LETÍCIA RODRIGUES BUENO

Sobre Redução de Cruzamentos de Arestas em Desenho Linear

Maringá - PR

Outubro de 2005

LETÍCIA RODRIGUES BUENO

Sobre Redução de Cruzamentos de Arestas em Desenho Linear

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual de Maringá, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador:
Candido Ferreira Xavier de Mendonça Neto

Maringá - PR

Outubro de 2005

Dedico este trabalho

à Jesus Cristo

Porque dEle, por Ele e para Ele são todas as coisas.

Agradecimentos

Ao meu orientador Xavier e sua esposa Marilice pelo incentivo, confiança e amizade.

Aos amigos Edmundo, Marco Aurélio, Caetano e colegas do Curso de Mestrado pelos conselhos e discussões úteis.

Aos funcionários e professores do Departamento de Informática pelo apoio e auxílio.

Ao professor Ademir Constantino pela paciência e dedicação no ensino de Otimização Combinatória que muito contribuiu nos trabalhos de dissertação. Também ao professor Maurício Figueiredo pelo auxílio e artigos em Redes Neurais.

Aos meus pais, José Augusto e Rosa, e aos meus irmãos Lívia, Michel e Maicon pelo incentivo, apoio e paciência durante a elaboração deste trabalho.

A todos, meus sinceros agradecimentos.

"...you do not fail in obedience through lack of love, but have lost love because you never attempted obedience."
C. S. Lewis, That Hideous Strength

Resumo

Um invariante de não-planaridade é uma medida de quão distante um grafo está em admitir um desenho planar. O invariante mais estudado é o *crossing number* e, por isso, é vasta a quantidade de trabalhos encontrados na literatura sobre este tema. Por esta razão, este trabalho investiga algumas abordagens para redução do número de cruzamentos de arestas no desenho linear de grafos, onde os vértices são dispostos ao longo de uma linha reta horizontal no plano. Um algoritmo baseado na meta-heurística Times Assíncronos é apresentado com resultados interessantes e razoável tempo de execução.

Palavras-chave: planarização de grafos, número de cruzamentos de arestas, *st*-numeração, times assíncronos, vértices falsos, desenho linear de grafos.

Abstract

A nonplanarity invariant is a measure of how nonplanar is a given graph. The most investigated invariant is the crossing number which can be observed by the amount of works in the literature on this subject. Therefore, this work investigates approaches to reduce the edges crossing number in linear graph drawing. An algorithm based in the meta-heuristic Asynchronous Teams is presented with interesting results and a reasonable time of execution.

Keywords: graph planarization, crossing number, *st*-numbering, asynchronous teams, dummy vertices, layout linear.

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

Grafos freqüentemente têm sido utilizados para representar a informação e, em diversos casos, eles são a melhor e, às vezes, a única forma de transmitir informação de maneira compreensível. Podemos citar, por exemplo, os mapas utilizados para indicar percursos e distâncias entre cidades e os autômatos finitos. No primeiro caso, as cidades são representadas pelos vértices e as estradas representadas pelas arestas do grafo, onde as arestas possuem informação geográfica. No segundo caso, estados são representados por vértices e transições são representadas por arestas, onde as arestas possuem informação sobre a transição.

Desenho de Grafos é uma área de pesquisa em Teoria dos Grafos que estuda técnicas para visualização de informação formulada sob a forma de grafos. Existem diversos critérios estéticos que procuram traduzir as características que tornam um desenho "agradável" e "legível" para o usuário. Normalmente, os grafos são considerados legíveis quando atendem a critérios tais como: simetria, poucos cruzamentos de arestas, distribuição uniforme de vértices e comprimento uniforme de arestas. Embora a avaliação do que é "legível" e o que é "agradável" em um desenho seja subjetivo a cada usuário, é possível definir alguns critérios e padrões úteis à maior parte das aplicações, dentre esses (critérios e padrões) a planaridade de grafos ocupa lugar de destaque. Este fato atraiu uma generosa quantidade de trabalhos dedicados a grafos planares e, em alguns casos, envolvendo métodos de planarização de grafos. Entre as aplicações de planarização de grafos destacamos o projeto de circuitos impressos e projeto de circuitos VLSI (*Very Large Scale of Integration*).

