SHAFI, A., 2006, "MPJ Express: Towards Thread Safe Java HPC". Submitted to the IEEE International Conference on Cluster Computing (Cluster 2006)

BARBOSA, V. C., GAFNI, E., 1989, "Concurrency in Heavily Loaded Neighborhood-Constrained Systems". ACM Transactions on Programming Languages and Systems, Vol. 11, Número 4, Pages 562-584.

BARBOSA, V. C., 1996, "An Introduction to Distributed Algorithms". Cambridge: The MIT Press, 365 p.

BARBOSA, V. C., 2000, "An Atlas of Edge-Reversal Dynamics". London: Chapman & Hall/CRC, 372 p.

BARRIÈRE, L., FLOCCHINI, P., FRAIGNAUD, P., SANTORO, N., 2002, "Capture of an intruder by mobile agents". In: Proc. 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA), Winnipeg, Canada.

BECCHETTI, L., CASTILLO, C., DONATO, D., LEONARDI, S., BAEZAYATES R., 2006, “Link-Based Characterization and Detection of Web Spam”. In: Proc. AIRWeb’06, Seattle.

BLIN, LÉLIA, FRAIGNIAUD, PIERRE, NISSE, NICOLAS, VIAL, RINE, 2006, “Distributed chasing of network intruders”, In Proc. 13th Colloquium on Structural Information and Communication Complexity (SIROCCO’06).

BOLDI, PAOLO, CODENOTTI, BRUNO, SANTINI, MASSIMO, VIGNA, SEBASTIANO, 2002, “Ubicrawler: A scalable fully distributed web crawler”. In Proc. AusWeb02. The Eighth Australian World Wide Web Conference.

BOLDI, PAOLO, VIGNA, SEBASTIANO, 2003a, “The WebGraph framework I: Compression techniques”. Technical Report 293-03, Universitá di Milano, Dipartimento di Scienze dell'Informazione.

BOLDI, PAOLO, VIGNA, SEBASTIANO, 2003b, “The WebGraph framework II: Codes for the World Wide Web”. Technical Report 294-03, Universitá di Milano, Dipartimento di Scienze dell'Informazione.

BOLDI, PAOLO, VIGNA, SEBASTIANO, 2004, “WebGraph: Things you thought you could not do with Java™”. In *Proc. of the 3rd International Conference on Principles and Practice of Programming in Java*, ACM International Conference Proceedings Series, pages 1–8, Las Vegas, Nevada, USA, 2004. Computer Science Press, Trinity College, Dublin, Ireland.

BORIE, R., TOVEY, C., KOENIG, S., 2009, “Algorithms and Complexity Results for Pursuit-Evasion Problem”. In: Proceedings of the 21st international Jont Conference on Artifical intelligence.

BORNEMANN, MARKUS, NIEUWPOORT , ROB V. VAN, KIELMANN, THILO, 2005, “MPJ/Ibis: A Flexible and Efficient Message Passing Platform for Java”, In Euro PVM/MPI 2005, volume 3666 of LNCS.

BRADSHAW, A., 1991, “The UK security and Fire Fighting Advanced Robot Project”, Advanced Robotic Initiatives in the UK, IEEE Colloquium, pp. 1/1-1/4.

BRAGA, R. R., YANG, Z., FRANÇA, F. M. *G.*, 2008, “IMPLEMENTING AN ARTIFICIAL CENTIPEDE CPG: Integrating appendicular and axial movements of the scolopendromorph centipede”. In International Conference on Bio-inspired Systems and Signal Processing(BIOSIGNALS), Funchal. Proceedings of. Setúbal : INSTICC Press, 2008. v. 2. p. 58-62.

BRODER, A., KUMAR, R., MAGHOUL, F., RAGHAVAN, P., RAJAGOPALAN, S., STATA, S., TOMKINS, A., WIENER, J., 2000, “Graph structure in the web”. Computer Networks, 33:309–320.

BURKE, R., 2008, “Fire Protection Systems and Response”. CRC Press Taylor & Francis Group.

