Universidade Federal
do Rio de Janeiro
Escola Politécnica

OBTENDO CONCORRÊNCIA MÍNIMA ATRAVÉS DE CICLOS MAXIMAIS SOB A DINÂMICA DE ESCALONAMENTO POR REVERSÃO DE ARESTAS

Carlos Eduardo Lopes Marciano

Projeto de Graduação apresentado ao Curso de Engenharia de Computação e Informação da Escola Politécnica da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Computação e Informação.

Orientadores: Felipe Maia Galvão França
Luidi Gelabert Simonetti

Rio de Janeiro
Março de 2019

OBTENDO CONCORRÊNCIA MÍNIMA ATRAVÉS DE CICLOS MAXIMAIS SOB A DINÂMICA DE ESCALONAMENTO POR REVERSÃO DE ARESTAS

Carlos Eduardo Lopes Marciano

PROJETO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE COMPUTAÇÃO E INFORMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE COMPUTAÇÃO E INFORMAÇÃO.

Examinadores:

Prof. Felipe Maia Galvão França, Ph.D.

Prof. Luidi Gelabert Simonetti, D.Sc.

Prof. Abilio Pereira de Lucena Filho, D.Sc.

Prof. Claudio Esperança, Ph.D.

RIO DE JANEIRO, RJ – BRASIL

MARÇO DE 2019

Marciano, Carlos Eduardo Lopes

Obtendo Concorrência Mínima através de Ciclos Maximais sob a dinâmica de Escalonamento por Reversão de Arestas/Carlos Eduardo Lopes Marciano. – Rio de Janeiro: UFRJ/POLI - COPPE, 2019.

XIII, 54 p.: il.; 29, 7cm.

Orientadores: Felipe Maia Galvão França

Luidi Gelabert Simonetti

Projeto (graduação) – UFRJ/ Escola Politécnica/ Curso de Engenharia de Computação e Informação, 2019.

Referências Bibliográficas: p. 50 – 53.

1. Escalonamento por Reversão de Arestas. 2. Grafos. 3. Concorrência Mínima. 4. Ciclos Maximais. I. França, Felipe Maia Galvão *et al.*. II. Universidade Federal do Rio de Janeiro, Escola Politécnica/ Curso de Engenharia de Computação e Informação. III. Título.

Aos meus quatro avós que, cada um do seu jeito, lutam diariamente pela vida de forma heroica.

Agradecimentos

Não considero estes agradecimentos como sendo referentes ao projeto final, mas sim à trajetória que culmina com a cerimônia de graduação. Portanto, as palavras aqui escritas não são protocolares, mas sim de uma grande sinceridade. Estes agradecimentos estão organizados em ordem cronológica, e todos aqui listados contribuíram, do seu jeito, para que o destino desta viagem tenha sido exatamente este.

Inicialmente, são inevitáveis os agradecimentos à família. Sem sua estabilidade e confiança, nada do que veio depois teria sido possível. Também irei tomar uma frase para agradecer aos amigos de infância – Jéssica, Thomas, Anthony, Hugo, Murillo e Jan, nesta ordem de cronologia – que deram um pouco de si para que minha infância e adolescência fossem recheadas de memórias inesquecíveis. Impossível também não agradecer à Iolanda e à Sebastiana que, cheias de simplicidade, cuidaram de mim com tanto amor desde pequeno; e à Melzinha, minha idosa canina que saiu das ruas e hoje alegra nossos dias incondicionalmente.

É uma grande honra agradecer também àqueles que entraram em minha vida pelo mais puro acaso dos jogos virtuais: ao Felipe Habitzreuter, Renan Sardim, Ricardo Andrade e aos demais membros da antiga *Edguild*, cujas amizades começaram em *Azeroth* e transbordaram para incríveis aventuras no mundo real; à Ester Habitzreuter, por ter acreditado no meu potencial antes mesmo de eu sequer ter algum; ao Juninho Correa e ao Rafael Pavanetti, pelo time de arena mais divertido de *Azeroth*; ao Caio Macedo e ao Gustavo Vargas, pela admiração constante sempre tão motivadora; e, finalmente, ao Deno, meu alter ego virtual cuja trajetória sempre se misturará à minha, enfrentando juntos as grandes adversidades que os mundos reais e virtuais colocarem em nosso caminho.

Mas, enquanto algumas pessoas participaram do início de minha jornada, outras foram essenciais à sua continuidade: ao Rodrigo Carvalho, por sempre estar

presente nos momentos mais necessários e ter me apresentado ao mundo da computação; à Catarina Ramos, por ter me apresentado ao Rodrigo Carvalho e pela simpatia infindável; ao Lucas Barcellos e ao João Victor Davim, cujos laços formados durante a faculdade se eternizarão em minha memória; ao Philipe Ferraz, Gabriel Rebello, Paulo Sanz e ao Felipe Assis, pelas grandes aventuras dentro e fora deste país Fundão; ao Rodrigo Pagliusi, Charles, Pedro Wagner e à Anna Bárbara, por agraciarem minhas idas ao Fundão com suas presenças como carona e impedirem que eu me atrasasse (ainda mais); e ao Paulo Valente, cuja ajuda foi chave para que eu conseguisse terminar meu trabalho de estatística.

E, claro, nada disso teria sido possível sem a orientação daqueles que já trilharam este caminho e hoje tanto contribuem para que outros também o trilhem: ao Heraldo Almeida, que me apresentou ao mundo de Algoritmos e orientou minha travessia por ele durante os dois anos seguintes; ao Vinícius Gusmão, cuja simpatia e sabedoria foram primordiais para construir minha paixão pela pesquisa; ao Felipe França, Luidi Simonetti e ao Abilio Lucena, por me orientarem tão amigavelmente em seus projetos e me salvarem de tantas enrascadas; e ao Daniel Figueiredo, pela confiança, empatia e gentileza, que se provaram essenciais para que eu continuasse sonhando novas aventuras.

Por fim, resta fugir da ordem cronológica e agradecer àqueles que, uma ou mais vezes, tornaram do cotidiano um lugar surpreendentemente agradável: ao Bruno Moneró, por me oferecer um estágio numa fantástica empresa durante um inesperado encontro no metrô; à Bia Prata e à Cláudia Prata, pela enorme simpatia que trouxeram à secretaria; à Rosa, do Burguesão, pelo carinho ilimitado ao anotar nossos pedidos; e ao funcionário do Grêmio da Coppe, responsável por anunciar as senhas, por sempre fazer brincadeiras ao pegar as bebidas (nunca descobri seu nome, mas pelo menos tiramos uma foto de recordação no último dia).

Ao terminar de escrever estes agradecimentos, é impossível não olhar pra trás e perceber a sorte que tive ao encontrar pessoas como estas. São memórias assim que tanto terei orgulho de carregar comigo nos muitos passos que ainda virão.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Computação e Informação.

OBTENDO CONCORRÊNCIA MÍNIMA ATRAVÉS DE CICLOS MAXIMAIS SOB A DINÂMICA DE ESCALONAMENTO POR REVERSÃO DE ARESTAS

Carlos Eduardo Lopes Marciano

Março/2019

Orientadores: Felipe Maia Galvão França
Luidi Gelabert Simonetti

Curso: Engenharia de Computação e Informação

Uma solução algorítmica efetiva para problemas de compartilhamento de recursos em sistemas de alta carga é o algoritmo denominado Escalonamento por Reversão de Arestas, essencialmente provendo algum nível de concorrência ao descrever uma ordem de *operação* para os nós de um grafo. A concorrência resultante é uma métrica difícil de ser otimizada, tendo em vista que os problemas de decisão associados com a obtenção de seu máximo e mínimo são provadamente NP-completos. Este projeto de graduação propõe uma nova técnica envolvendo ciclos maximais para a obtenção de concorrência mínima, sendo desejável para problemas associados com descontaminação em grafos e teoria musical. Experimentalmente, é mostrado que instâncias razoavelmente grandes do problema de concorrência mínima podem ser resolvidas à otimalidade provada sob tempos aceitáveis de CPU. Este trabalho também recorda diversos conceitos associados com Escalonamento por Reversão de Arestas, coletando algumas de suas aplicações propostas ao longo dos anos.

Abstract of the Undergraduate Project presented to Poli/COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Computer and Information Engineer.

OBTAINING MINIMUM CONCURRENCY VIA MAXIMUM CYCLES UNDER THE DYNAMICS OF SCHEDULING BY EDGE REVERSAL

Carlos Eduardo Lopes Marciano

March/2019

Advisors: Felipe Maia Galvão França
Luidi Gelabert Simonetti

Course: Computer and Information Engineering

An effective algorithmic solution for resource-sharing problems in heavily loaded systems is Scheduling by Edge Reversal, essentially providing some level of concurrency by describing an order of *operation* for nodes in a graph. The resulting concurrency is a hard metric to optimize, given that the decision problems associated with obtaining its *extrema* have been proved to be NP-complete. This undergradate project proposes a novel approach involving maximal cycles for attaining minimum concurrency, which is desired for problems associated with graph decontamination and music theory. Empirically, it is also shown that reasonably large instances may be solved to proven optimality under acceptable CPU times. This work also recalls many concepts associated with Scheduling by Edge Reversal, collecting some of its applications proposed throughout the years.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O compartilhamento de recursos é um desafio recorrente no mundo moderno. Desde sinais elétricos dividindo o mesmo barramento de dados até pedestres e carros compartilhando a mesma rua, os problemas resultantes da interação entre duas ou mais partes competindo por recursos são bastante conhecidos na literatura. Em especial, a área de Sistemas Operacionais nutre um interesse especial pelo assunto, tendo em vista que o escalonamento dos múltiplos processos a serem executados é chave para o desempenho dos sistemas operacionais modernos.

1.1 Motivação

Em 1965, Edsger Dijkstra propôs e resolveu um problema de sincronização que ele mesmo intitulou *O Jantar dos Filósofos* [1], que viria a se tornar um dos mais importantes modelos de experimentação para os diversos algoritmos de escalonamento que estariam por vir. Sua formulação é bastante simples: cinco filósofos estão sentados ao redor de uma mesa circular, cada um com um prato de macarrão tão deslizante que cada filósofo precisa de dois garfos para comê-lo. Entre cada par de pratos, encontra-se um único garfo, como ilustra a Figura 1.1.

Apesar da formulação clássica do problema permitir que os filósofos estejam pensando (sem intenção de comer), com fome (tentando obter dois garfos) ou comendo (possuindo os dois talheres), o presente trabalho possui um interesse diferente. Portanto, devemos reformular o problema para permitir somente dois estados: *"com fome"*, ou seja, apenas esperando a liberação dos recursos para que sua tarefa seja

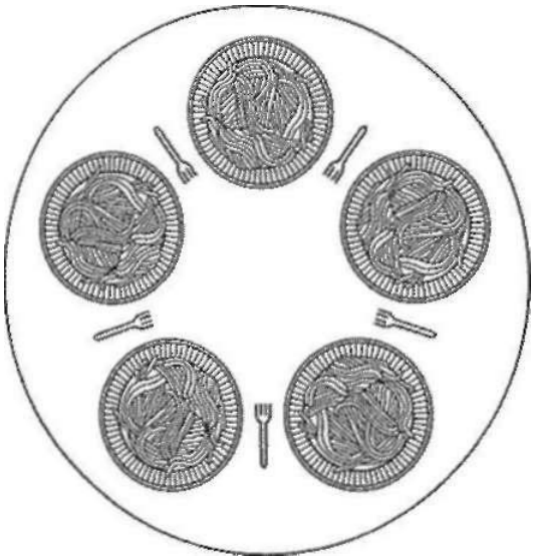


Figura 1.1: O Jantar dos Filósofos. Fonte: [1].

realizada; ou *"comendo"*, em pleno manuseio dos recursos e, por consequência, impedindo que seus vizinhos estejam também degustando o macarrão.

Esta ideia de vizinhança é justamente a inspiração para enxergar o *O Jantar dos Filósofos* sob a luz de Teoria dos Grafos. Com estes artifícios, poderemos propor soluções para alguns conflitos que eventualmente surgirão durante a dinâmica do jantar. Por exemplo, como garantir que cada filósofo tenha um tempo *justo* para degustar seu macarrão? Ou como tratar o caso em que dois filósofos pegam o mesmo garfo ao mesmo tempo, numa espécie de corrida? Ou como evitar que cada um dos filósofos pegue um único garfo e trave, por completo, a dinâmica do sistema?

Há pelo menos dois problemas oriundos desta dinâmica cuja análise deve ser comentada. O primeiro, conhecido como *deadlock*, ocorre quando todos os processos bloqueiam-se mutuamente devido à necessidade de acesso aos recursos compartilhados. O segundo problema decorrente da dinâmica chama-se *starvation*, em que os programas participantes continuam sua execução perpetuamente sem realizar um progresso. No exemplo dos filósofos, tais situações ocorrem caso cada filósofo pegue, simultaneamente, o garfo à sua esquerda, chegando a um estado conjunto de bloqueio devido à indisponibilidade do garfo direito.

No entanto, problemas de escalonamento normalmente tomam proporções consideravelmente maiores. Por exemplo, vamos considerar a pergunta: como podemos

alternar sinais de trânsito em uma avenida movimentada, como na Figura 1.2, a fim de garantir que a dinâmica de carros e pedestres não caia nos mesmos problemas que discutimos ao analisar a mesa de filósofos? Ou como escalonar centenas de processos competindo pelas mesmas regiões de memória em uma máquina, de forma que nenhum deles seja prejudicado?

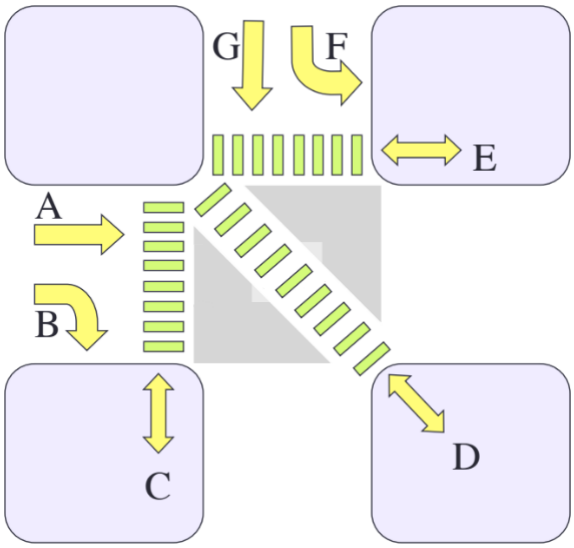


Figura 1.2: Um cruzamento de vias e seus fluxos. Fonte: [2].

No caso de sistemas com alta carga (isto é, em que nós estão constantemente demandando acesso aos seus recursos) e restringidos pela vizinhança, uma possível resposta para tais perguntas é o algoritmo conhecido por Escalonamento por Reversão de Arestas (*SER*, do inglês *Scheduling by Edge Reversal*). Apesar de sua análise detalhada ser apresentada somente no capítulo 2, sabemos que é possível expandir seu uso para problemas que, originalmente, não parecem possuir uma relação imediata com dinâmicas de escalonamento. Muitas destas aplicações serão detalhadas no capítulo 3, trazendo diferentes contextos para a utilização do algoritmo.

Por exemplo, um cenário derivado do problema de descontaminação de grafos em que as dinâmicas de Escalonamento por Reversão de Arestas podem ser aplicadas é o combate ao incêndio utilizando robôs autônomos. Nestas circunstâncias, cada nó é dito "contaminado" quando existe um foco de incêndio presente na região por ele representada. Um robô autônomo, ao chegar em um nó através de uma aresta, pode extinguir suas chamas. Caso um nó esteja "limpo" (ou seja, sem focos de incêndio) e sem robôs autônomos presentes, seu fogo permanecerá apagado somente

se seus vizinhos também estiverem no estado “limpo”. Caso contrário, as chamas retornarão. A Figura 1.3 ilustra esta situação em um apartamento, em que arestas entre nós codificam a possibilidade de recontaminação de um nó para outro.

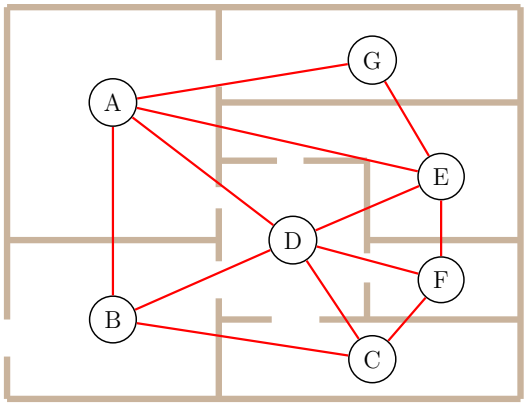


Figura 1.3: Planta de um apartamento, em que cada nó representa um foco de incêndio e arestas codificam meios de recontaminação. Adaptado de [3].

Devido ao alto custo dos robôs envolvidos no combate ao incêndio, surge a necessidade de minimizar o número de agentes atuando no grafo. Um número reduzido de robôs está diretamente relacionado a uma baixa concorrência da dinâmica de Escalonamento por Reversão de Arestas [7], cuja obtenção constitui-se como tema central do capítulo 4 deste trabalho. Através das técnicas aqui propostas, espera-se que muitos problemas relacionados com descontaminação de grafos possam usufruir de soluções mais eficientes em tempos viáveis de processamento.