Como exemplo prático, mostramos na Figura 1.1 o circuito impresso de um *floppy* com cerca de 13 cruzamentos, os quais foram substituídos por um "*jumper*" (veja Figura 1.2). No destaque da Figura 1.1, vemos a substituição de um cruzamento pelo *jumper* $P1$.

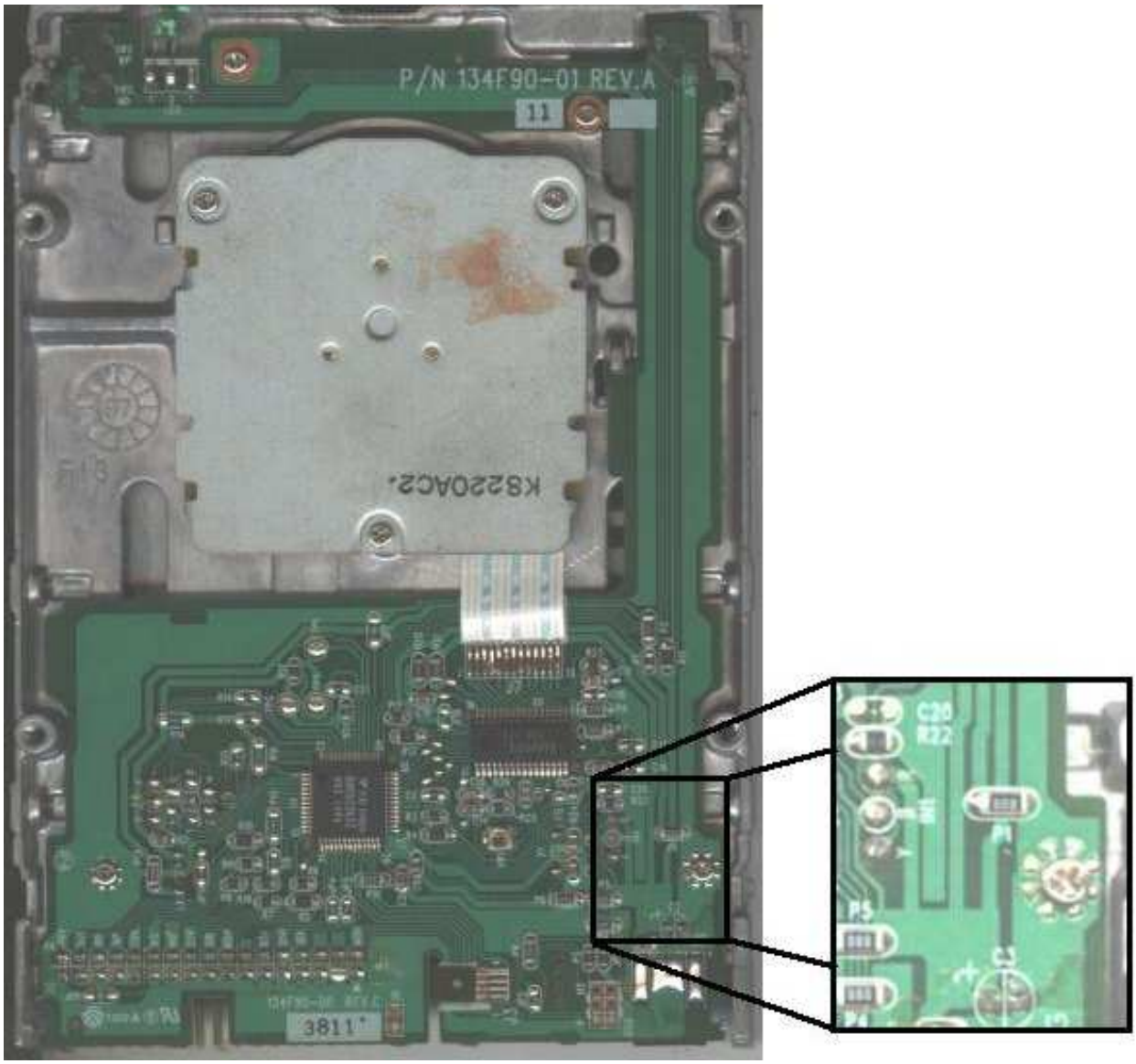


Figura 1.1: Placa de circuito impresso de um *floppy*

No layout de placas de circuito impresso bem como no projeto de circuitos VLSI, para o caso de fios não isolados, a sobreposição de fios entre componentes elétricos causaria curto-circuito. Uma solução possível é substituir o cruzamento por um dispositivo como mostrado na Figura 1.2. Desta forma, ao reduzirmos o número de cruzamentos nestes circuitos, diminuímos o número desses dispositivos numa placa.

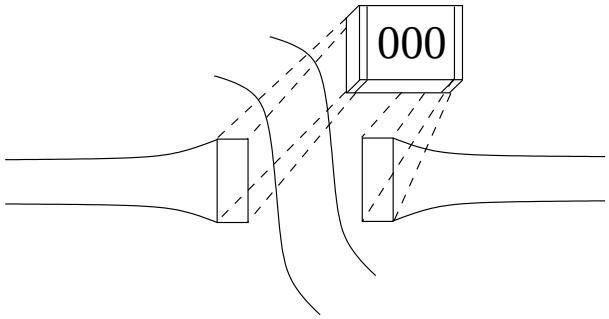


Figura 1.2: *Jumper*

Neste trabalho, investigamos abordagens para redução do número de cruzamentos de arestas em desenho linear de grafos. Para a otimização do problema utilizamos a meta-heurística Times Assíncronos.

No próximo Capítulo apresentamos os conceitos básicos utilizados no decorrer do trabalho. No Capítulo 3 apresentamos um Time Assíncrono para otimização do problema do número de cruzamentos em desenho linear. No