CASSIA, RICARDO F., ALVES, VLADIMIR C., BERNARD, FEDERICO G.-D., FRANÇA, FELIPE M. G., 2009, "Synchronous-to-asynchronous conversion of cryptographic circuits". Journal of Circuits, Systems, and Computers, v. 18, p. 271-282.

CARPENTER, BRYAN, 1998, "mpiJava: A Java interface to MPI". In First UK Workshop on Java for High Performance Network Computing, Europar '98.

CARVALHO, D., PROTTI, F., GREGORIO, M., FRANÇA, F. M. G., 2005, "A novel distributed scheduling algorithm for resource sharin*G* under near-heavyload". Lecture Notes in Computer Science, v. 3544, p. 431-442.

CORMEN, THOMAS H., LEISERSON, CHARLES E., RIVEST, RONALD L., STEIN, CLIFFORD, 2001, "Introduction to Algorithms". Cambridge: The MIT Press, 984 p.

COUTINHO, F. ; OGASAWARA, E. ; OLIVEIRA, D. ; BRAGANHOLO, V. ; LIMA, A. A. B. ; DAVILA, A. M. R. ; MATTOSO, M. L. Q., 2010, "Data Parallelism in Bioinformatics Workflows Using Hydra". In: Emerging Computational Methods for the Life Sciences Workshop, 2010, Chicago, IL. Proc. of Emerging Computational Methods for the Life Sciences Workshop, 2010. p. 507-515.

DIJKSTRA, E. W., 1965, "Solution of a problem in concurrent program control".Communications of the ACM 8, 569.

DONATO, D., LEONARDI, S., MILLOZZI, S., TSAPARAS, P., 2005, "Mining the inner structure of the Web graph". In 8th International Workshop on the Web and Databases (WebDB 2005), Baltimore, Maryland.

FLOCCHINI, P., NAYAK, A., SCHULZ, A., 2005, "Cleaning an arbitrary network with mobile agents". Proc. 2nd Int. Conference on Distributed Computing & Internet Technology. LNCS 3816, 132-142.

FLOCCHINI, P., SANTORO, N., 2006 , "Distributed security algorithms by mobile agents". In Proc. 8th International Conference on Distributed Computing and Networking (ICDCN'06).

FLOCCHINI, P., NAYAK, A., SCHULZ, A., 2007, "Decontamination of Arbitrary Networks using a Team of Mobile Agents with Limited Visibility". Proc. of 6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007) .

FOUKIA, N., HULAAS, J. G., HARMS, J., 2001, "Intrusion Detection with Mobile Agents". In 11th Annual Conference of the Internet Society (INET '01).

FRANÇA, F. M. G., 1993, "Scheduling weightless systems with self-timed boolean networks".Workshop on Weightless Neural Networks, pp. 87 -92.

FRANÇA, F. M. G., FARIA, L., 1995, “Optimal mappin*G* of neighbourhood-constrained systems”. Lecture Notes in Computer Science, Lyon, França, v. 980, p. 165-170.

FRANÇA, F. M. G., MARTINHON, C. A. J., ARANTES Jr, G. M., FARIA, L., 2002, “A Multistart Approach to Near-optimal Concurrency Dynamics in Neighborhood-constrained Systems”. In: The Second International Conference on Optimization and Control with Applications (OCA2002).

FRANÇA, FELIPE M. G., ALVES, V.C., GRANJA, E. P., 2007, “Processo de Síntese e Aparelho para Temporização Assíncrona de Circuitos e Sistemas Digitais Multi-Fásicos”. Revista da Propriedade Industrial, 1904, INPI, PI 9703819-9.

GALLIAN, J. A., 2007, "Dynamic Survey DS6: Graph Labeling." *Electronic J. Combinatorics*, DS6, 1-58.

GAREY, M.R., JOHNSON, D.S., TARJAN, R.E., 1976, “The planar hamiltonian circuit problem is np-complete”. SIAM Journal on Computing, v. 5, pp. 704–414.

GONÇALVES, VANESSA C. F., FRANÇA, FELIPE M.G, MACULAN, N., LIMA, P. M. V., 2010, “SER-Based Web Graph Decontamination”. First Worskhop INCT WebScience, PUC-RIO.