Muitos problemas de decisão acerca do assunto de escalonamento de tarefas são intratáveis [8], dado que não conhecemos maneiras eficientes de resolvê-los à medida que o tamanho de suas entradas cresce. Ao longo deste trabalho, alguns problemas apresentados serão igualmente intratáveis e, por consequência, os algoritmos determinísticos que objetivamos propor para resolvê-los serão enumerativos. No entanto, ao combinarmos o uso de limites para a função a ser otimizada junto com a melhor solução encontrada até um dado momento, poderemos podar subconjuntos do espaço de busca, chegando a uma enumeração implícita e, consequentemente, obtendo soluções ótimas em um tempo aceitável para um dado subconjunto de instâncias do problema [9].

1.2 Objetivos

Antes de introduzir modelos de otimização e algoritmos para resolvê-los, este trabalho tem como objetivo reunir, de forma acessível, a literatura existente acerca das técnicas de Escalonamento por Reversão de Arestas, servindo como uma revisão bibliográfica. Dentre os tópicos abordados nesta etapa, destacam-se o funcionamento da dinâmica do SER; os algoritmos que geram orientações iniciais acíclicas; e os problemas práticos e teóricos aos quais a técnica é aplicável.

Subsequentemente, este trabalho possui como objetivo central apresentar um algoritmo determinístico para encontrar uma orientação inicial acíclica que, sob a dinâmica de SER, leve o sistema à sua concorrência mínima. Por fim, serão apresentados tempos empíricos de execução do algoritmo proposto para diversas instâncias relacionadas ao problema de descontaminação de grafos, assim como instâncias notoriamente difíceis provenientes de outros problemas relacionados. Ao final deste trabalho, espera-se que o método proposto tenha sido suficientemente elucidado, assim como sua corretude e viabilidade.

1.3 Organização do Projeto

Inicialmente, no capítulo 2, será apresentada a teoria já desenvolvida por outros autores acerca do *Escalonamento por Reversão de Arestas*, abordando as suas características, propriedades e definições julgadas mais relevantes. Tendo sido discutida a base referente ao algoritmo, o capítulo 3 apresenta uma série de aplicações práticas e teóricas nas quais o *Escalonamento por Reversão de Arestas* pode ser empregado, discutindo diferentes modificações aplicadas à técnica a fim de especializá-la para cada cenário. Após a conclusão dos capítulos de revisão bibliográfica, o capítulo 4 reúne a principal contribuição teórica deste trabalho, apresentando uma nova técnica para a obtenção de concorrências mínimas envolvendo a identificação de ciclos maximais, incluindo tempos de execução para a estratégia proposta. Por fim, o capítulo 5 extrai conclusões dos dados anteriormente apresentados e propõe novos rumos para a pesquisa envolvendo concorrência em dinâmicas de *Escalonamento por Reversão de Arestas*.

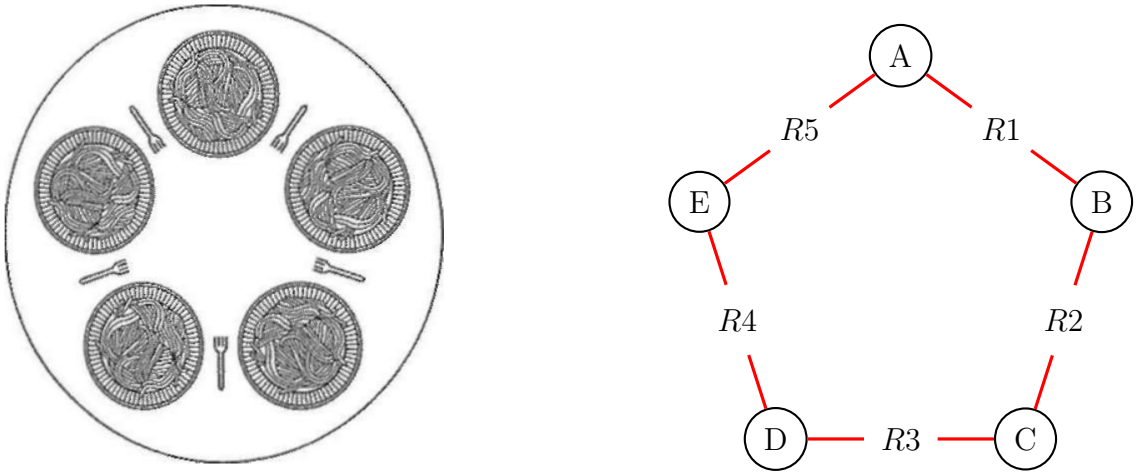
Capítulo 2

Escalonamento por Reversão de Arestas (SER)

Inspirada em uma proposta de 1981 para solucionar falhas de roteadores [10] e formalmente introduzida por Barbosa e Gafni em 1989 [11], a técnica de Escalonamento por Reversão de Arestas (*SER*, do inglês *Scheduling by Edge Reversal*) permite modelar sistemas cujos processos estão constantemente necessitando de todos os seus recursos associados para operar. Uma dinâmica como esta recebe o nome de "limitada pela vizinhança" (do inglês *neighborhood-constrained*), capturando a ideia de que quaisquer dois processos que compartilhem os mesmos recursos jamais poderão operar simultaneamente [12]. Neste capítulo, serão discutidas as modelagens necessárias anteriores à aplicação do algoritmo *SER*, juntamente com as propriedades conhecidas e previamente exploradas por outros autores acerca desta técnica de escalonamento. O objetivo, portanto, consiste em abordar os conceitos necessários para o entendimento dos capítulos seguintes, a fim de fornecer base para as discussões posteriores.

2.1 Construindo um Grafo de Recursos

Antes dos conceitos referentes ao *SER* e suas propriedades serem expandidos, é necessário estabelecer uma ponte entre os problemas do mundo real e as abstrações utilizadas. Em especial, o *SER* opera em um grafo acíclico direcionado (*DAG*, do inglês *Directed Acyclic Graph*), mas são necessários passos intermediários antes de



(a) A mesa original.

(b) Grafo de Recursos.

Figura 2.1: Modelando o *Jantar dos Filósofos*.

obtermos um DAG correspondente às relações do sistema. Assim, precisaremos inicialmente construir um grafo não-direcionado, que apenas capture corretamente as relações entre os diferentes processos e seus recursos, sejam eles pedestres e carros compartilhando uma via ou sinais elétricos dividindo um barramento. No entanto, neste primeiro momento, não iremos analisar nenhuma destas duas situações, mas sim revisitar um problema discutido na seção 1.1: o *Jantar dos Filósofos*.

Considere novamente a mesa com cinco filósofos e cinco garfos, em que cada filósofo apenas consegue comer seu macarrão caso obtenha os dois talheres mais próximos (Figura 2.1a). A técnica de modelagem consiste em identificar quais elementos do sistema possuem o papel de recursos e quais outros agem como processos. Em especial, para o *Jantar dos Filósofos*, é bastante razoável que os talheres representem os recursos sendo utilizados e os filósofos correspondam aos processos, que apenas operam (comem macarrão) ao obter seus respectivos recursos (talheres mais próximos).

Tendo em mãos estas atribuições, podemos criar um grafo simples não-direcionado $G = (V, E)$ em que cada vértice $v \in V$ representará um processo. Por sua vez, uma aresta $e_{ij} \in E$ entre os nós v_i e v_j existirá apenas caso tais vértices compartilhem algum recurso. O resultado, para o *Jantar dos Filósofos*, será um grafo simples no formato de anel com 5 nós representando os filósofos e 5 arestas correspondendo a cada talher (Figura 2.1b).

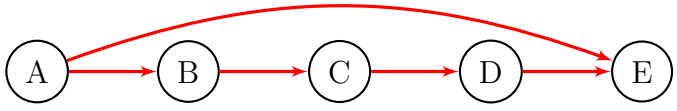
2.2 Gerando uma Orientação Acíclica Inicial

Antes de ser possível aplicar o processo de escalonamento por reversão de arestas, é necessário orientar o grafo de recursos G previamente construído de modo que não haja formação de ciclos. Como veremos a seguir, esta orientação inicialmente proposta irá impactar diretamente na *concorrência* do sistema que, apesar de ainda não ter sido formalmente definida, pode ser entendida como a média do número de vezes que cada nó irá operar em um dado intervalo síncrono de tempo [11]. Tendo a orientação acíclica inicial em mãos, será possível aplicar o *SER*.

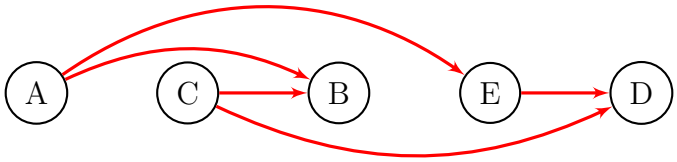
2.2.1 Ordenação Topológica

Uma técnica distribuída e extremamente simples para gerar esta primeira orientação acíclica consiste em enumerar cada nó de G com identificadores distintos e, em seguida, orientar G de modo que suas arestas sempre saiam do menor identificador e cheguem no maior. Com isto, teremos uma *ordenação topológica* de G, que apresenta a propriedade de não possuir ciclos [13]. O grafo resultante será um DAG e, por consequência, uma entrada válida para o algoritmo *SER*.

Com n identificadores, é necessário enfatizar as $n!$ maneiras de se ordenar os vértices de um grafo não-direcionado (podemos atribuir um identificador único n para o primeiro vértice; restarão $n-1$ para o segundo, $n-2$ para o terceiro e assim por diante). Mesmo que algumas ordenações produzam o mesmo DAG, ainda há



(a) Ordenação com A=1, B=2, C=3, D=4, E=5.



(b) Ordenação com A=1, B=3, C=2, D=5, E=4.

Figura 2.2: Duas ordenações topológicas para o *Jantar dos Filósofos*.

um número exponencial de ordenações possíveis ao utilizarmos esta técnica. Para o problema do *Jantar dos Filósofos*, podemos observar duas delas na Figura 2.2.

Tal técnica, no entanto, não leva em conta possíveis heurísticas para obter uma orientação inicial que resulte numa concorrência elevada do sistema. De fato, como veremos adiante, ao executarmos o *SER* com a orientação ilustrada na Figura 2.2b, iremos obter uma concorrência superior à da Figura 2.2a. Portanto, desde esta etapa, o questionamento acerca das orientações iniciais capazes de resultar nas concorrências máximas e mínimas de um dado sistema começa a ser levantado. Apesar de ambos os problemas terem sido previamente provados como sendo NP-completos [11, 14, 15], o capítulo 4 apresentará técnicas de otimização combinatória para viabilizar a minimização da métrica de concorrência.

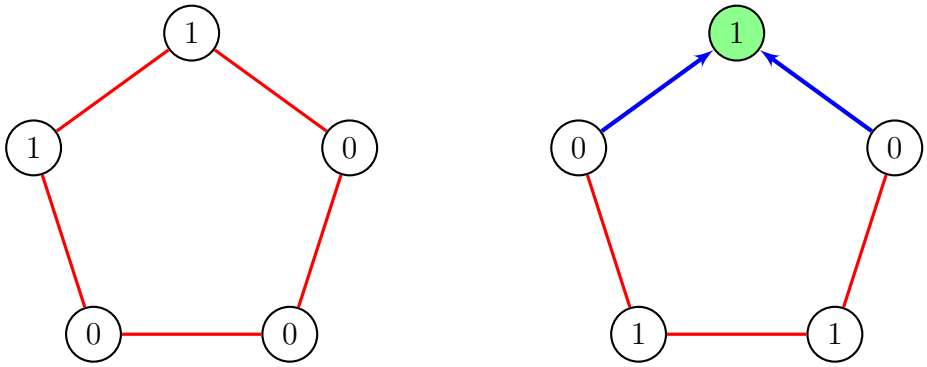
2.2.2 Algoritmos Calabrese-França e Alg-Viz

Outro método distribuído para a geração de orientações acíclicas iniciais é o algoritmo Calabrese-França [16], cuja execução ocorre em rodadas síncronas. A cada rodada, os vértices que ainda precisam orientar suas arestas (denominados "nós probabilísticos") sorteiam 0 ou 1, simulando a jogada de uma moeda. Se um certo nó sortear 1 e todos os seus vizinhos probabilísticos sortearem 0, este nó ganhador orientará todas as suas arestas ainda não-direcionadas em sua direção. O algoritmo termina, portanto, quando todas as arestas do grafo tiverem sido orientadas.

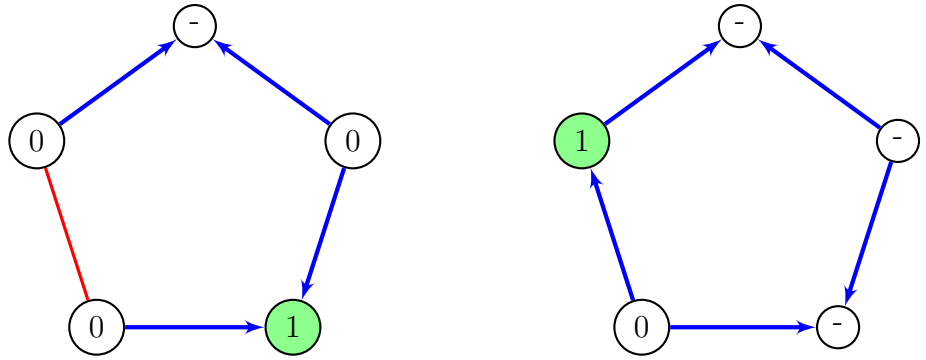
A Figura 2.3 ilustra uma das possíveis execuções do Calabrese-França para o *Jantar dos Filósofos*. Logo de imediato, percebemos que podem existir rodadas, como na da Figura 2.3a, em que nenhum nó sairá vencedor. De fato, pode ser necessário um longo tempo de convergência caso a probabilidade de se obter 0 ou 1 seja 1/2. Assim, sendo $viz(n_i)$ o número de vizinhos probabilísticos do nó n_i, convém polarizar a "moeda" da seguinte forma:

$$P\{moeda_i = 1\} = 1/(viz(n_i) + 1)$$
$$P\{moeda_i = 0\} = 1 - P\{moeda_i = 1\}$$

Ao viciarmos a "moeda", é possível obter uma complexidade média de convergência $O(n)$ para o caso de grafos completos [17]. Há ainda uma outra variante



(a) 1º sorteio: nenhum ganhador. (b) 2º sorteio: um ganhador.



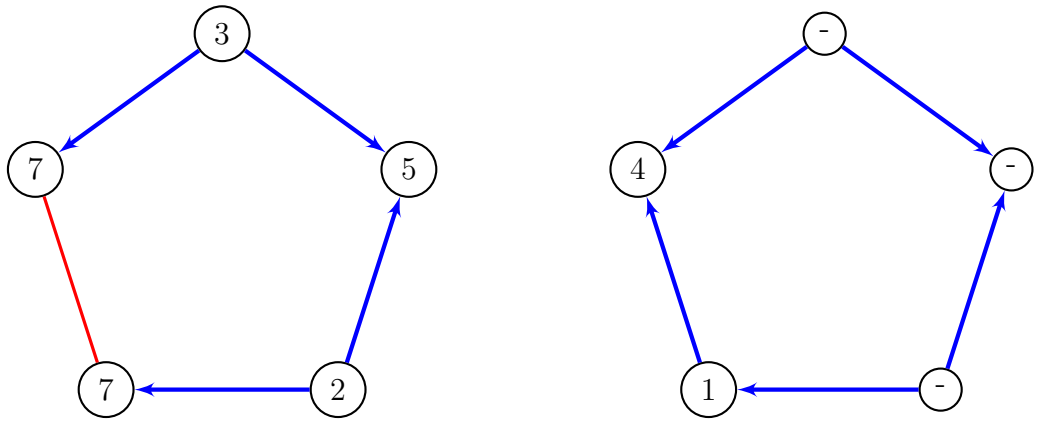
(c) 3º sorteio: um ganhador. (d) 4º sorteio: um ganhador.

Figura 2.3: Execução do Algoritmo Calabrese-França para o *Jantar dos Filósofos*.

do Calabrese-França, denominada Alg-Viz, que se propõe a utilizar uma faixa de números inteiros para a realização do sorteio. A convergência, no entanto, mantém-se $O(n)$ mesmo com a utilização de mais valores além de 0 e 1 [16].

2.2.3 Algoritmo Alg-Arestas

A fim de diminuir o número de passos necessários até a geração de uma orientação acíclica inicial, é possível empregar outra técnica que adota uma filosofia diferente. Denominado Alg-Arestas [16], este algoritmo propõe que cada nó realize um sorteio individualmente contra seus vizinhos pela orientação de cada uma de suas arestas. Assim, se um nó v_i sortear um número maior que seu vizinho v_j, a aresta e_{ij} será orientada em direção a v_i independente do resultado dos demais vizinhos de v_i. Uma execução do algoritmo, com um número de "faces do dado" $f = 10$, pode ser observada na Figura 2.4.



(a) 1º sorteio: 4 arestas orientadas.

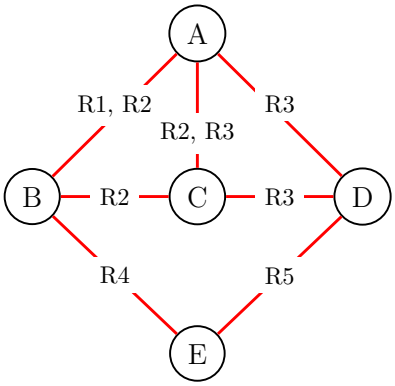
(b) 2º sorteio: aresta final orientada.