Capítulo 4 discutimos alguns resultados obtidos pela execução do Time e, finalmente, no último Capítulo concluímos este trabalho de dissertação apresentando algumas idéias de trabalhos futuros.

2 Fundamentação Teórica

Neste capítulo, apresentamos conceitos básicos utilizados no decorrer do trabalho. O leitor habituado às definições básicas de grafos pode saltar para a seção 2.2.

2.1 Grafos

Um *grafo* G é uma tripla ordenada $G=(V,E,\psi)$, onde V é um conjunto finito e não vazio de pontos denominados *vértices*, E é um conjunto finito de *arestas* e $\psi : E \to V \times V$ é uma função de incidência que associa a cada aresta e de G um par de vértices u e v, notação $\psi(e) = \{u, v\}$ ou $\psi(e) = \{v, u\}$, onde a ordem não é importante. Os vértices u e v são chamados de *extremos* de e. Neste caso, dizemos que o vértice u é *vizinho* de v e vice-versa, a aresta e é *incidente* em u e em v e que o par de vértices u e v são *adjacentes*.

Um grafo contém arestas *paralelas* ou *múltiplas* se possui arestas diferentes compartilhando os mesmos extremos. Se a função de incidência admite arestas com vértices u e v iguais ($u=v$), então, o grafo contém *laços*. Um grafo é *simples* se não contém laços nem arestas múltiplas.