GONÇALVES, VANESSA C. F., FRANÇA, FELIPE M.G, MACULAN, N., LIMA, P. M. V., 2010, “A distributed dynamics for Web Graph Decontamination”. 10th ISoLA, Heraklion, Creete.

GRANOVETTER, MARK., 1973, “The strength of weak ties”. American Journal of Sociology. Vol 78, n° 6, p. 1360-1380.

GUAN, X., YANG, Y., YOU, J., 2000, “POM - A Mobile Agent Security Model against Malicious Hosts”. Proceedings of the 4 th International Conference on High Performance Computing in the Asia-Pacific Region.

GYÖNGYI, Z., GARCIA-MOLINA, H., 2005, “Web Spam Taxonomy”. In: Proc. AIRWeb’05, China.

HALPIN, HARRY, LAVRENKO, VICTOR, 2010, “A Generative Model of Tagging, Search and Social Networks”. In: Proceedings of the WebSci10.

HELMER, G. G., WONG, J. S. K., HONAVAR, V., MILLER, L., 1998, “Intelligent agents for intrusion detection”. In IEEE Information Technology Conference, 121-124.

HIRAI, JUN, RAGHAVAN, SRIRAM, GARCIA-MOLINA, HECTOR, PAEPCKE, ANDREAS, 2000, “WebBase: A repository of web pages. In Proc. of WWW9, Amsterdam, The Netherlands.

JANSEN, W., MELL, P., KARYGIANNIS, T., MARKS D., 2000, “Mobile agents in intrusion detection and response”. In 12th Annual Canadian Information Technology Security Symposium.

KALRA, NIDHI, FERGUSON DAVID, STENTZ, ANTHONY, 2006, "Constrained Exploration for Studies in Multirobot Coordination", In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).

KOH, K. M., ROGERS, D. G.;, TEO, H. K., YAP, K. Y., 1980, "Graceful Graphs: Some Further Results and Problems." *Congr. Numer.* 29, 559-571.

LANGE, DANNY B., OSHIMA, MITSURU, 1999, “Seven Good Reasons for Mobile Agents”. Commun. ACM 42(3): 88-89.

LAPAUGH, A., 1993, “Recontamination does not help to search a graph”. Journal of the ACM, 40 (2), 224-245.

LENGERKE, O., CARVALHO, D., LIMA, P. M. V., DUTRA, M. S., MORA-CAMINO, F., FRANÇA, F. M. G., 2008a, “Controle distribuído de sistemas JOB SHOP usando escalonamento por reversão de arestas”. In: XIV Latin Ibero-American Congress on Operations Research (CLAIO 2008), Cartagena de Indias.

LENGERKE, O., DUTRA, MAX S., FRANÇA, FELIPE M.G., TAVERA, MAGDA J.M., 2008b, “Automated Guided Vehicles (AGV): Searchin*G* a Path in the Flexible Manufacturin*G* Systems”. Journal of Konbin, v. 8, p. 113-124.

LUCCIO, F., PAGLI, L., SANTORO, N., 2006, “Network Decontamination with Local Immunization”. APDCM, 110-118.

LUCCIO F., PAGLI L., SANTORO N., 2007**,** ”Network decontamination with local majority-based immunization”. International Journal of Foundations of Computer Science,vol. 18/3, p 453-474.

LUCCIO, F., PAGLI, L., 2007, “Web Marshals Fighting Curly Link Farm”. Proc. of FUN 2007, LNCS, 4475, pp. 240–248.

MAKINO, K., UNO, T., 2004, “New Algorithms for Enumerating All Maximal Cliques”. Proc. 9th Scand. Worksh. Algorithm Theory (SWAT 2004), pp. 260.272. Springer-Verlag, Lecture Notes in Computer Science 3111.

MPJ, 2010a, “MPJ Express”. Disponível em http://sourceforge.net/projects/mpjexpress/files/releases/ acessado em 27 de novembro de 2010.

MPJ, 2010b, disponível em http://parallel.vub.ac.be/documentation/mpi/ acessado em 5 de dezembro de 2010.

MPJ-DOCS, 2010, disponível em http://mpj-express.org/docs/javadocs/index.html acessado em 4 de dezembro de 2010.