Figura 2.4: Execução do Alg-Arestas, com $f = 10$, para o *Jantar dos Filósofos*.

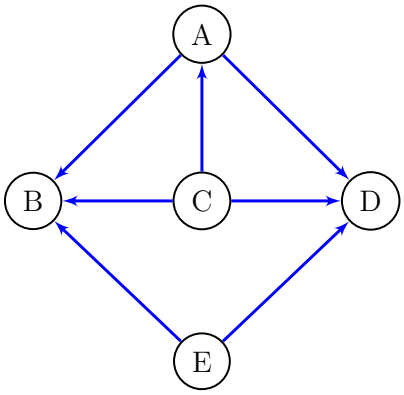
Além do Alg-Arestas nunca formar ciclos, o tempo necessário para convergência é, em média, bem menor. Se considerarmos um dado de f faces e um grafo de m arestas, em que $m \gg f$, podemos imaginar que $1/f$ das arestas a serem orientadas sofrerão um empate (quando dois nós vizinho sorteiam o mesmo número). Neste caso, a próxima iteração do algoritmo trabalhará com $1/f$ arestas, das quais, novamente, $1/f$ não poderão ser orientadas devido a empates. Assim, se considerarmos que este processo continue a se repetir, podemos imaginar que a convergência do algoritmo varie perto de $\lceil log_f m \rceil$ passos. Apesar desta análise não ser precisa e do resultado exato ser diferente, ela se aproxima bastante da realidade e é suficiente para averiguarmos que o Alg-Arestas de fato possui um tempo de convergência mais desejável quando comparado com o Alg-Viz ou com o Calabrese-França [16].

2.3 Execução do *SER*

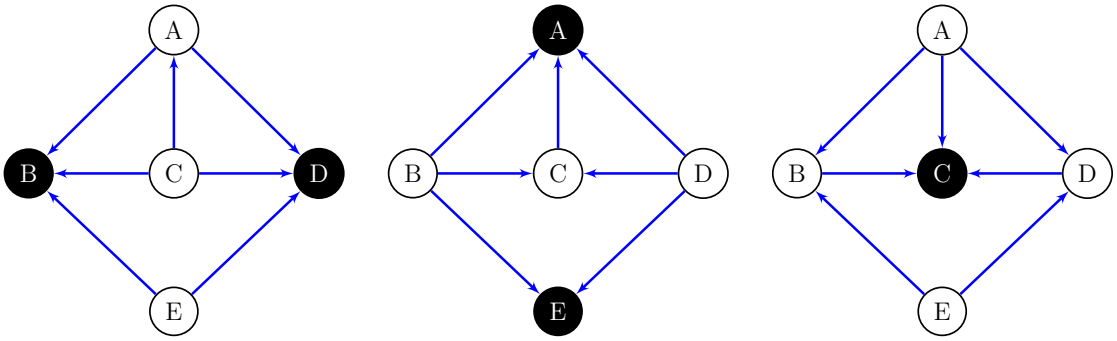
Tendo ocorrido a modelagem do problema através de um grafo de recursos e a geração de uma respectiva orientação acíclica inicial, é possível aplicar o algoritmo de Escalonamento por Reversão de Arestas a fim de estabelecer uma ordem para que cada vértice "opere", isto é, consuma seus recursos e realize suas tarefas. Ao final do algoritmo, será obtido um escalonamento periódico que solucione os problemas discutidos na seção 1.1, como *deadlocks* e *starvation* [11].



(a) Um grafo de recursos. Nós compartilhando recursos são conectados por uma aresta.



(b) Uma dada orientação acíclica aplicada ao grafo de recursos apresentado em (a).



(c) Um escalonamento (da esquerda para a direita) tomando (b) como orientação inicial. Sumidouros (em preto) podem operar, revertendo suas arestas subsequentemente para que novos sumidouros sejam formados. Após o nó C operar, a orientação mais à esquerda será repetida, levando o sistema a uma dinâmica periódica.

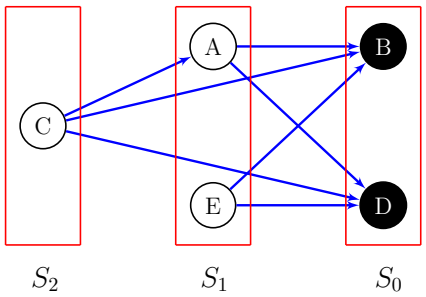
Figura 2.5: Execução do algoritmo de Escalonamento por Reversão de Arestas.

Devido à restrição de que a orientação inicial seja acíclica, haverá necessariamente pelo menos um sumidouro (nó cujo grau de saída é zero) no grafo, ao qual será concedida a permissão de operar. Todos os sumidouros de uma dada orientação operam de forma síncrona, revertendo suas arestas ao final da utilização de seus recursos. Este procedimento irá gerar um novo DAG com novos sumidouros que, por sua vez, poderão realizar suas operações. Eventualmente, o sistema entrará em um comportamento cíclico, em que cada nó opera o mesmo número de vezes dentro de um período. A Figura 2.5c ilustra um possível escalonamento para os vértices do grafo apresentado na Figura 2.5a.

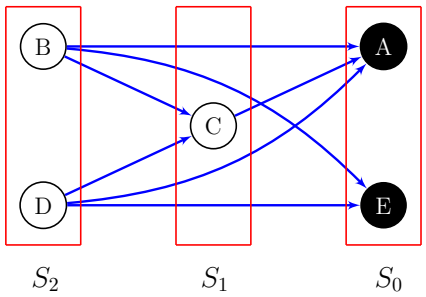
2.3.1 Decomposição em Sumidouros

É possível ainda realizar uma *decomposição em sumidouros* (do inglês, *sink decomposition*), possibilitando visualizar uma ordenação parcial de operação dos nós de G. Para tal, cada nó será atribuído a um conjunto $S_0, ..., S_{\lambda-1}$, em que o conjunto S_k irá conter todos os nós cujo caminho máximo até um sumidouro seja composto por k arestas. Ao final, serão obtidos λ conjuntos e, a cada iteração do *SER*, uma nova configuração de conjuntos será formada [11]. A Figura 2.6 ilustra este procedimento para o período apresentado na Figura 2.5c.

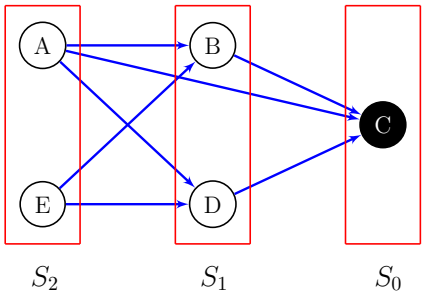
De imediato, podemos notar que, por definição, todos os sumidouros de uma orientação serão sempre designados ao conjunto S_0. Além disso, todos os nós em um conjunto S_k possuem pelo menos um vizinho em S_{k-1}, para $1 \leq k \leq \lambda - 1$. Por fim, dada uma orientação qualquer pertencente a um certo período, é possível dizer de imediato se os nós escalonados irão operar somente uma vez por período (como é o caso do escalonamento da Figura 2.5c). Isto ocorre se e somente se cada



(a) Primeira orientação.



(b) Segunda orientação.



(c) Terceira orientação.

Figura 2.6: Decomposição em sumidouros do período apresentado na Figura 2.5c. Cada orientação que compõe o período possui uma decomposição em sumidouros diferente.

nó em S_0 (o conjunto dos sumidouros) possuir pelo menos um vizinho em $S_{\lambda-1}$ (o conjunto com os nós de maior distância a um sumidouro) [11]. Verifica-se que, nas três decomposições apresentadas na Figura 2.6, isto de fato é verdade.

2.4 Definições Teóricas

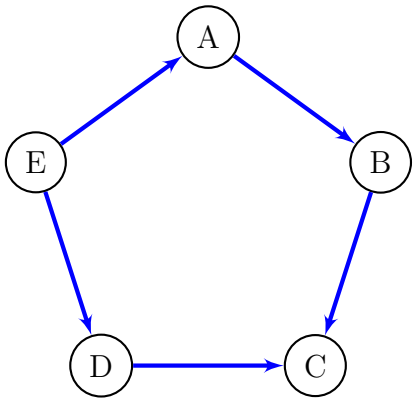
Nesta seção, serão abordadas definições necessárias para o entendimento do restante deste trabalho. Em todos os casos, supõe-se que o grafo $G = (V, E)$ é conectado e não-direcionado, com $|V| \geq |E|$ (*i.e.* G não é uma árvore).

2.4.1 Ciclos e Percursos

Seja κ um ciclo simples de G, isto é, um conjunto de vértices que formam a sequência de $|\kappa| + 1$ vértices $i_0, i_1, ..., i_{|\kappa|-1}, i_0$. Se κ é percorrido de i_0 a $i_{|\kappa|-1}$, dizemos que κ é percorrido no sentido horário. Do contrário, dizemos que κ é percorrido no sentido anti-horário. O conjunto contendo todos os ciclos simples de G é denominado K.

2.4.2 Orientações Acíclicas

Formalmente, uma orientação acíclica de G é uma função expressa como $\omega : E \to V$ tal que nenhum ciclo não-direcionado κ da forma $i_0, i_1, ..., i_{|\kappa|-1}, i_0$ existe para o qual $\omega(i_0, i_1) = i_1$, $\omega(i_1, i_2) = i_2$, ..., $\omega(i_{|\kappa|-1}, i_0) = i_0$. Portanto, a função ω recebe uma



$$\omega(A, B) = B$$
$$\omega(B, C) = C$$
$$\omega(C, D) = C$$
$$\omega(D, E) = D$$
$$\omega(E, A) = A$$

Figura 2.7: Uma orientação acíclica com seus respectivos valores para a função ω.

aresta de E e retorna o vértice ao qual esta aresta está direcionada. A Figura 2.7 ilustra os valores de ω para uma dada orientação acíclica.

Adicionalmente, dado um ciclo κ e uma orientação acíclica ω, podemos definir $n_{cw}(\kappa, \omega)$ como o número de arestas orientadas no sentido horário por ω em κ. De forma análoga, $n_{ccw}(\kappa, \omega)$ corresponde ao número de arestas orientadas em κ no sentido anti-horário.

2.4.3 Concorrência

Tendo em vista que o número de orientações acíclicas possíveis para a inicialização do *SER* é exponencial à medida que o número de arestas de G aumenta, é natural se perguntar quais orientações levam a períodos mais curtos, em que mais vértices operam concorrentemente, e quais levam a períodos mais longos. Porém, a fim de ser possível comparar duas ou mais orientações acerca de sua concorrência, é necessário estabelecer uma métrica que traduza esta ideia. Nesta seção, serão abordadas duas elaborações distintas que levam a fórmulas equivalentes, sempre produzindo o mesmo valor. Ambas as definições são da forma $\gamma : \Omega \to \mathbb{R}$, recebendo uma orientação acíclica ω como parâmetro e retornando um número real.

2.4.3.1 Definição Dinâmica

A primeira métrica de concorrência leva em conta as diversas orientações que compõem o período a ser analisado. Sendo p o comprimento de um período $\alpha_0, ..., \alpha_{p-1}$, definimos m como sendo o número de vezes que um nó opera dentro deste período (m é igual para todos os nós). Com isso, definimos γ, a concorrência do sistema, como:

$$\gamma(\omega) = \frac{m}{p} \tag{2.1}$$

O tempo computacional para o cálculo da concorrência através desta definição dinâmica é linear com o comprimento do período p. Ao realizarmos as reversões de arestas inerentes à operação do *SER*, basta contarmos o número de vezes que um certo nó opera durante um período, tornando esta estratégia extremamente viável. Em breve, veremos que este não é o caso para a definição estática.

2.4.3.2 Definição Estática

Outra fórmula para o cálculo da métrica de concorrência pode ser obtida estaticamente, isto é, sem a necessidade de execução do algoritmo *SER*. No entanto, devido à sua natureza enumerativa, seu uso restringe-se a um ambiente teórico. Ainda assim, esta é possivelmente a fórmula mais relevante para este trabalho, da qual serão derivadas, no capítulo 4, as conclusões acerca de como minimizar a métrica de concorrência.

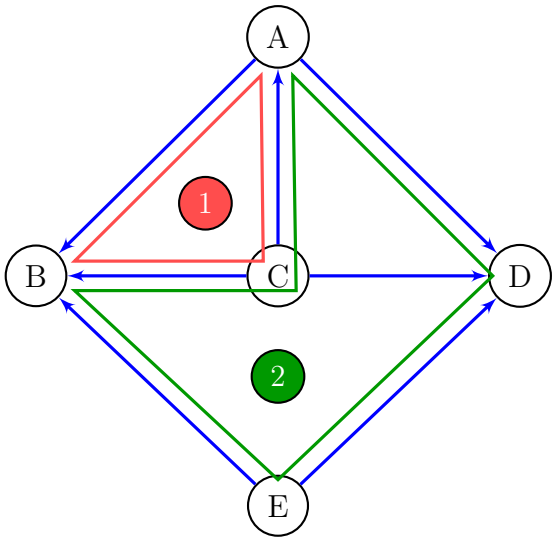
Como definido anteriormente nas subseções 2.4.1 e 2.4.2, seja K o conjunto de todos os ciclos simples de G. Seja ω uma orientação acíclica de G. Dado um ciclo $\kappa \in K$, sejam $n_{cw}(\kappa, \omega)$ e $n_{ccw}(\kappa, \omega)$ o número de arestas orientadas no sentido horário por ω em κ e no sentido anti-horário, respectivamente. Temos, então, uma definição estática para $\gamma(\omega)$, a métrica de concorrência:

$$\gamma(\omega) = \min_{\kappa \in K} \left\{ \frac{min\left\{n_{cw}(\kappa, \omega), n_{ccw}(\kappa, \omega)\right\}}{|\kappa|} \right\} \tag{2.2}$$

Em outras palavras, dada uma orientação ω, devemos checar todos os ciclos simples não-direcionados κ de G e calcular o número de arestas orientadas no sentido horário, assim como as arestas orientadas no sentido anti-horário. Então, extraímos o mínimo entre estes dois valores e dividimos o resultado pelo tamanho $|\kappa|$ do ciclo não-direcionado. O ciclo $\kappa \in K$ que retornar o menor valor irá ditar a concorrência do sistema.

A Figura 2.8 ilustra uma aplicação deste procedimento. Por questões de espaço, apenas dois ciclos simples de G são mostrados. Se levarmos em conta os demais ciclos simples além dos dois assinalados, será possível notar que o ciclo 1 (em vermelho) de fato produz o menor valor de fração para todo $\kappa \in K$. Por exemplo, o ciclo C, D, E, B, C retornaria o valor $1/2$ que, por sua vez, é maior que $1/3$ e não limita a concorrência global. Um outro ciclo que também produz o valor $1/3$ é o ciclo A, D, C, A.

De todo modo, ao minimizarmos a expressão 2.2 para todo $\kappa \in K$, encontraremos uma concorrência de $\gamma(\omega) = 1/3$. Este valor é coerente com a definição dinâmica apresentada na subseção 2.4.3.1: se analisarmos a Figura 2.5c, que ilustra o período



Pra o ciclo 1 (vermelho), temos:

$$n_{cw} = 1,\ n_{ccw} = 2 \text{ e } |\kappa| = 3$$

$$\frac{min\{1,2\}}{3} = \frac{1}{3}$$

Pra o ciclo 2 (verde), temos:

$$n_{cw} = 3,\ n_{ccw} = 2 \text{ e } |\kappa| = 5$$

$$\frac{min\{3,2\}}{5} = \frac{2}{5}$$

Figura 2.8: Exemplo do cálculo da métrica de concorrência $\gamma(\omega)$ utilizando a definição estática.

induzido pela orientação da Figura 2.8, percebemos que cada vértice opera 1 única vez em um período de comprimento 3. Assim, $m = 1$, $p = 3$ e $\gamma(\omega) = 1/3$.

Por fim, como discutido previamente, esta definição estática possui o mesmo domínio, mesma imagem e a mesma regra de correspondência da definição dinâmica apresentada na subseção 2.4.3.1. No entanto, seu caráter enumerativo dificulta seu uso na prática, devido à necessidade de identificar todos os ciclos simples de G. Sua importância é, portanto, teórica.

2.4.3.3 NP-Completude

Devido ao fato de que cada orientação acíclica ω aplicada a G leva a valores possivelmente diferentes de $\gamma(\omega)$, é inevitável levantarmos algumas questões: qual o valor máximo e mínimo que $\gamma(\omega)$ pode assumir, e qual orientação $\omega^* \in \Omega$ leva a estes valores? No entanto, os problemas de decisão associados com tais questionamentos são provadamente NP-completos, o que, dentre outras coisas, significa que um algoritmo polinomial para resolvê-los é desconhecido, mas podemos validar de forma eficiente se um dado certificado de fato é a solução do modelo.

O problema de maximizar a métrica de concorrência foi provado como sendo NP-completo através de uma redução envolvendo coloração em grafos [11]. Mais tarde, provou-se que a NP-completude é mantida para grafos com grau máximo 4 [14]. No

entanto, existem algoritmos polinomiais para encontrar a concorrência máxima de algumas famílias de grafos. Em especial, este problema é fácil para árvores e grafos bipartidos (que possuem $\gamma_{max} = 1/2$), grafos completos K_n (possuindo $\gamma_{max} = 1/n$) e grafos circulares C_n (com $\gamma_{max} = \lfloor n/2 \rfloor / n$, não confundir com grafos circulantes Ci_n).