Neste trabalho utilizamos apenas grafos simples, o que denotaremos simplesmente por grafos. Como a função de incidência está bem definida pelos extremos de cada aresta, omitiremos a função de incidência da definição de grafos. Portanto, um grafo é uma dupla $G=(V,E)$ tal que uma aresta $e = \{u, v\}$ onde anteriormente $\psi(e) = \{u, v\}$.

Definimos como *grau* de um vértice, denotado por $d(u)$, o tamanho do conjunto dos vértices adjacentes ao vértice u. Um grafo G é dito *regular* se todos os seus vértices tem

o mesmo grau.

Um grafo *G=(V,E)* é dito ser *cúbico* quando para todo vértice u em G, $d(u) = 3$. A Figura 2.1 ilustra um grafo cúbico.

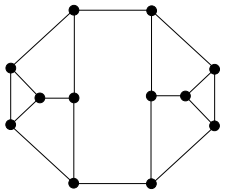


Figura 2.1: Exemplo de grafo cúbico

Seja $G = (V, E)$ um grafo com $|V| = 2^n$ vértices, numerados $1, 2, \ldots, 2^n$. Seja l_v a representação binária com n dígitos da numeração de cada vértice $v \in V$. O grafo G é chamado de *n-cubo*, denotado por Q_n, se o conjunto de arestas consiste dos pares $\{u, v\}$ tal que l_u difere de l_v em exatamente um dígito. A Figura 2.2 apresenta os grafos Q_1, Q_2, Q_3 e Q_4.

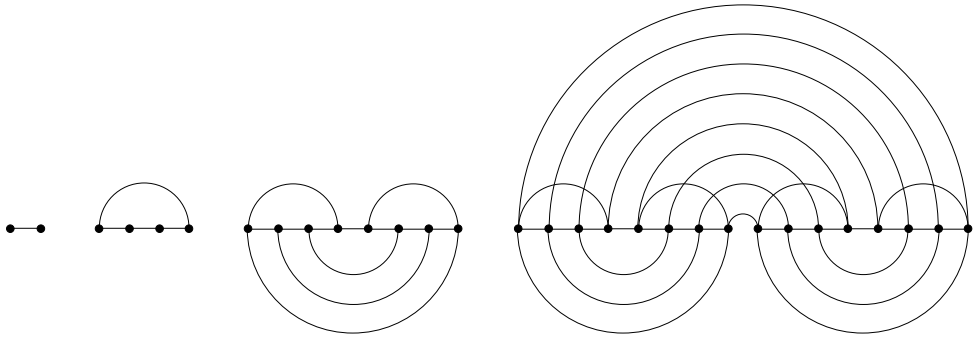


Figura 2.2: Grafos Q_1, Q_2, Q_3 e Q_4

A *vizinhança* de um vértice u, denotada por *N(u)*, consiste no conjunto de vértices adjacentes a u, conforme ilustra a Figura 2.3 (neste trabalho não incluímos u em sua vizinhança).

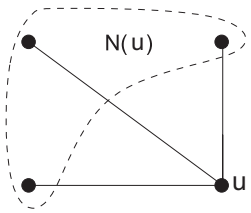


Figura 2.3: Vizinhança de um grafo

Um grafo é dito *completo*, denotado por K_n, se todos os seus vértices são dois a dois adjacentes. A Figura 2.4 ilustra um grafo completo com 4 vértices (K_4).

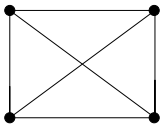


Figura 2.4: Grafo completo K_4

Um grafo *G=(V,E)* é dito *bipartido* se existe uma bipartição de V em dois subconjuntos V_1 e V_2, de maneira que toda aresta e em E tem exatamente um extremo em V_1 e um extremo em V_2. O grafo G será chamado de *bipartido completo*, denotado por $K_{n,m}$, se $|V| = n + m$, $|V_1| = m$, $|V_2| = n$ e $N(v) = V_2$ para todo vértice $v \in V_1$. A Figura 2.5 mostra os grafos $K_{2,2}$ e $K_{3,3}$.