MOSCARINI, MARINA; PETRESCHI, ROSSELLA; SZWARCFITER, JAYME L., 1998, “On node searching and starlike graphs”. Congr. Numerantium 131, 75-84.

PEREIRA, M. R., VARGAS, P. K. ; FRANÇA, F. M. G. ; CASTRO, M. C. S., DUTRA, I. C., 2003, “Applying Scheduling by Edge Reversal to Constraint Satisfaction Problems”. In: SBAC-PAD 2003 - Symposium on Computer Architecture and High Performance Computing, 2003, São Paulo. Proceedings of the Symposium on Computer Architecture and High Performance Computing. Los Alamitos, CA : IEEE Computer Society Press, 2003. v. 1. p. 134-141.

PINHO, A. C., SANTOS, A. A., FIGUEIREDO, D. R., FRANÇA, FELIPE M.G., 2009, “Two ID-Free Distributed Distance-2 Edge Coloring Algorithms for WSNs”. In: 8th International IFIP-TC Networkin*G* Conference (NETWORKIN*G* 2009), Aachen. LNCS. Berlin / Heidelber*G* : Springer, 2009. v. 5550. p. 919-930.

PROTTI, FÁBIO, FRANÇA, FELIPE M. G, SZWARCFITER, J. L., 1997, “On Computing All Maximal Cliques Distributedly”. IRREGULAR 1997, p 37-48.

PSN, 2011, “*PlayStation®Network*”. Disponível em http://us.playstation.com/ acessado em 24 de abril de 2011.

QUINTIERE, J. G., 2006, “Fundamentals of fire phenomena”. John Wiley and Sons.

RAGHAVAN, S., GARCIA-MOLINA, H., 2003, “Representing web graphs. In Proc. of the IEEE Intl. Conference on Data Engineering, 2003.

RANDALL, K., STATA, R., WICKREMESINGHE, R., WIENER J., 2001, “The LINK database: Fast access to graphs of the Web”. Research Report 175, Compaq Systems Research Center, Palo Alto, CA .

ROGERS, E. M., KINCAID, D. L., 1981, “Communication network; toward a new paradigm for research”.New York : Free Press.

SPAFFORD, E. H., ZAMBONI D., 2000, “Intrusion detection using autonomous agents”. Computer Networks, 34(4): 547-570, October 2000.

SZWARCFITER, J. L., 1986, “Grafos e Algoritmos Computacionais”, Segunda Edição, p.216.

WEB, 2010, “World Wilde Web”. Disponível em http://pt.wikipedia.org/wiki/World_Wide_Web acessado em 21 de novembro de 2010.

WEBGRAPH, 2010, “WebGraph”. Disponível em http://webgraph.dsi.unimi.it/ acessado em 17 de outubro de 2010.

WEBGRAPH-DATA, 2010, disponível em http://law.dsi.unimi.it/datasets.php acessado em 17 de outubro de 2010.

WEBGRAPH-DOCS, 2010, disponível em http://webgraph.dsi.unimi.it/docs/index.html acessado em 17 de outubro de 2010.

YANG, Z., FRANÇA, FELIPE M.G., 2003, “A generalised locomotion CP*G* architecture based on oscillatory building blocks”. Biological Cybernetics, Heidelberg,v.89,n.1,p.34-42.

Anexo A

Algoritmo de Calculo dos Lâmbdas – Versão Distribuída

```
PARA cada nó em G
        Limpar o buffer de msgs recebidas

        SE todas as arestas orientadas para si
           lambda = 0
        SENAO
           lambda = -1

FIM PARA

EQTO não alcançou terminação global (todos os nós têm seu lâmbda calculado)

        PARA cada no

        ehSumidouro <- todas as arestas orientadas para si

                SE ehSumidouro

                        SE lâmbda!= 0
                                maiorLambda = -1;

                                PARA cada vizinho
                                        SE  recebeu MSG deste canal
                                                Se msgRecebida.conteudo >  maiorLambda
                                                        maiorLambda = msgRecebida.conteudo

                                FIM PARA

                                lambda  =  maiorLambda + 1;
                        FIM SE

                        inverteArestas();
                        Envia MSG com seu índice e lambida para todos os vizinhos

                FIM SE

        FIM PARA
FIM EQTO
```

Anexo B

WebGraph - Instalação e Utilização

Neste trabalho, a IDE utilizada foi o Netbeans com o Java 5.0.