Por outro lado, o problema de encontrar concorrências mínimas foi provado como sendo NP-completo para grafos cúbicos planares através de uma redução envolvendo ciclos hamiltonianos [14]. Apesar desta limitação, o capítulo 4 deste trabalho se dedica a apresentar técnicas de otimização combinatória para resolver instâncias significativamente grandes de concorrência mínima em tempos aceitáveis de *CPU*.

Capítulo 3

Aplicações do Escalonamento por Reversão de Arestas

Neste capítulo, serão abordadas algumas das modelagens presentes na literatura que utilizam o algoritmo de Escalonamento por Reversão de Arestas para reger a dinâmica de seus sistemas. Em especial, temos interesse em dividir tais abordagens em dois grupos distintos: no primeiro, detalhado na seção 3.1, encontraremos os problemas cuja concorrência máxima é desejável para uma maior eficiência de suas soluções; em contrapartida, no segundo grupo, apresentado na seção 3.2, serão encontrados os problemas nos quais desejamos minimizar a concorrência do sistema para obter uma boa solução. Em ambos os casos, as dinâmicas de reversão de arestas detalhadas no capítulo 2 permanecem inalteradas.

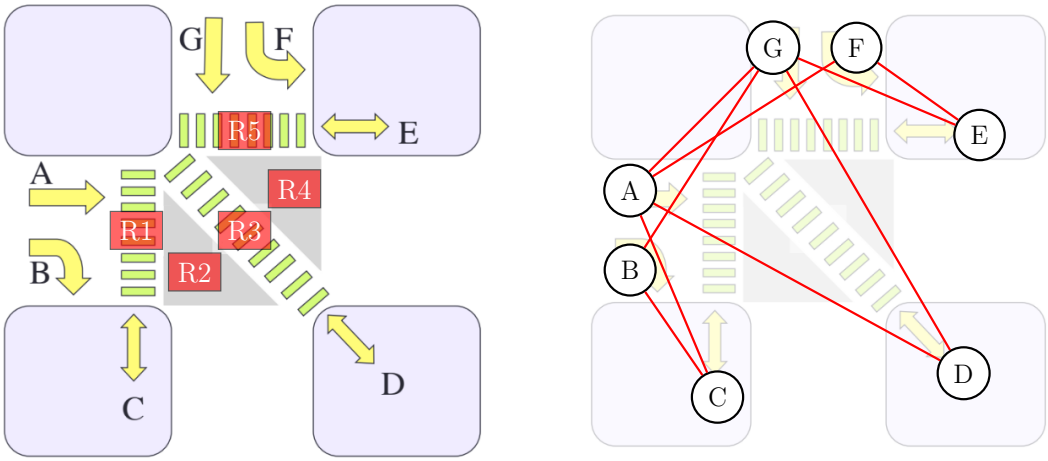
3.1 Aplicações com Concorrência Máxima

Nesta seção, serão apresentados problemas cujas soluções desejáveis são obtidas através de orientações acíclicas que levam à concorrência máxima do sistema. De modo geral, o objetivo é que o compartilhamento de recursos ocorra da forma mais eficiente possível, diminuindo o tempo que cada nó permanece em espera.

3.1.1 Controle Distribuído de Semáforos em um Cruzamento

A Figura 1.2, apresentada na seção 1.1 e reproduzida com alterações na Figura 3.1a, ilustra um cruzamento entre duas vias em que a passagem de pedestres e carros

deve ser continuamente coordenada. Neste cenário, os recursos compartilhados correspondem aos espaços físicos destinados à travessia dos participantes da dinâmica, enquanto cada fluxo de carros ou pedestres pode ser representado por um nó. No contexto da técnica de Escalonamento por Reversão de Arestas, a "operação" de um nó corresponderá a um sinal verde, possibilitando que o fluxo por ele representado ocorra. Isto só é possível devido à garantia de que os demais nós que compartilham os mesmos recursos do fluxo em questão estarão sem "operar", aguardando seus respectivos turnos [2, 18]. A Figura 3.1 ilustra esta modelagem.

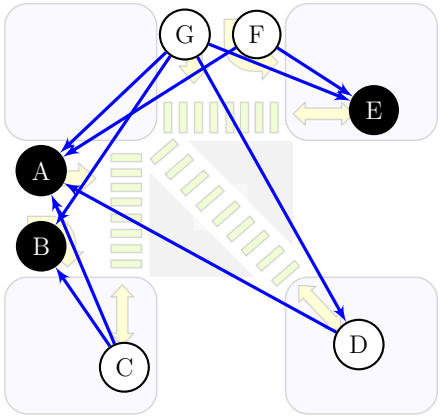


(a) Cruzamento com as regiões de recursos compartilhados em destaque.

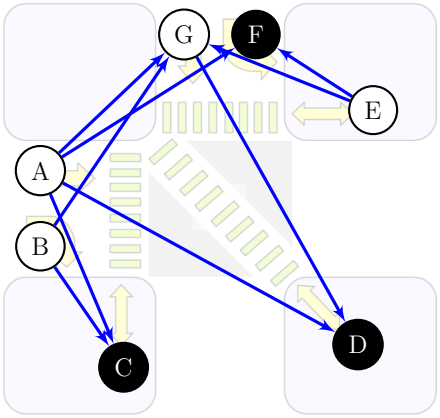
(b) Grafo de recursos representando o cruzamento e seus fluxos.

Figura 3.1: Modelagem do *Cruzamento de Shibuya*, em Tóquio, utilizando um grafo de recursos.

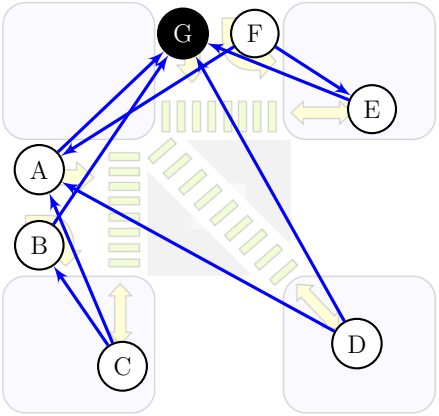
Após a construção do grafo de recursos representado na Figura 3.1b, resta propor uma orientação acíclica inicial ω de modo a obter altos valores da métrica de concorrência $\gamma(\omega)$, implicando em mais pedestres e carros realizando suas dinâmicas simultaneamente. Por exemplo, a Figura 3.2 ilustra um período a partir de uma orientação inicial ω que leva o sistema à sua concorrência máxima, de valor $\gamma(\omega) = 1/3$. No entanto, podemos notar que, durante a terceira orientação (Figura 3.2c), há recursos ociosos. Isto ocorre devido ao fato do *Escalonamento por Reversão de Arestas* garantir um tempo igual de operação para todos os nós, o que acaba levando a uma sub-utilização dos recursos disponíveis.



(a) Primeira orientação.



(b) Segunda orientação.



(c) Terceira orientação.

Figura 3.2: Período originado a partir de uma orientação inicial ω que leva o sistema do *Cruzamento de Shibuya* à sua concorrência máxima, de valor $\gamma(\omega) = 1/3$. Sumidouros estão representados em preto.

3.1.2 Aplicações em *Job shops*

Um sistema de logística é compreendido por um fluxo de recursos entre um ponto de origem e um ponto de destino, podendo incluir itens físicos ou abstratos. Em especial, a logística de uma fábrica impacta diretamente na eficiência da produção e no seu consumo de energia, sendo crucial otimizar o uso das diversas máquinas disponíveis e evitar possíveis gargalos de produção. Em um *job shop*, diferentes pedidos customizados de uma variedade de clientes possuem tempos de produção diferentes,

e nem todos os pedidos utilizam as mesmas máquinas para serem manufaturados. Portanto, neste cenário, os diversos parâmetros combinatórios do problema são um desafio para a otimização do uso das máquinas, além de impactar na elaboração de rotas para os veículos automatizados que atuam entre cada estação.

3.1.2.1 Escalonamento de Tarefas em Máquinas de Produção

Nesta subseção, estamos interessados em utilizar uma variante assíncrona do algoritmo de *Escalonamento por Reversão de Arestas* para solucionar o problema de escalonar n tarefas em m máquinas. Cada tarefa possui uma ordem específica de operação, que deve ser respeitada ao formularmos uma solução. A Tabela 3.1 ilustra um exemplo de problema de *job shop*, em que cada tarefa J_i possui exatamente 3 operações. Podemos ver, por exemplo, que a tarefa J_1 deverá passar 3 unidades de tempo na máquina M_A, seguir por 1 unidade de tempo na máquina M_B e, finalmente, utilizar a máquina M_C por 5 unidades de tempo.

	Operação 1		**Operação 2**		**Operação 3**	
	Tempo de proc.	Id. da máquina	Tempo de proc.	Id. da máquina	Tempo de proc.	Id. da máquina
J_1	3	M_A	1	M_B	5	M_C
J_2	9	M_C	10	M_B	3	M_A
J_3	10	M_B	8	M_C	6	M_A

Tabela 3.1: Exemplo de problema de *job shop*. Fonte: [6].

Formalmente, um *job shop* é um problema combinatório em que um conjunto J de n tarefas $J_1, J_2, ..., J_n$ deve ser processado em um conjunto M de m diferentes máquinas $M_1, M_2, ..., M_m$. Além disso, cada tarefa J_i consiste de uma sequência de m_i operações $O_{i,1}, O_{i,2}, ..., O_{i,m_i}$ que deve ser necessariamente escalonada nesta ordem. Cada máquina do *job shop* pode lidar com apenas uma tarefa por vez, e cada operação $O_{i,j}$ possui um tempo de processamento diferente representado por $p_{i,j}$. O objetivo do problema é minimizar a função objetivo $C_{final} = max\ C_i$, em que C_i denota o tempo de conclusão da última operação da tarefa J_i, para $i = 1, ..., n$. A Figura 3.3 ilustra um *job shop* genérico em operação sob estas condições [4]. O problema de decisão associado a encontrar sua solução ótima, no entanto, é conhecidamente NP-difícil até mesmo com apenas 3 tarefas e 3 máquinas [19].

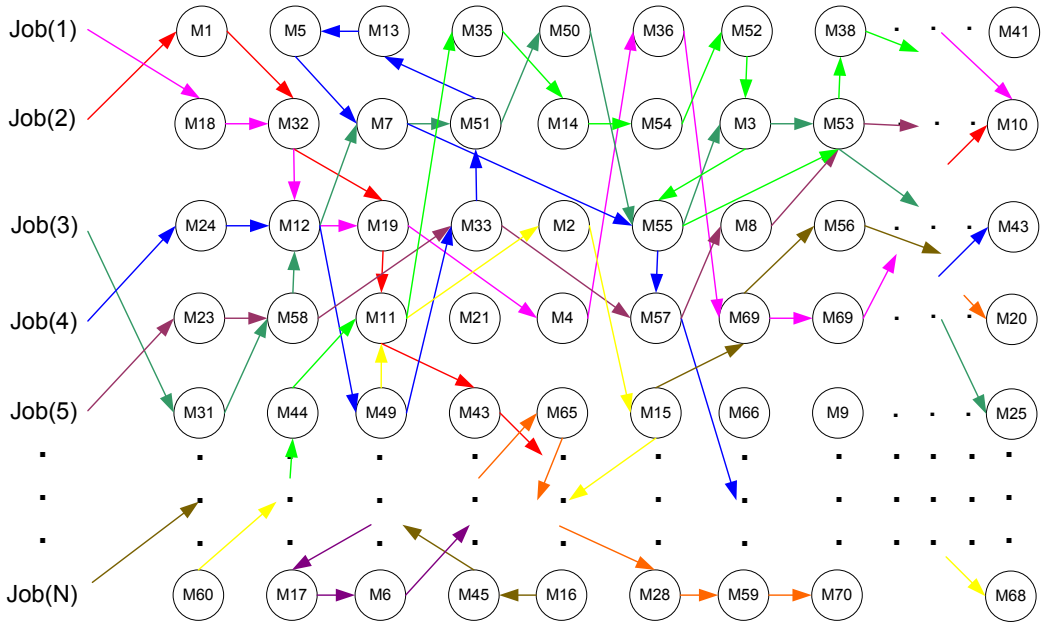


Figura 3.3: Esquematização de um problema de *job shop*, em que cada nó representa uma máquina. Arestas da mesma cor definem, para uma tarefa i que utiliza m_i máquinas, uma sequência de operações $O_{i,1}, O_{i,2}, ..., O_{i,m_i}$. Fonte: [4].

Devido às características intrínsecas do problema, a variante da técnica de *Escalonamento por Reversão de Arestas* a ser utilizada para obter uma de suas soluções viáveis é assíncrona. Isto significa que *sumidouros* de uma mesma orientação acíclica podem reverter suas arestas em momentos diferentes um do outro, sem depender de um "clock" para coordenar as reversões.

O grafo de recursos a ser gerado será constituído por subgrafos completos (cliques) que representam cada tarefa, sendo que cada nó dentro de um clique representará uma máquina. Ao codificarmos este compartilhamento de recursos entre máquinas, impedimos que uma mesma tarefa possua múltiplas operações simultâneas em diferentes máquinas. Logo, se, por exemplo, um nó M_A do clique J_2 for um sumidouro, isto representa que a tarefa J_2 está utilizando a máquina M_A.

No entanto, através desta modelagem, existem diferentes nós espalhados em cada processo representando a mesma máquina M_j. Devemos, portanto, formar cliques entre todos os nós com o rótulo M_j a fim de impossibilitar que uma mesma máquina lide com mais de uma tarefa por vez. Por fim, orientamos as arestas dentro de cada processo J_i na ordem em que suas operações devem ocorrer. Por exemplo, de acordo

com a Tabela 3.1, a tarefa J_1 deve operar inicialmente em M_A, depois em M_B e, por fim, em M_C. Logo, de um ponto de vista de decomposição em sumidouros e considerando apenas cada clique individualmente, orientamos o clique J_1 de modo que C fique no conjunto S_0, B esteja em S_1 e, C, no conjunto S_2 do clique.

A Figura 3.4 ilustra as configurações do grafo de recursos discutidas nos parágrafos

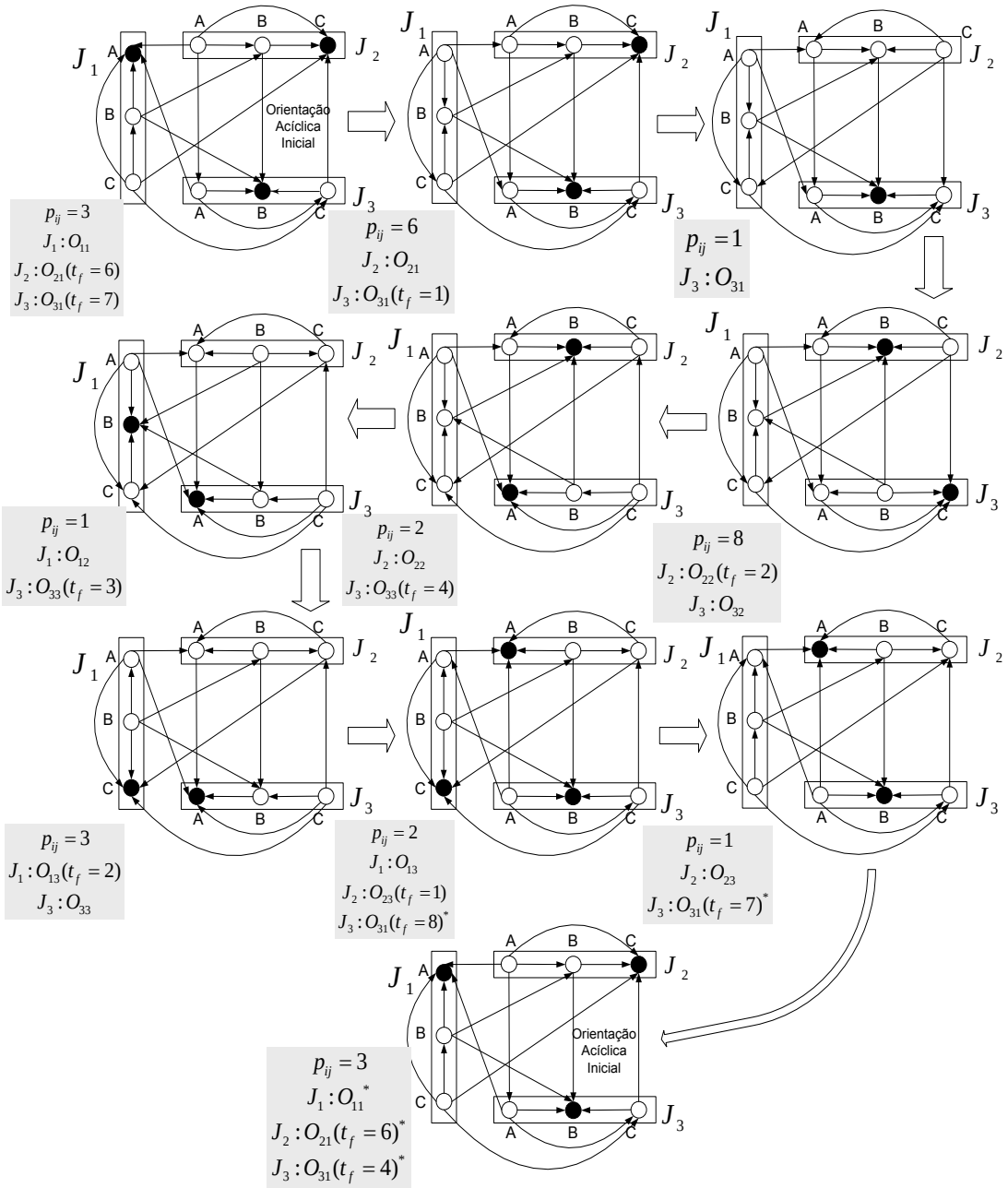


Figura 3.4: Possível escalonamento das tarefas da Tabela 3.1 em cada máquina do *job shop* utilizando uma variante assíncrona do *SER*. Fonte: [4].

anteriores para os parâmetros da Tabela 3.1, juntamente com um escalonamento que leva o sistema a uma solução viável do problema. A variável p_{ij} representa o tempo decorrido desde a orientação imediatamente anterior. Por sua vez, a variável t_f ao lado de cada operação indica a quantidade de tempo restante até que tal operação seja concluída, enquanto sua ausência sinaliza uma conclusão. Uma vantagem desta abordagem é a descentralização do controle de cada tarefa, permitindo que imprevistos no tempo de conclusão de cada operação não quebrem a dinâmica do sistema [4].