(a) $K_{2,2}$ (b) $K_{3,3}$

Figura 2.5: Grafos completos bipartidos $K_{2,2}$ e $K_{3,3}$

Chamamos de *passeio* P em um grafo G uma seqüência de vértices e arestas $P = (u = u_1, \{u_1, u_2\}, u_2 \ldots, u_{k-1}, \{u_{k-1}, u_k\}, u_k = v)$. Um *caminho* é um passeio que não repete vértices. Dizemos que dois caminhos C_1 e C_2 entre u e v são *disjuntos* se $(C_1 \cap C_2) \backslash \{u, v\} = \varnothing$. Seja u e v vértices distintos, seja também C_1 e C_2 caminhos disjuntos entre u e v, então, chamamos de *ciclo* a união $C_1 \cup C_2$.

Corolário 1. *Todo ciclo em um grafo sem arestas múltiplas tem, pelo menos, 3 vértices.*

Demonstração: Segue trivialmente das definições de arestas múltiplas e de ciclo. □

Se um caminho contém todos os vértices do grafo dizemos que este caminho é *hamiltoniano*. Um ciclo é *hamiltoniano* se contém todos os vértices de G.

O produto cartesiano $C_n \times C_m$ de dois ciclos C_n e C_m é o grafo contendo nm vértices $\{v_{i,j}\}$ e $2nm$ arestas $\{v_{i,j}, v_{(i+1) \ mod \ n,j}\}$ e $\{v_{i,j}, v_{i,(j+1) \ mod \ m}\}$, para $0 \leq i < n$ e $0 \leq j < m$. Considerando que cada vértice do $C_n \times C_m$ é representado por um ponto no plano com coordenadas (i, j), chamamos as duas famílias de arestas acima de *horizontal* e *vertical*, respectivamente. Um ciclo do grafo $C_n \times C_m$ é chamado *meridiano* se usa apenas arestas verticais e *paralelo* se usa apenas arestas horizontais. Assim, o grafo $C_n \times C_m$ possui n meridianos isomorfos ao C_m e m paralelos isomorfos ao C_n. A Figura 2.6 apresenta o grafo $C_3 \times C_3$.

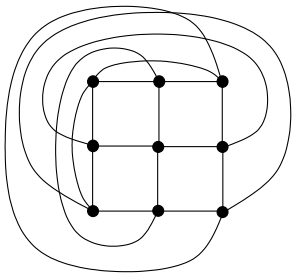


Figura 2.6: Grafo $C_3 \times C_3$

Um grafo G é dito *conexo* se existe um caminho entre quaisquer vértices de G, caso contrário, dizemos que G é *desconexo*. Um vértice $v \in G$ é chamado de *ponto de articulação* ou *vértice de corte* se G é conexo e $G \backslash v$ é desconexo. Dizemos que G é *biconexo* se G não possui pontos de articulação. A Figura 2.7 ilustra a estrutura de um grafo não biconexo. São pontos de articulação os vértices a e b.

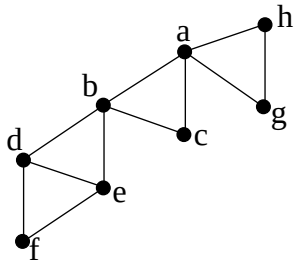


Figura 2.7: Estrutura de um grafo não biconexo

Um grafo *G'=(V',E')* é chamado de *subgrafo* de *G=(V,E)* se $V' \subseteq V$ e se $E' \subseteq E$ (Figura 2.8). Dizemos que G' é *subgrafo próprio* se $V' \neq V$ ou $E' \neq E$, notação $G' \subsetneq G$. Dizemos que $G' \subsetneq G$ é um *subgrafo induzido* de G se para todo par de vértices u e v em

G', $\{u, v\} \in E'$ se e somente se $\{u, v\} \in E$.