1) Para iniciar a utilização da plataforma *WebGraph* deve-se fazer o *download* da mesma no *site* WEBGRAPH (2010) na parte superior direita.

2) Após o *download*, os pacotes devem ser descompactados e colocados em um diretório de escolha do usuário.

Bibliotecas

Principal

- webgraph-**2.4.5**.jar

Dependências

- colt-**1.2.0**.jar
- dsiutils-**1.0.12**.jar
- fastutil-**6.0.0**.jar
- jakarta-commons-collections-**3.2.1**.jar
- jakarta-commons-configuration-**1.4**.jar
- jakarta-commons-io-**1.4**.jar
- jakarta-commons-lang-**2.3**.jar
- jsap-**2.1**.jar
- junit-**3.8.2**.jar
- log4j-**1.2.14**.jar
- sux4j-**2.1**.jar

Observação: os valores em negrito indicam as versões de cada biblioteca utilizada neste trabalho. Posteriormente, ao fazer o download a partir da página indicada, as versões podem ter sido modificadas.

3) Incluir o pacote webgraph-*v*.jar (*v* significando a versão atual disponível) e as dependências no seu *classpath* do projeto (verificar como fazê-lo na IDE escolhida).

4) Após a inclusão dos arquivos no *classpath* do seu ambiente de trabalho, deve-se baixar para teste, os arquivo(s) da base de dados que podem ser encontrados em WEBGRAPH-DATA(2010). Uma vez baixados, deve-se descompactar os mesmos e extrair os arquivos de extensao *.graph* e *.properties* para o diretório de trabalho. Estes arquivos devem ficar no diretório raiz do projeto.

5) Após estes passos, as instruções para usar a base de dados podem ser encontradas em WEBGRAPH-DOCS (2010).

Anexo C

Alg-D Paralelo (Pseudo-Código)

Entrada: Grafo G com *n* nós divididos em *p* intervalos

```
l <- ⌊n/p⌋
PARA cada p_i
intervalo[p_i][inicio] <- p_i l
SE p_i = p-1
    intervalo[p_i][fim] <- n-1
SENAO
    intervalo[p_i][fim] <- p_i+1(l -1)
FIM PARA

EQTO nósContaminados > 0
    PARA  cada nó no intervalo
        PARA cada vizinho
            SE nó vizinho esta em { intervalo[p_i][inicio] , intervalo[p_i][fim] }
              recuperaEstado():
            SENAO
              envia MSG para processo p_i detentor
        FIM PARA

        SE todos os vizinhos menores foram alcançados
            é sumidouro

        SE é sumidouro

            SE totalVizinhosLimpos/ totalVizinhos > percentualCritérioRecontaminacao
              nó limpo
            SENAO
              nó guardado

        FIM SE
    FIM PARA
    PARA  cada nó guardado { intervalo[p_i][inicio] , intervalo[p_i][fim] }

            totalVizinhosLimpos <- calculaVizinhoLimpos();
            SE totalVizinhosLimpos/ totalVizinhos > percentualCritérioRecontaminacao
                    nó limpo

    FIM PARA
    verificaMsgsRecebidas();
    nósContaminados <-  ∑p_i nósContaminados
FIM EQTO
```

Anexo D

MPJ - Instalação e utilização

1) Baixar de MPJ (2010a) a última versão disponível da biblioteca MPJ Express.

2) Descompactar e colocar em diretório de escolha do usuário. Colocar no *classpath* do projeto.

3) Utilizar o *javadoc* disponibilizado em MPJ-DOC (2010) para instruções de procedimentos utilizar os métodos da API.

4) Instruções de como executar o arquivo java gerado (*.jar*) podem se encontradas em MPJ (2010b). Neste projeto, o *middleware* Hydra (COUTINHO et. al, 2010) foi utilizado para encapsular a alocação de recursos para hospedagem dos processos no cluster disponibilizado pelo Núcleo de Atendimento a Computação de Alto Desempenho (NACAD/COPPE-UFRJ).