3.1.2.2 Planejamento de Rotas para *AGVs*

Recentemente, o transporte entre cada máquina de alguns *job shops* vem sendo feito por Veículos Guiados Automatizados (*AGVs*, do inglês *Automated Guided Vehicles*), representados na Figura 3.5 e capazes de aumentar consideravelmente a eficiência da produção [5]. Portanto, um *AGV* é responsável por recolher produtos intermediários de uma máquina e levá-los para a seguinte sem que ocorram *deadlocks* ou colisões com outros veículos. A fim de atingir este objetivo, é comum a utilização de algoritmos distribuídos para coordenar o roteamento de *AGVs* e seus caminhos até cada máquina. Em especial, sob certas restrições, uma possível solução para este problema é através da técnica de *Escalonamento por Reversão de Arestas*.

(a) Exemplo de *AGV*.

(b) Prateleiras de uma fábrica.

Figura 3.5: Fotos de um *AGV* e de um armazém de fábrica construído para a operação de *AGVs*. Fonte: [5].

Nesta variante do problema, o compartilhamento de recursos entre *AGVs* não se limita apenas à ordem de uso das máquinas, mas também às regiões espaciais nas quais elas estão localizadas. Por exemplo, a Figura 3.6 ilustra a planta de uma

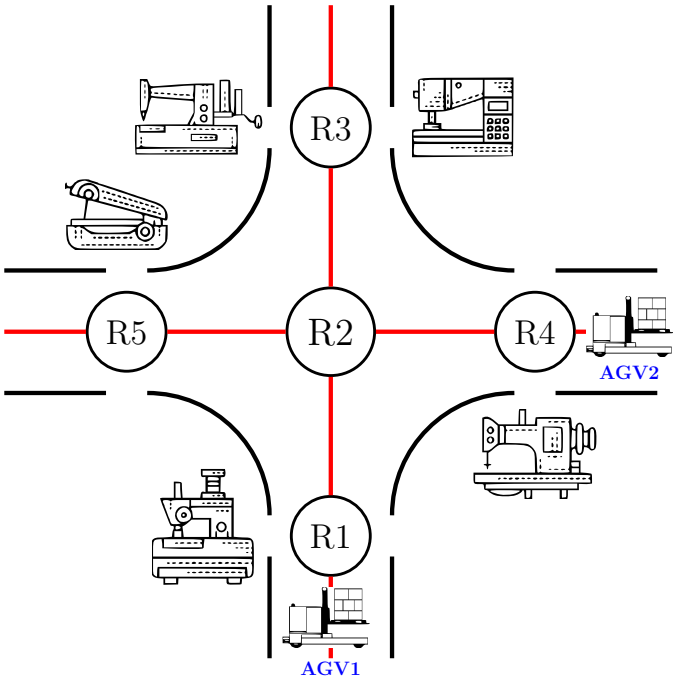


Figura 3.6: Planta fictícia de uma fábrica, em que nós representam estações onde existem máquinas de produção, enquanto arestas codificam caminhos de *AGVs*. Neste exemplo, o *AGV1* está em *R1*, enquanto o *AGV2* está em *R4*.

fábrica em que cada região $R1, ..., R5$ fornece acesso a uma ou duas máquinas, e cada AGV pode ocupar apenas uma região por vez. Assim, caso o *AGV1* precise utilizar a máquina da região *R4*, o *AGV2* deve manobrar até a região *R2* e, posteriormente, seguir para *R3* ou *R5*, possibilitando que o *AGV1* realize a rota $R1 \rightarrow R2 \rightarrow R4$.

Portanto, a fim de simplificar a modelagem do problema, podemos levar em conta apenas o tempo que cada veículo deve permanecer em uma dada região espacial. Por exemplo, tendo como base a planta apresentada na Figura 3.6 e a posição inicial dos *AGVs* nela retratados, e sendo t_0 uma unidade de tempo arbitrária, suponhamos que cada veículo precise realizar as seguintes rotas:

$$\begin{aligned} AGV1 &= R_1(4t_0) \rightarrow R_2(3t_0) \rightarrow R_3(4t_0) \\ AGV2 &= R_4(3t_0) \rightarrow R_2(4t_0) \rightarrow R_3(2t_0) \end{aligned} \tag{3.1}$$

De forma análoga à montagem do grafo de recursos discutida na subseção 3.1.2.1, devemos montar subgrafos completos (cliques) para cada AGV (pois cliques garan-

tem que um AGV nunca estará em duas regiões simultaneamente) e conectar todos os demais nós que codifiquem o mesmo recurso físico. A orientação acíclica inicial escolhida deve refletir a ordem, expressa na equação 3.1, em que cada veículo ocupa suas regiões espaciais. Por exemplo, sob uma análise de decomposição de sumidouros (discutida na subseção 2.3.1), o clique representando o *AGV1* deve ter o nó que codifica *R1* pertencendo a S_0 (sumidouro), o nó *R2* em S_1 e, por fim, o nó *R3* em S_2. Assim como discutido na subseção 3.1.2.1, usaremos uma variante assíncrona do *SER*. A Figura 3.7 ilustra um escalonamento em que o nó A representa *R1*, D representa R4, B e E representam *R2* e, por fim, C e F representam *R3*.

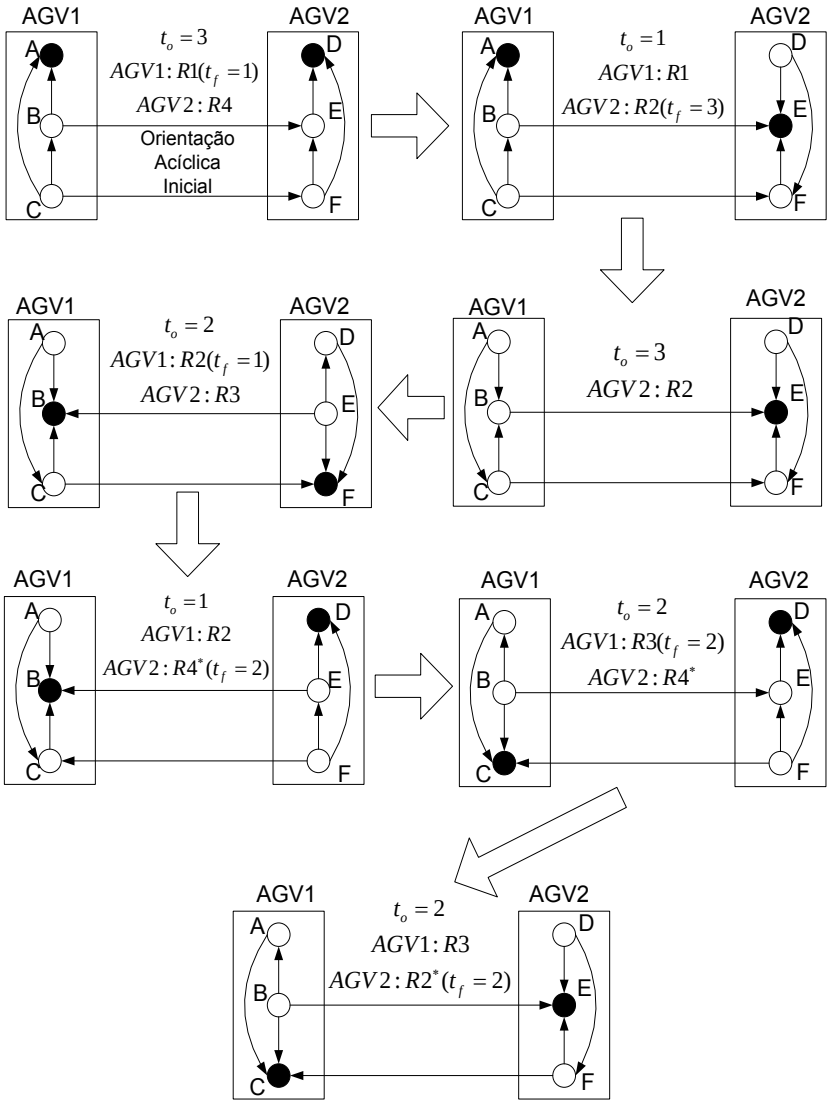


Figura 3.7: Possível escalonamento das rotas para $AGVs$ da Equação 3.1, em que cada nó representa uma região espacial da Figura 3.6. Fonte: [4].

3.2 Aplicações com Concorrência Mínima

Os problemas discutidos nesta seção possuem soluções mais eficientes quando escalonados sob concorrência mínima. Em outras palavras, desejamos encontrar uma orientação acíclica ω^* cujo valor de $\gamma(\omega^*)$ seja mínimo para todo $\omega \in \Omega$. Nesta seção, tais problemas serão apresentados e analisados, podendo ser diretamente beneficiados pela técnica posteriormente descrita no capítulo 4.

3.2.1 Descontaminação de *WebGraphs*

WebGraphs são modelos propostos para capturar a relação, em forma de *links*, entre páginas da *Web*. Em especial, estruturas denominadas *link farms* surgem quando um *spammer* cria *links* artificiais para uma certa página T , com a intenção de aumentar sua visibilidade em ferramentas de busca (ex.: tornar a página T mais relevante ao realizar buscas através do *Google*). Posteriormente, o *spammer* disponibiliza *links* para as páginas que compõem a *link farm* em *blogs* ou outros sites públicos em que comentários possam ser escritos [20]. A Figura 3.8 ilustra esta estrutura.

Apesar da *link farm* F ter sido esquematizada por uma elipse na Figura 3.8, sua representação interna também é a de um grafo, já que seu papel continua sendo o de relacionar páginas da *Web* através de *links*. Análises estatísticas experimentais utilizando robôs (*crawlers*) mostram que tais grafos são densamente conexos [21].

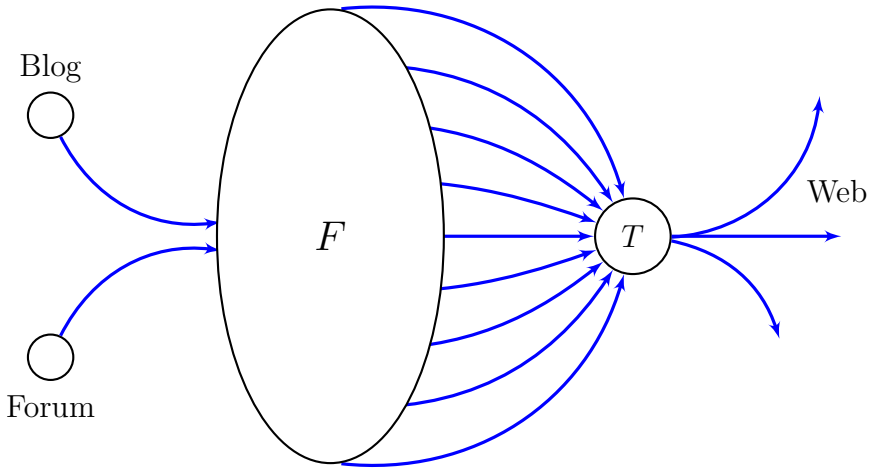
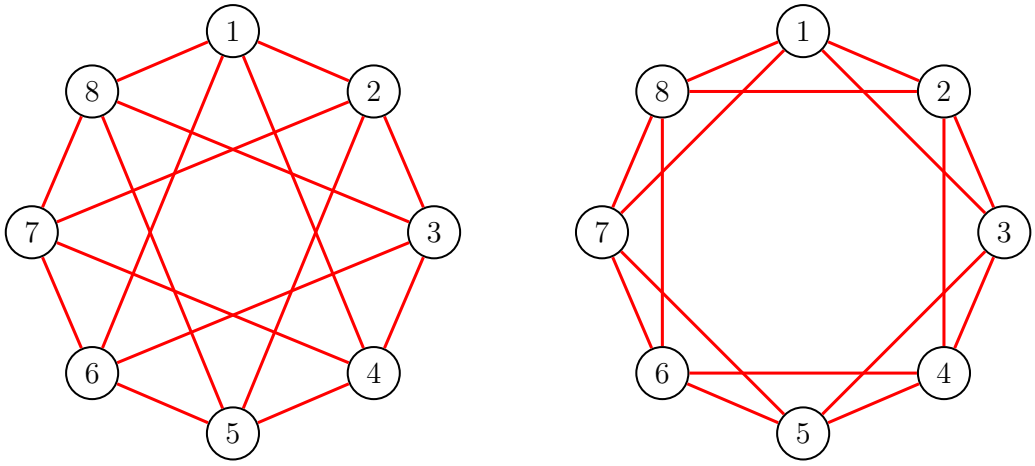


Figura 3.8: Esquematização de uma rede de *spam* com uma *link farm* F.

3.2.1.1 Grafos Circulantes

Devido à sua característica de densa conectividade, uma classe popular para representar o interior de uma *link farm* é a de grafos circulantes [20], em que cada vértice i é conectado ao vértice $i + j$ e $i - j$, para cada j em uma lista L. Portanto, tais grafos são expressos pelo prefixo *"Ci"*, seguido do número de vértices n e, por fim, de uma lista com os valores de j. Por exemplo, o grafo circulante $Ci_n(1, 2, ..., \lfloor n/2 \rfloor)$ expressa o grafo completo K_n, enquanto $Ci_n(1)$ define o grafo circular C_n. A Figura 3.9 ilustra outros exemplos desta família de grafos, que também será relevante para futuras seções deste trabalho.



(a) Grafo $Ci_8(1, 3)$.

(b) Grafo $Ci_8(1, 2)$.

Figura 3.9: Exemplos de grafos circulantes.

3.2.1.2 Dinâmica de Descontaminação

A fim de fragilizar a rede de *spam*, a estratégia empregada é a descontaminação da *link farm* F. Inicialmente, consideramos que todos os vértices de F estejam "contaminados", ou seja, possuem um *link* para a página T. Um *web marshall (WM)* pode ser enviado a um vértice v para descontaminá-lo, levando v ao estado de "vigiado". Em algum momento, este *WM* sairá de v, levando-o ao estado "descontaminado". No entanto, uma recontaminação é possível dependendo do número de vizinhos de v ainda contaminados. Em algumas variantes do problema, basta que a maioria dos vizinhos de v estejam no estado "vigiado" ou "descontaminado" para que v permaneça "descontaminado"; em outras variantes, é necessário que todos os vizinhos

de v estejam "vigiados" ou "descontaminados". A dinâmica acaba quando todos os vértices de F atingem o estado de "descontaminado" [20].

Na literatura, o problema de minimizar o número de *WMs* (ou qualquer outro agente que atue para descontaminar um grafo G) é chamado de *node searching problem*, enquanto o número mínimo de agentes necessários para tal descontaminação de G é representado por $ns(G)$ [22]. Este problema foi provado como sendo NP-completo, tornando-se polinomial apenas para algumas classes de grafos, como árvores e cografos [23].

Uma estratégia para a descontaminação de G é a utilização da técnica de Escalonamento por Reversão de Arestas. Adaptada e renomeada sob o nome *Alg-Descontaminação* [7], tal algoritmo fornece soluções tão boas ou melhores do que outras estratégias propostas em trabalhos distintos [22] para qualquer grafo conexo G. Esta estratégia se baseia na decomposição em sumidouros apresentada na subseção 2.3.1, em que o conjunto S_k contém todos os nós cujo caminho máximo até um sumidouro possui comprimento k.

De modo geral, dada uma orientação acíclica inicial ω, *web marshalls* são enviados para os sumidouros de ω (*i.e.* todo $v \in S_0$) a fim de iniciar o processo de descontaminação. Ao desestabilizarem os *links* dos sumidouros para a página T cuja visibilidade o *spammer* objetivava aumentar, cada *WM* checa se há risco de recontaminação. Dependendo da regra de recontaminação empregada (ex.: um nó não-vigiado é recontaminado se a maioria de seus vizinhos estiver em "contaminado"; ou um nó não-vigiado é recontaminado se houver pelo menos um vizinho em "contaminado"), o *WM* pode finalizar sua execução ou criar réplicas de si mesmo e enviá-las para seus vizinhos "contaminados" que se tornarão sumidouros na próxima rodada. O processo acaba quando todos os vértices de G estiverem no estado "descontaminado". O Algoritmo 1 esquematiza este raciocínio.

De imediato, podemos notar que o número de *WMs* empregados está relacionado com a quantidade de sumidouros existentes em ω. Em geral, níveis de concorrência $\gamma(\omega)$ elevados implicam em vértices operando simultaneamente e, por consequência, em um uso excessivo de *WMs* [7]. Portanto, ao utilizarmos a técnica de *Escalonamento por Reversão de Arestas* para a descontaminação de um grafo conexo $G = (V, E)$, é comum buscarmos uma orientação acíclica inicial que leve a uma

Algoritmo 1: Alg-Descontaminação, uma adaptação da técnica de Escalonamento por Reversão de Arestas para descontaminar grafos conexos.

Input : Um grafo não-direcionado $G = (V, E)$ e uma orientação acíclica ω

Output: Uma estratégia para a descontaminação de G

Posicionar *WMs (web marshalls)* em sumidouros, tornando-os "vigiados"

enquanto *todos os vértices em* V *não estiverem "descontaminados"* **faça**

  para *cada sumidouro* v *em* S_0 **faça**

    WM destrói os links de v para a página T, descontaminando-o

    se *condição de não-recontaminação for satisfeita* **então**

      WM termina sua execução e declara v como "descontaminado"

    fim

    senão

      $N_c(v)$ = vizinhos "contaminados" de v

      $U = N_c(v) \cap S_1$, ou seja, vizinhos de v que se tornarão sumidouros

      WM cria $|U|$ cópias de si mesmo

      WM envia cópias para todo $u \in U$, marcando-os como "vigiados"

    fim

  fim

  Passo síncrono: reverter as arestas de todos os nós (sumidouros) em S_0

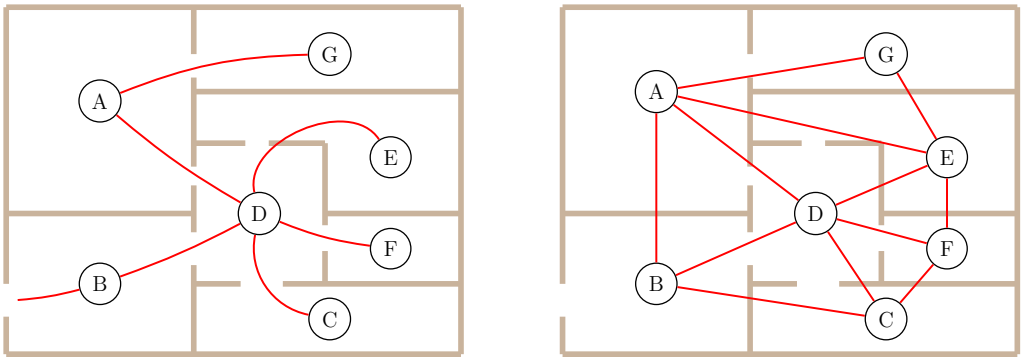
fim

concorrência o mais perto possível de $1/|V|$. No entanto, até o momento, ainda não está clara na literatura a relação exata entre a concorrência do sistema $\gamma(\omega)$ e $ns(G)$, o número mínimo de agentes necessários para descontaminar G.

3.2.2 Combate ao Incêndio através de Robôs Autônomos

A filosofia por trás da dinâmica de descontaminação introduzida na subseção 3.2.1 pode ser adaptada e estendida para outros cenários além da *Web*. Em especial, projetos de combate ao incêndio utilizando a dinâmica de reversão de arestas para guiar robôs autônomos foram testados e simulados em laboratório [3]. Em situações como esta, a dinâmica de descontaminação está restringida também ao espaço físico, dado que um nó pode ser recontaminado por outro mesmo que não haja uma relação

de caminho entre os dois (ex.: existência de uma janela; de uma parede pouco resistente; etc). Portanto, cria-se a necessidade de modelar o problema utilizando dois grafos, como mostra a Figura 3.10 ao exemplificar um típico apartamento: no primeiro grafo, denominado *grafo de locomoção* e representado pela Figura 3.10a, codificamos a relação entre caminhos físicos a serem utilizados pelos robôs. No segundo, mostrado na Figura 3.10b, relacionamos as possibilidades de recontaminação de cada nó. Neste exemplo, especificamente, cômodos podem se recontaminar caso sejam adjacentes entre si.



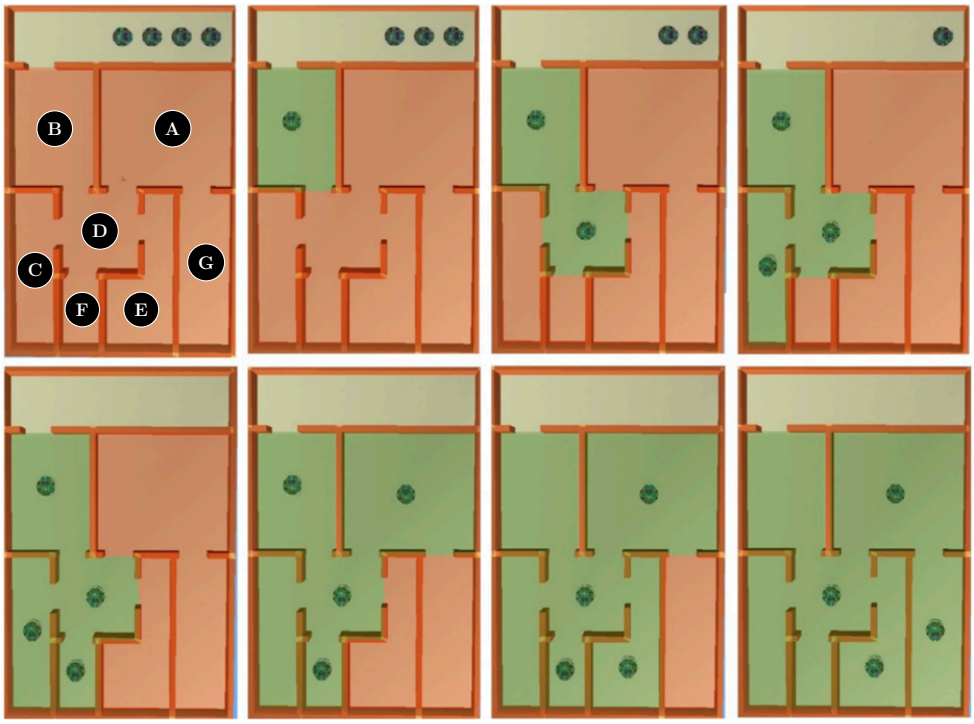
(a) Grafo de locomoção.

(b) Grafo de recontaminação.

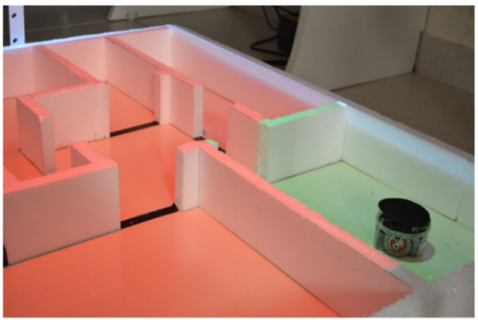
Figura 3.10: Esquema de planta de um apartamento em chamas. Uma aresta no grafo em (a) representa um caminho entre um cômodo e outro, enquanto uma aresta no grafo em (b) codifica cômodos que podem se recontaminar.

Além disso, novas restrições surgem devido às limitações físicas do problema. Por exemplo, novos robôs não podem surgir ou desaparecer sob demanda, como era o caso envolvendo *web marshalls* discutido na subseção 3.2.1. Uma boa solução especialista deve levar em conta também outros fatores de contaminação: por exemplo, é possível que um robô possa sair um pouco mais cedo de um nó desde que as chances de recontaminação neste curto intervalo de tempo em que tal nó ficará sem um agente sejam suficientemente baixas.

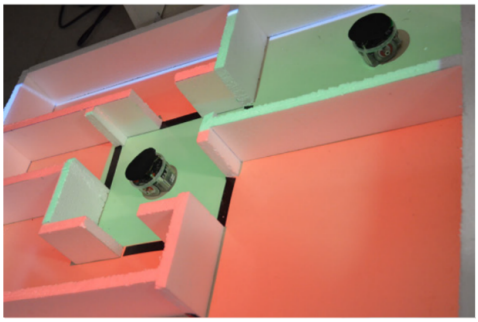
A Figura 3.11 apresenta simulações de uma típica utilização da técnica de *Escalonamento por Reversão de Arestas* para a descontaminação da planta de apartamento apresentada na Figura 3.10. Por fim, o emprego da técnica parte do princípio de que os robôs possuem comunicação entre si, sendo necessário o uso paralelo de algoritmos de propagação da informação [24].



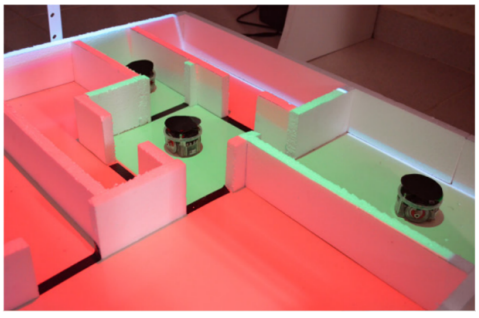
(a) Sequência de descontaminação completa baseada no ambiente *Webots* de simulação.



(b) Primeiro robô descontamina o nó B.



(c) Segundo robô descontamina o nó D.



(d) Terceiro robô descontamina o nó F.

Figura 3.11: Descontaminação da planta de apartamento da Figura 3.10. Fonte: [3].

3.2.3 Escalonamento de *Loops* Musicais Máximos

Como expressado por Shan e Chiu [25], a geração efetiva de música por computador é o sonho dos pesquisadores da área. Abordagens anteriores explícitas (em que as regras de composição são especificadas por humanos) recorreram a *Modelos de Markov Escondidos* para capturar os requisitos de sequência que uma melodia possui [26], mas geralmente ficam limitadas a compor *contraponto* ou *harmonização* para músicas já existentes [27].

Nesta subseção, será mostrado como um sistema sob concorrência mínima do algoritmo *SER* é capaz de gerar um *loop* de comprimento máximo de *frases musicais* pré-gravadas, respeitando conceitos essenciais de teoria musical e criando melodias originais para *blues, jazz* e *rock*. Na subseção 3.2.3.1, serão introduzidos os termos a serem usados durante toda a subseção 3.2.3.2, que fornece uma estratégia para representar *frases musicais* como um grafo. Por fim, na subseção 3.2.3.3, serão discutidos detalhes específicos da implementação de uma simulação complementar incluída no Apêndice A.

3.2.3.1 Definições de Teoria Musical

Inicialmente, iremos recorrer ao livro de Schmidt-Jones [28] para definirmos a terminologia necessária de teoria musical empregada ao longo desta seção. Uma **frase musical** corresponde a um grupo de notas individuais que, juntas, expressam uma ideia melódica definida. É comum que tais *frases* apareçam em pares: a primeira *frase* costuma soar inacabada até que seja concluída pela segunda, quase como se a última estivesse respondendo a uma pergunta feita pela primeira. *Frases* que respeitam esta dinâmica são chamadas de **antecedentes** e **consequentes**, respectivamente.

Um **compasso** é um grupo de *batidas* que ocorrem durante um segmento de tempo. Quando mais de uma melodia independente ocorre durante o mesmo *compasso*, podemos chamar uma música de **polifônica** (por exemplo, “Canon” de Pachelbel; último refrão de “One Day More”, do musical “Les Miserables”). Finalmente, um **lick** corresponde a uma breve ideia musical que aparece em muitas composições do mesmo gênero. Neste trabalho, um par de *frases antecedentes* e *consequentes*, quando tocadas sequencialmente, também será chamado de *lick*.

3.2.3.2 Representação em Grafos

Apesar de acreditarmos que a geração de música através do *SER* possa ser empregada para agregar qualquer unidade musical (como acordes ou notas individuais) em uma composição, a aplicação que propomos gira em torno do escalonamento de *frases musicais*. Especificamente, gostaríamos de capturar os seguintes requisitos:

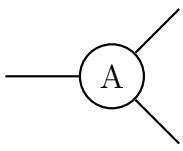
(i) Uma frase **consequente** apenas pode ser tocada após uma frase **antecedente**, formando um *lick*;

(ii) Se duas ou mais *frases* estão tocando ao mesmo tempo, ou todas são **antecedentes** ou todas são **consequentes**;

(iii) *Frases* de diferentes intensidades (*e.g.* número de notas) podem não soar bem juntas;

(iv) A composição final deve ser um *loop*, conter todas as *frases* disponíveis e possuir **comprimento máximo**.

Ao organizarmos *frases* previamente gravadas (ou geradas) em um grafo, nosso objetivo é estruturar quais *frases* podem ser tocadas sequencialmente e quais podem ser tocadas simultaneamente, formando uma *polifonia*. Ao representarmos cada *frase* como um nó, seremos capazes de capturar as restrições previamente mencionadas através da inserção de arestas. Em um *grafo de recursos*, uma aresta entre dois nós representa a inabilidade destes nós de *operarem* ao mesmo tempo. Portanto, uma aresta entre duas *frases* é capaz de prevenir suas ocorrências durante o mesmo *compasso*, enquanto permite que cada uma destas *frases* possa ocorrer sequencialmente.



(a) Uma *frase antecedente*.

Property	Value
Type	*Antecedent*
Genre	Blues
Note Count	8
File	antec02.mp3

(b) Atributos de nó.

Figura 3.12: Exemplo de um nó e seus atributos.

A Figura 3.12 apresenta as informações contidas em cada nó. Um *número de notas*, correspondendo ao total de notas contidas em uma *frase*, é usado para medir sua intensidade. Para este exemplo em específico, dois nós serão conectados por uma aresta se e somente se:

(1) os nós são de diferentes tipos (*antecedente* e *consequente*);

(2) seus *números de notas* estão dentro de um limiar pré-determinado;

(3) os nós pertencem ao mesmo gênero musical.

Além disso, gostaríamos de tornar este exemplo mais interessante ao permitirmos que uma transição entre os gêneros *blues* e *jazz* ocorra durante algum momento. Ao introduzirmos *frases de transição* que incorporam elementos de ambos os gêneros, uma passagem mais fluida pode ser obtida. *Frases antecedentes* de *blues* e *jazz*, quando conectadas a *nós de transição*, podem agir como portas de entrada para seus respectivos gêneros.

A Figura 3.13 ilustra todos os tópicos previamente discutidos. Nós marcados como "A" e "C" representam *frases antecedentes* e *consequentes*, respectivamente. Quanto mais distante um nó estiver de um *nó de transição*, mais intensa é a frase que ele representa. Devido à dinâmica de *antecedente / consequente*, o grafo resultante é bipartido, para o qual o Problema de Concorrência Mínima permanece NP-completo [29].

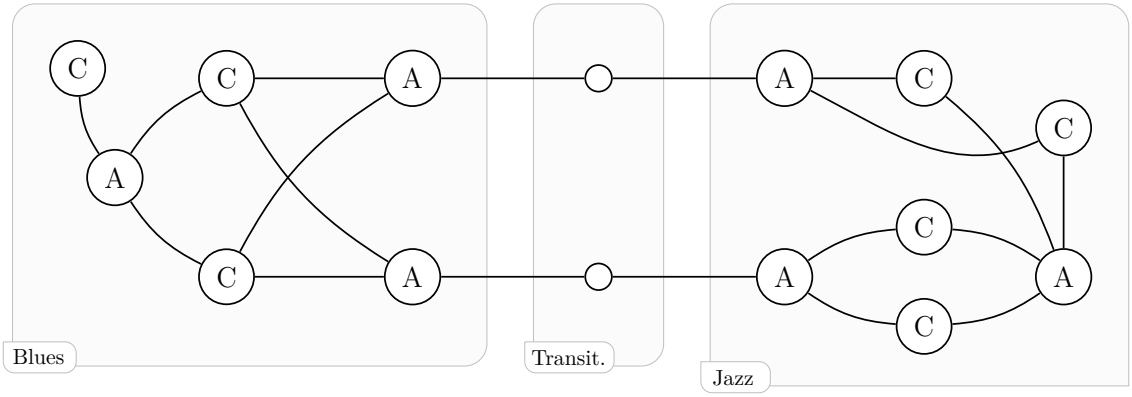


Figura 3.13: Um grafo de recursos em que nós marcados como "A" e "C" representam *frases antecedentes* e *consequentes*, respectivamente. Arestas impedem nós de *operar* ao mesmo tempo, mas permitem uma execução em sequência.

3.2.3.3 Detalhes de Implementação

A fim de demonstrar as ideias discutidas na subseção 3.2.3.2, foi desenvolvida uma simulação evidenciando como a dinâmica de escalonamento de *frases*, quando aplicada a um grafo tal como o da Figura 3.13, é capaz de produzir *loops* musicais de comprimento máximo. Nesta subseção, os passos realizados serão documentados e breves discussões sobre detalhes de implementação serão realizadas a fim de facilitar trabalhos futuros. O resultado final, mostrando o grafo de recursos da Figura 3.13, é apresentado no Apêndice A.

De um ponto de vista de compatibilidade, uma simulação na *Web* construída com **JavaScript** é leve e fácil de ser acessada na maioria das plataformas. Além disso, duas bibliotecas convenientes, disponíveis sob a *Licença MIT*, tornaram esta escolha ainda mais atrativa: **Vis.js** [30], que nos permitiu representar visualmente qualquer grafo e lidar com a dinâmica de reversão de arestas; e **Howler.js** [31], fornecendo uma interface de áudio confiável ao lidar com múltiplos arquivos.

Em seguida, foram escolhidas as faixas de áudio responsáveis pelas seções de *ritmo* (também conhecidas como *backing tracks*) e gravadas, em uma guitarra, todas as *frases antecedentes* e *consequentes*. Dado que esta simulação possui apenas 15 nós, o processo de sincronizar cada *frase* à sua *backing track* correspondente foi feito manualmente. Por exemplo, uma composição de *Blues de 12 Compassos* costuma alternar entre frases *antecedentes* e *consequentes* a cada 2 *compassos*. Diferentes *frases* possuem diferentes pontos de início dentro desta janela, necessitando de um *offset* para possibilitar a sincronização. No entanto, uma vez sincronizadas, *frases musicais* podem ser tocadas sempre que uma nova janela de *2 compassos* começar. Portanto, ao definirmos a frequência de reversão de arestas para *2 compassos*, todas as *frases* irão soar naturalmente quando seus nós correspondentes forem transformados em sumidouros.

Capítulo 4

Concorrência Mínima via Ciclos Maximais

Neste capítulo, serão abordadas as principais contribuições teóricas e experimentais deste trabalho para a literatura. Em especial, mostraremos que a obtenção de concorrência mínima em um grafo de recursos G sob uma dinâmica de *Escalonamento por Reversão de Arestas* pode ser feita em tempo linear caso um ciclo maximal de G seja adrede identificado. Inicialmente, na seção 4.1, a relação entre concorrência mínima e ciclos maximais será demonstrada de um ponto de vista teórico. Com isso, a seção 4.2 se dedica a elaborar e avaliar um método algorítmico para a obtenção de concorrência mínima, através de um algoritmo de duas etapas discutido na subseção 4.2.1. Por fim, a subseção 4.2.2 inclui resultados experimentais de instâncias relacionadas ao problema de descontaminação em grafos previamente discutido no capítulo 3, demonstrando a viabilidade da proposta.

4.1 Derivação Teórica

O objetivo desta seção é desenvolver uma expressão para o cálculo da concorrência mínima de um grafo conexo não-direcionado $G = (V, E)$, com $|E| \geq |V|$ (ou seja, G não é uma árvore). Os conceitos teóricos aqui utilizados foram formalmente apresentados na seção 2.4, sendo especialmente úteis as definições: de *ciclo simples* (representado por κ, sendo K o conjunto de todos os ciclos simples de G); de *orientação acíclica* (representada por ω, sendo Ω o conjunto de todas as orientações

acíclicas de G); de $n_{cw}(\kappa, \omega)$ e $n_{ccw}(\kappa, \omega)$, que são o número de arestas orientadas em κ por ω no sentido horário e anti-horário, respectivamente; e de *concorrência*, cuja definição estática encontra-se reproduzida abaixo:

$$\gamma(\omega) = \min_{\kappa \in K} \left\{ \frac{min\left\{n_{cw}(\kappa, \omega), n_{ccw}(\kappa, \omega)\right\}}{|\kappa|} \right\} \tag{4.1}$$

O conjunto Ω é finito, e corresponde a todas as orientações acíclicas possíveis de G. Cada orientação ω deste conjunto possui um valor $\gamma(\omega)$ de concorrência associado, sendo que nosso objetivo é varrer o conjunto Ω e buscar o menor desses valores, ao qual chamaremos de γ^*. Formalmente, podemos minimizar a equação 4.1 para todo $\omega \in \Omega$:

$$\gamma^* = \min_{\omega \in \Omega} \left\{ \min_{\kappa \in K} \left\{ \frac{min\left\{n_{cw}(\kappa, \omega), n_{ccw}(\kappa, \omega)\right\}}{|\kappa|} \right\} \right\} \tag{4.2}$$

Alternativamente, o seguinte lema vale:

Lema 1. $\gamma^* = \min_{\kappa \in K} \left\{ \frac{1}{|\kappa|} \right\}$

Demonstração. Considere a Equação 4.1. Para um dado ω', seja κ' o ciclo que produzirá o resultado escolhido pelo minimizador externo. Seja x definido como $x = min\left\{n_{cw}(\kappa', \omega'), n_{ccw}(\kappa', \omega')\right\}$, levando a equação 4.1 ao valor de $x/|\kappa'|$. No entanto, sabemos que, para todo $\kappa \in K$, sempre existirá uma orientação acíclica $\omega \in \Omega$ tal que $n_{cw}(\kappa, \omega) = 1$ e $n_{ccw}(\kappa, \omega) = |\kappa| - 1$, ou vice-versa (isto decorre imediatamente do fato de que um ciclo direcionado apenas é formado se e somente se $n_{cw}(\kappa, \omega) = 0$ ou $n_{ccw}(\kappa, \omega) = 0$). Logo, deverá haver também uma orientação ω de κ' tal que ou $n_{cw}(\kappa', \omega) = 1$ ou $n_{ccw}(\kappa', \omega) = 1$. Consequentemente, se a orientação ω', aplicada a κ', não produziu o resultado $x = 1$, necessariamente existirá outra orientação acíclica ω que produzirá $\gamma(\omega) = 1/|\kappa'|$. Agora, considere a Equação 4.2. Se $\gamma*$ for menor do que $1/|\kappa'|$, deverá necessariamente existir um ciclo simples κ^* que, sob uma orientação ω^*, produzirá $1/|\kappa^*| < 1/|\kappa'|$. Logo, a Equação 4.2 tornou-se um problema de minimização sob todo $\kappa \in K$. □

O Lema 1 é essencialmente o problema de encontrar um ciclo maximal não-direcionado de G, cuja concorrência mínima será igual ao inverso do tamanho de sua circunferência. Com isso, caso seja possível obtermos o tamanho do maior ciclo simples do grafo de recursos G, saberemos imediatamente a menor concorrência possível que G poderá alcançar sob a dinâmica de *Escalonamento por Reversão de Arestas*.

4.2 Encontrando uma Orientação Inicial que leve à Concorrência Mínima

Apesar dos resultados da seção 4.1 permitirem o cálculo do valor da concorrência mínima que o grafo de recursos G pode alcançar, ainda não está claro como obter ω^*, uma orientação acíclica tal que $\gamma(\omega^*) = \gamma^*$. Esta seção de dedica a formular um algoritmo que leve a ω^*, além de discutir sua complexidade. Em especial, o Teorema 1 é o principal resultado deste trabalho:

Teorema 1. *Dado qualquer ciclo maximal $\kappa^* \in K$ como entrada, existe um algoritmo de complexidade linear para encontrar uma orientação $\omega^* \in \Omega$ tal que $\gamma(\omega^*)$ é mínimo para todo $\omega \in \Omega$.*

Demonstração. A equação 1 indica que a concorrência mínima será atingida se uma orientação acíclica ω^* for aplicada a G sob a condição de que $n_{cw}(\kappa^*, \omega^*) = 1$ e $n_{ccw}(\kappa^*, \omega^*) = |\kappa^*| - 1$ ou vice versa, em que κ^* é um ciclo maximal. Orientar κ^* sob as condições previamente mencionadas pode ser realizado em tempo linear ao percorrermos o ciclo κ^* e atribuirmos um número de identificação crescente $1, ..., |\kappa^*|$ para cada vértice visitado, resultando em uma ordenação topológica do ciclo. Ao orientarmos as arestas correspondentes em direção aos vértices com números de identificação maiores, apenas uma aresta (conectando os vértices de maior e menor números de identificação) será orientada no sentido oposto às outras $|\kappa^*| - 1$, cumprindo o requisito.

Torna-se necessário, então, demonstrar que é possível orientar as demais arestas de G tal que a orientação resultante ω^* é sempre acíclica. Seja $S = V - \kappa^*$ o conjunto dos vértices restantes de G. Vamos atribuir um número de identificação

crescente $|\kappa^*| + 1, ..., |V|$ para cada vértice em S, e então orientar todas as arestas de G na direção dos vértices com maior identificador. Por absurdo, se a orientação resultante ω^* for cíclica, existirá um caminho $i_0, i_1, ..., i_0$ (*i.e.* um ciclo direcionado). No entanto, como cada aresta sempre leva a vértices com números de identificação maiores, é impossível retornar a i_0 após a partida, para qualquer $i_0 \in V$. Portanto, nenhum ciclo será formado. □

4.2.1 Algoritmo Proposto

Finalmente, podemos agora elaborar um procedimento algorítmico em duas etapas que encontre uma orientação acíclica ω^* que leve o sistema à concorrência mínima. A primeira etapa consistirá em identificar um ciclo maximal κ^* do grafo de recursos G através de técnicas de otimização combinatória, cujas formulações serão discutidas na subseção 4.2.1.1. Em sequência, a segunda etapa deverá utilizar o ciclo κ^* para obter, em tempo linear, a orientação ω^*. Para tal, os conceitos já discutidos na demonstração do Teorema 1 serão reestruturados e transformados em um algoritmo, cuja formalização encontra-se na subseção 4.2.1.2.

4.2.1.1 Etapa Branch-and-Cut

O primeiro passo para a técnica proposta neste trabalho é encontrar um ciclo maximal em um grafo conexo não-direcionado G. O problema de decisão associado a identificar um ciclo cujo número de vértices é máximo em G é conhecidamente NP-completo, pois inclui implicitamente o caso especial de encontrar um ciclo hamiltoniano [8]. Logo, é vantajoso recorrermos a técnicas de otimização combinatória denominadas *branch-and-cut* [32], que percorrem todo o espaço de soluções possíveis e podam, da árvore de busca, eventuais combinações de respostas que não sejam viáveis ou promissoras. Para tal, primeiro será necessário modelarmos o problema de programação linear inteira, estabelecendo a função objetivo e suas restrições.

A modelagem proposta nesta etapa é baseada em uma recente pesquisa [33] que propõe uma nova abordagem para o problema de ciclos simples (*SCP*, do inglês *Simple Cycle Problem*). Neste paradigma, cada ciclo é decomposto em um caminho simples e em uma aresta adicional, que conecta os vértices extremos do caminho. A ideia é que o algoritmo separe restrições generalizadas de eliminação de subciclos

(*GSEC*, do inglês *Generalized Subtour Elimination Constraints*) em tempo real (à medida em que elas são violadas em uma relaxação de programação linear na árvore de enumeração do *branch-and-cut*).

A formulação proposta para o *SCP* utiliza as variáveis binárias $\{y_i \in \{0,1\} : i \in V\}$ para representar os vértices de G (logo, se y_i for igual a 1, o vértice i está dentro da solução). De forma análoga, a variável $\{x_e \in \{0,1\} : e \in E\}$ codifica as arestas do caminho simples a ser formado, enquanto a variável $\{z_e \in \mathbb{R} : e \in E\}$ codifica a aresta adicional que transformará o caminho em um ciclo. Para este presente trabalho, a formulação foi modificada para que a variável de custo fosse removida (já que todas as arestas de um grafo de recursos possuem custo unitário) e a função objetivo passasse a buscar uma região de máximo ao invés de mínimo, como apresentada na equação 4.3.

$$\max \{\sum_{e \in E}(x_e + z_e) : (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathcal{R} \cap (\mathbb{B}^{|E|}, \mathbb{B}^{|V|}, \mathbb{R}_+^{|E|})\}, \tag{4.3}$$

Para todo subconjunto $S \subseteq V$, iremos representar por $\delta(S) \subseteq E$ o conjunto de arestas com exatamente uma extremidade em S (logo, $\delta(\{i\})$, ou simplesmente $\delta(i)$, corresponde às arestas com uma extremidade no vértice i). De forma análoga, representamos por $E(S) \subseteq E$ as arestas com ambas as extremidades em S (ou seja, que começam e terminam em vértices do conjunto S). Por sua vez, (x, y, z) denotam uma solução do problema que, para ser viável, deve estar contida na região espacial definida como uma interseção de $\mathcal{R}$ (o espaço de restrições) com os valores que cada variável pode assumir (por exemplo, o símbolo $\mathbb{B}^{|V|}$, binário, denota $\{0,1\}^{|V|}$). Finalmente, definimos como $\mathcal{R}$ a região poliédrica limitada pelo conjunto de restrições a seguir:

$$\sum_{e \in E(S)} x_e \leq \sum_{i \in S \setminus \{j\}} y_i, \;\; \forall j \in S, \;\; S \subset V \tag{4.4}$$

$$\sum_{e \in E} x_e \geq 2 \tag{4.5}$$

$$\sum_{e \in E} z_e = 1 \tag{4.6}$$

$$x_e + z_e \leq y_k, \ \ \forall e = \{i, j\} \in E, \ \ k = i \vee k = j \tag{4.7}$$

$$\sum_{e \in \delta(i)} (x_e + z_e) = 2y_i, \ \ \forall i \in V \tag{4.8}$$

$$x_e, z_e \geq 0, \ \ \forall e \in E \tag{4.9}$$

$$0 \leq y_i \leq 1, \forall i \in V. \tag{4.10}$$

A ideia central da formulação é que um caminho simples, representado pelas variáveis (x, y), seja unido à aresta adicional da variável z, formando o ciclo máximo. A restrição 4.4 corresponde à eliminação de subciclos ($GSEC$), impondo que o conjunto induzido pelas variáveis x e y seja de fato um caminho simples (*i.e.* não seja um ciclo). A restrição 4.5, por sua vez, define que o caminho simples das variáveis (x, y) possua no mínimo 2 arestas. Isto é necessário pois um ciclo simples apenas pode ser caracterizado com pelo menos 3 arestas (2 do caminho simples + 1 aresta da variável z). A restrição 4.6 impõe que z corresponda a apenas uma aresta, enquanto a restrição 4.7 não permite que z seja uma mesma aresta já presente no caminho simples. A restrição 4.8, por fim, define que cada vértice pertencente à solução do problema possua exatamente 2 arestas incidentes que também façam parte da solução. As demais restrições apenas impõem valores permitidos para (x,y,z).

4.2.1.2 Etapa Linear

Tendo em mãos um ciclo maximal do grafo de recursos G, devemos utilizar os passos descritos na demonstração do Teorema 1 para construirmos a orientação acíclica ω^* que levará o sistema à sua concorrência mínima. Inicialmente, devemos atribuir identificadores crescentes para cada vértice do ciclo κ^* em ordem horária ou anti-horária. Tendo concluído este passo, os demais vértices de G devem receber números de identificação crescentes a partir do valor $|\kappa^*| + 1$, a fim de estabelecer uma ordenação topológica para G. Ao final deste processo, orientamos todas as arestas de G na direção dos vértices de maiores identificadores, resultando em uma

Algoritmo 2: Um algoritmo de tempo linear para encontrar uma orientação acíclica que leve à concorrência mínima dado um ciclo maximal como *input*.

Input : Grafo não-direcionado $G = (V, E)$ e ciclo maximal $\kappa^* \subseteq V$

Output: Orientação acíclica ω^* para a qual $\gamma(\omega^*)$ é mínimo

$id = 1$

$v = \kappa^*.obterPrimeiroVertice()$

para *i=1 até* $\kappa^*.tamanho()$ **faça**

  Atribuir id a v

  Incrementar id

  v = $\kappa^*.obterVizinhoHorarioDe(v)$

fim

enquanto *um vértice* $v \in V$ *sem id existir* **faça**

  Atribuir id a v

  Incrementar id

fim

Criar uma orientação vazia ω^*

para cada *aresta não-direcionada* $uv \in E$ **faça**

  se $id(v) > id(u)$ **então**

    Orientar uv tal que $\omega^*(u, v) = v$

  fim

  senão

    Orientar uv tal que $\omega^*(u, v) = u$

  fim

fim

retorna ω^*

orientação acíclica que minimiza a expressão de concorrência. Tais passos estão formalizados sob a forma do Algoritmo 2.

A corretude do Algoritmo 2 se baseia na demonstração do Teorema 1. Nota-se que o tempo linear apenas é obtido caso o método $obterVizinhoHorarioDe(v)$ possua complexidade $O(1)$. Isto irá depender da estrutura de dados utilizada para guardar κ^*, que geralmente é um *array* contendo os vértices do ciclo na ordem em

que devem ser visitados. Neste caso, $obterVizinhoHorarioDe(v)$ simplesmente irá retornar o próximo elemento no *array*, cumprindo o requisito de tempo constante. Como G é um grafo conexo em que $|E| \geq |V|$, a complexidade global do Algoritmo 2 é $O(m)$, em que $m = |E|$.

4.2.2 Resultados Experimentais

Nesta subseção, discutiremos a implementação do algoritmo de duas etapas para obter orientações acíclicas que alcancem concorrência mínima. Como elaborado na subseção 4.2.1.1, devido ao fato do problema de decisão associado a encontrar um ciclo maximal de G ser NP-completo [8], a primeira fase do algoritmo consiste em empregar uma estratégia *branch-and-cut* para identificar um destes ciclos. A segunda fase, por sua vez, corresponde aos passos discutidos na subseção 4.2.1.2 e formalizados como parte do Algoritmo 2. Com base nos resultados experimentais obtidos, será possível mostrar que instâncias razoavelmente grandes, relacionadas ao problema de descontaminação de grafos discutido na subseção 3.2.1, podem ser resolvidas à otimalidade provada em tempos aceitáveis de *CPU*.

Toda a implementação foi realizada na linguagem C de programação. Para a primeira etapa, utilizamos o pacote *XPRESS Mixed Integer Programming* v8.5.3 para resolver problemas de programação linear e administrar a árvore de *branch-and-cut*. Todas as outras funcionalidades do programa (como pré-processamento, heurísticas primais e geração automática de corte) foram desabilitadas. Os experimentos foram conduzidos em uma máquina *Intel Core i9-8950HK* com 16 *Gbytes* de memória *RAM*, utilizando o sistema operacional *Linux Ubuntu 18.04.1* e apenas uma *thread*.

A fim de avaliar a viabilidade de nossa proposta, dois conjuntos de instâncias foram selecionados. Para o primeiro conjunto, sendo $n = |V|$, foram criados grafos circulantes (ver subseção 3.2.1.1) da forma $Ci_n(1, ..., \lfloor n/2 \rfloor)$, em que cada aresta foi gerada dada uma probabilidade p (*e.g.* para $p = 1$, será gerado o grafo completo K_n. Para $p = 0$, nenhuma aresta existe). Em caso de múltiplas componentes serem formadas, a estratégia empregada foi conectá-las aleatoriamente ao final do processo para garantir que nosso programa sempre gerasse grafos conexos. Esta natureza probabilística de geração das instâncias nos permitiu controlar a densidade dos grafos e capturar estruturas menos previsíveis.

Para o segundo conjunto de instâncias, a fim de avaliar a viabilidade da estratégia proposta para grafos que são conhecidamente mais desafiadores, foram selecionadas instâncias presentes no *website* do grupo de Otimização Discreta e Combinatória da Universidade de Heidelberg [34] referentes ao problema de encontrar ciclos hamiltonianos. No entanto, devido ao fato de termos limitado a execução de nosso programa para 3600 segundos, apenas foi possível resolver 2 das 9 instâncias disponíveis.

Dado que nossa intenção é apenas demonstrar viabilidade, todas as instâncias foram executadas uma única vez, com o tempo limite programado para 1 hora de execução e o tempo de *CPU* arredondado para o inteiro mais próximo. Os resultados para o primeiro e o segundo conjunto de instâncias são mostrados na tabela 4.1 e na tabela 4.2, respectivamente.

Nós	**Arestas**	p	$\lvert\kappa^*\rvert$	**Conc. Mín.**	**Tempo *CPU* (s)**
200	392	0.01	183	1/183	1
200	3826	0.1	200	1/200	2
1000	1912	0.002	882	1/882	169
1000	19912	0.02	1000	1/1000	552
1000	180151	0.2	-	-	> 3600
2000	4079	0.001	1807	1/1807	874
2000	40034	0.01	2000	1/2000	3599
2000	380147	0.1	-	-	> 3600
2000	1999000	1	-	-	> 3600

Tabela 4.1: Experimentos para encontrar a concorrência mínima de grafos circulantes gerados aleatoriamente.

Instância	**Nós**	**Arestas**	**Conc. Mín.**	**Tempo *CPU* (s)**
alb1000	1000	1998	1/1000	23
alb3000e	3000	5996	1/3000	2842

Tabela 4.2: Experimentos para encontrar a concorrência mínima de instâncias desafiadoras do problema de identificar ciclos hamiltonianos.

Capítulo 5

Conclusões

5.1 Considerações Finais

No contexto do compartilhamento de recursos entre processos, a dinâmica de *Escalonamento por Reversão de Arestas*, como aprofundada no capítulo 2, surge como uma estratégia natural para atribuir uma sequência de operação ao sistema. Dentre as aplicações discutidas no capítulo 3, as que se beneficiam de uma concorrência mínima entre processos provaram-se grandes inspirações para o desenvolvimento das técnicas descritas neste trabalho. Em especial, os resultados experimentais obtidos no capítulo 4 são de grande importância para sustentar a proposta nele desenvolvida, sendo o principal objeto de análise desta seção.

A tabela 4.1, incluída no capítulo 4, mostra que, apesar de grafos muito densos ainda serem um desafio, foi viável resolver instâncias consideravelmente grandes em menos de 1 hora. Isto apenas foi possível graças às técnicas modernas de otimização combinatória e à formulação relativamente simples do problema de encontrar ciclos maximais. Caso tentássemos modelar diretamente o problema de minimizar a métrica de concorrência, haveria grandes chances de que o modelo resultante fosse demasiadamente complexo a ponto de dificultar as podas da árvore de *branch-and-cut*. Ao recorrermos a outro problema relacionado e bem estudado na literatura para resolver um menos conhecido, podemos utilizar todo o conhecimento já produzido para alcançar tempos de execução bastante aceitáveis.

5.2 Trabalhos Futuros

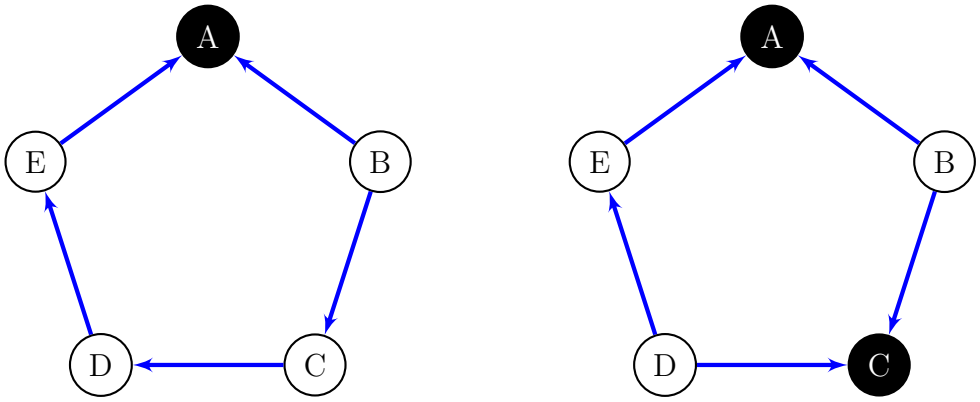
Nesta seção, serão discutidas possíveis direções de pesquisa para uma ampliação da literatura acerca de concorrência sob uma dinâmica de *Escalonamento por Reversão de Arestas*.

5.2.1 Concorrência Máxima

Um dos grandes desafios ainda em aberto é a elaboração de um modelo de otimização combinatória para maximizar a métrica de concorrência. Neste problema, a intuição indica que concorrências elevadas são atingidas ao orientarmos arestas intercaladamente, variando entre o sentido horário e o anti-horário. Algumas ideias já levantadas capturam a natureza do problema, mas não o solucionam: por exemplo, ao invés de olharmos para ciclos maximais (como é o caso da concorrência mínima), olharíamos para ciclos mínimos ímpares.

Por exemplo, seja κ um ciclo com um número par de vértices. Caso orientemos metade das arestas de κ no sentido horário e, a outra metade, no sentido anti-horário, obteríamos uma contribuição de $1/2$ referente a κ para a fração da equação 2.2. No entanto, seja κ' um ciclo com um número ímpar de vértices: neste caso, uma orientação intercalada de κ' poderia produzir valores de $n_{cw} = (|\kappa'| - 1)/2$ e $n_{cw} = (|\kappa'|+1)/2$. Por exemplo, se $|\kappa'| = 5$ com arestas orientadas intercaladamente, poderíamos obter $n_{cw} = 2$ e $n_{cw} = 3$. A fração interna da equação 2.2 retornaria uma contribuição de $2/5$ por parte de κ' e, se nenhum outro ciclo retornasse um valor menor, esta seria a concorrência do sistema. A intenção de buscarmos o ciclo mínimo se deve ao fato de que, neste caso em que $|\kappa'| = 5$, outros ciclos ímpares com mais vértices retornariam valores da fração interna da equação 2.2 maiores que $2/5$ (ex.: $3/7$), não impactando no minimizador que percorre todos os ciclos simples de G.

O desafio da proposta discutida acima encontra-se justamente na difícil garantia de que, ao orientarmos o restante dos ciclos de G, um valor menor para a fração interna da equação 2.2 seria evitado. Diferente do problema de concorrência mínima, orientar um ciclo em G pode prejudicar a orientação de outros ciclos adjacentes, impedindo que suas arestas sejam orientadas de forma intercalada. A figura 5.1



(a) Concorrência mínima. (b) Concorrência máxima.

Figura 5.1: Exemplos de orientações acíclicas que levam o sistema do Jantar dos Filósofos à sua concorrência mínima e máxima, com sumidouros em preto.

revisita o exemplo do Jantar dos Filósofos discutido na seção 1.1, mostrando orientações acíclicas iniciais que alcançam concorrência mínima e máxima.

5.2.2 Otimização de Parâmetros para o *Branch-and-Cut*

Apesar dos resultados presentes nas tabelas 4.1 e 4.2 serem considerados satisfatórios, acredita-se haver uma boa chance de ser possível resolver instâncias ainda maiores através de uma melhor manipulação dos diversos parâmetros do pacote *XPRESS Mixed Integer Programming*. Em especial, existem meta-algoritmos que utilizam aprendizado de máquina para buscar uma combinação de parâmetros que seja ideal para o problema específico sendo otimizado. Um exemplo é o *ParamILS* [35], um meta-algoritmo desenvolvido por pesquisadores da área de otimização combinatória que atingiu boas colocações em variadas competições.

5.2.3 Novas Aplicações para o *SER*

Apesar da lista de aplicações conhecidas para o *Escalonamento por Reversão de Arestas* possuir temas bastante variados, ainda há espaço para aumentá-la. Propor o uso do algoritmo em novos cenários (como na aviação) ou estender o conceito de descontaminação para outros ambientes (como em florestas) são alguns dos caminhos naturais para futuras pesquisas.

Referências Bibliográficas

[1] TANENBAUM, A. S., *Modern Operating Systems*. 3rd ed., pp. 143–165. Upper Saddle River, NJ, USA: Pearson Prentice Hall, 2007.

[2] CARVALHO, D., PROTTI, F., DE GREGORIO, M., et al., "A Novel Distributed Scheduling Algorithm for Resource Sharing Under Near-Heavy Load", *Lecture Notes in Computer Science*, v. 3544, pp. 431–442, 2004.

[3] ALVES, D. S. F., SOARES, E. E., STRACHAN, G. C., et al., *A Swarm Robotics Approach to Decontamination. In: Mobile Ad Hoc Robots and Wireless Robotic Systems: Design and Implementation.* 1st ed., pp. 107–122. Hershey, PA, USA: IGI Publishing Hershey, 2012.

[4] LENGERKE, O., ACUÑA, H. G., DUTRA, M. S., et al., "Distributed control of job-shop systems via edge reversal dynamics for automated guided vehicles", *1st International Conference on Intelligent Systems and Applications*, pp. 25–30, 2012.

[5] SABATTINI, L., DIGANI, V., SECCHI, C., et al., "Technological roadmap to boost the introduction of AGVs in industrial applications", *IEEE 9th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pp. 203–208, 2013.

[6] LIAO, C.-J., YOU, C.-T., "An Improved Formulation for the Job-Shop Scheduling Problem", *The Journal of the Operational Research Society*, v. 43, no. 11, pp. 1047–1054, 1992.

[7] GONÇALVES, V. C. F., LIMA, P. M. V., MACULAN, N., et al., "A Distributed Dynamics for WebGraph Decontamination", *Lecture Notes in Computer Science*, v. 6415, pp. 462–472, 2010.

[8] GAREY, M. R., JOHNSON, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1st ed., pp. 236–244. New York, NY, USA: W.H.Freeman & Co Ltd, 1979.

[9] CLAUSEN, J., *Branch and Bound Algorithms - Principles and Examples*, Tech. rep., Dept. Comput. Sci., Univ. Copenhagen, 1999.

[10] GAFNI, E., BERTSEKAS, D., "Distributed algorithms for generating loop-free routes in networks with frequently changing topology", *ACM Trans. on Comm.*, v. 29, no. 1, pp. 11–18, 1981.

[11] BARBOSA, V. C., GAFNI, E., "Concurrency in heavily loaded neighborhood-constrained systems", *ACM Trans. on Program. Lang. and Syst.*, v. 11, no. 4, pp. 562–584, 1989.

[12] BARBOSA, V. C., *An Atlas of Edge-Reversal Dynamics*. 1st ed., p. 6. Chapman and Hall/CRC, 2000.

[13] KNUTH, D. E., *The Art of Computer Programming, Vol. 1*. 3rd ed., pp. 262–263. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1997.

[14] ARANTES JR, G. M., *Trilhas, Otimização de Concorrência e Inicialização Probabilística em Sistemas sob Reversão de Arestas*, Ph.D. Thesis, Prog. de Eng. de Sist. e Comp., Univ. Fed. do Rio de Janeiro, 2006.

[15] ARANTES JR, G. M., FRANÇA, F. M. G., FARIA, L., *Concurrency in graphs with maximum degree 4: complexity and an approximation algorithm*, Tech. rep., Prog. de Eng. de Sist. e Comp., Univ. Fed. do Rio de Janeiro, 2005.

[16] ARANTES JR, G. M., FRANÇA, F. M. G., "Geração quase-instantânea de orientações acíclicas em sistemas distribuídos anônimos", *Anais do II Workshop em Sistemas Computacionais de Alto Desempenho*, pp. 55–62, Pirenópolis, GO, Brasil„ 2001.

[17] CALABRESE, A., "Distributed acyclic orientation of asynchronous anonymous networks", *11th Int. Symposium of Fund. of Comp. Theory*, pp. 129–137, Kraków, Poland, 1997.

[18] FRANÇA, F. M. G., *Sharing Rhythms: A Distributed Computing Approach*, Tech. rep., Prog. de Eng. de Sist. e Comp., Univ. Fed. do Rio de Janeiro, 2015.

[19] SOTSKOV, Y. N., SHAKHLEVICH, N., "NP-hardness of shop-scheduling problems with three jobs", *Discrete Applied Mathematics*, v. 59, no. 3, pp. 237–266, 1995.

[20] LUCCIO, F., PAGLI, L., "Web Marshals Fighting Curly Link Farms", *Lecture Notes in Computer Science*, v. 4475, pp. 240–248, 2007.

[21] BECCHETTI, L., CASTILLO, C., DONATO, D., et al., "Link-Based Characterization and Detection of Web Spam", *SIGIR Forum*, v. 42, pp. 68–72, 2006.

[22] MOSCARINI, M., PETRESCHI, R., SZWARCFITER, J. L., "On node searching and starlike graphs", *Congressus Numerantium*, v. 131, pp. 75–84, 1998.

[23] KIROUSIS, L. M., PAPADIMITRIOU, C. H., "Searching and pebbling", *Lecture Notes in Computer Science*, v. 47, pp. 205–218, 1986.

[24] BARBOSA, V. C., *An Introduction to Distributed Algorithms*. 1st ed., pp. 95–113. Cambridge, MA, USA: The MIT Press, 1996.

[25] SHAN, M.-K., CHIU, S.-C., "Algorithmic compositions based on discovered musical patterns", *Multimedia Tools and Applications*, v. 46, n. 1, pp. 1–23, Jan. 2010.

[26] NIERHAUS, G., *Algorithmic Composition: Paradigms of Automated Music Generation*. Springer-Verlag: Vienna, Austria, 2009.

[27] FERNANDEZ, J. D., VICO, F., "AI Methods in Algorithmic Composition: A Comprehensive Survey", *Journal of Artificial Intelligence Research*, v. 48, n. 1, pp. 513–582, Nov. 2013.

[28] SCHMIDT-JONES, C., *Understanding Basic Music Theory*. OpenStax CNX: Houston, TX, USA, 2007.

[29] KRISHNAMOORTHY, M. S., "An NP-hard problem in bipartite graphs", *SIGACT News*, v. 7, n. 1, pp. 26–26, Jan. 1975.

[30] ALMENDE, B. V., "vis.js - A dynamic, browser based visualization library", acessado em 15/02/2019. Disponível em: `http://visjs.org/`.

[31] SIMPSON, J., "howler.js - JavaScript audio library for the modern Web", acessado em 15/02/2019. Disponível em: `https://howlerjs.com/`.

[32] MITCHELL, J. E., *Branch-and-Cut Algorithms for Combinatorial Optimization Problems. In: Handbook of Applied Optimization*. 1st ed., pp. 65–77. Oxford, UK: Oxford University Press, 2002.

[33] LUCENA, A., DA CUNHA, A. S., SIMONETTI, L., "A New Formulation and Computational Results for the Simple Cycle Problem", *Electronic Notes in Discrete Mathematics*, v. 44, no. 5, pp. 83–88, 2013.

[34] DISCRETE & COMBINATORIAL OPTIMIZATION GROUP, H. U., "Hamiltonian cycle problem data", acessado em 25/11/2018. Disponível em: `https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/hcp/`.

[35] HUTTER, F., HOOS, H. H., STÜTZLE, T., "Automatic algorithm configuration based on local search", *AAAI'07 Proceedings of the 22nd National Conference on Artificial Intelligence*, v. 2, pp. 1152–1157, 2007.

Apêndice A

Simulação Musical

A simulação musical citada na subseção 3.2.3 está disponível no seguinte *website:* `https://carloseduardov8.github.io/Song-Generator/`

Esta simulação é um projeto de código aberto distribuído sob a *Licença GNU GPL v3.0*. O código-fonte está disponível no seguinte *website:* `https://github.com/carloseduardov8/Song-Generator/`