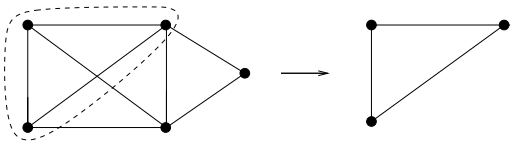


Figura 2.8: Subgrafo

Dizemos que G' é uma *componente biconexa* (ou *bloco*) de G, se G' é um subgrafo induzido de G e G' é biconexo.

Seja P uma propriedade qualquer de um grafo G. Dizemos que $H \subseteq G$ é *maximal* (*resp., minimal*) relativo à propriedade P se não existe um subgrafo $H' \subseteq G$ que possui a propriedade P onde $H \subsetneq H'$ (resp., $H' \subsetneq H$). A definição não implica necessariamente que H seja o maior (resp., menor) subgrafo de G com a propriedade P. Quando isso ocorre, o subgrafo H é dito ser *máximo* (resp., *mínimo*) relativo à propriedade P. Na Figura 2.9, considerando a definição de subgrafo completo como a propriedade P a ser verificada, o subgrafo induzido pelo conjunto de vértices $\{u, v\}$ é completo maximal, porém não é completo máximo, já o subgrafo induzido por $\{x, y, z\}$ é completo máximo.

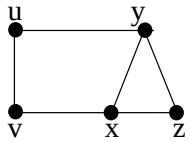


Figura 2.9: Subgrafo maximal e minimal

Dois grafos G e H são chamados de *isomorfos* se existe uma bijeção $\varphi : G \to H$ tal que u e v são adjacentes em G se e somente se os vértices $\varphi(u)$ e $\varphi(v)$ são adjacentes em H.

A *subdivisão de uma aresta* $e = \{u, v\}$ é a operação em que se remove e e se insere um novo vértice w e duas arestas $\{u, w\}$ e $\{w, v\}$. Um grafo G contém uma subdivisão de um grafo H se existe uma subdivisão do grafo H isomorfa a um subgrafo de G.

Dizemos que um grafo conexo G é uma *árvore* se G não contém ciclos. Uma proposição fácil de demonstrar é que em uma árvore T existe um único caminho entre quaisquer dois

(a) Grafo G (b) Árvore de expansão T

Figura 2.10: Árvore de expansão de um grafo. Arestas de ciclo tracejadas

vértices v e $w \in T$. Uma árvore é dita *enraizada* (T, r) se escolhemos aleatoriamente algum vértice $r \in T$ ao qual chamamos de *raiz*. Dada uma árvore enraizada (T, r) e um par de vértices v e w quaisquer, dizemos que v é *ancestral* de w e w é um *descendente* de v se v pertence ao caminho de r para w. Se v é ancestral de w e $\{v, w\}$ é uma aresta de T, então dizemos que v é *pai* de w e que w é *filho* de v. Uma árvore enraizada (T, r) é uma *árvore de expansão* de um grafo G se T é um subgrafo de G e existe um vértice r em T tal que para toda aresta $e = \{u, v\}$, ou $e \in T$ ou u é ancestral de v ou v é ancestral de u em (T, r). Neste caso as arestas de $G \setminus T$ são chamadas de *arestas de retorno*, denotadas por $u \hookrightarrow v$, e as demais arestas de *arestas da árvore*, denotadas por $u \rightarrow v$. Denotamos por $u \xrightarrow{*} v$ um caminho C de u para v (possivelmente vazio) quando todas as arestas de C são arestas de (T, r).

Arestas de árvore	**Arestas de ciclo**	**Caminhos**
$a \rightarrow b$ $\quad c \rightarrow e$	$f \hookrightarrow a$	$a \xrightarrow{*} c$ $\quad b \xrightarrow{*} d$ $\quad c \xrightarrow{*} f$
$b \rightarrow c$ $\quad e \rightarrow f$	$d \hookrightarrow b$	$a \xrightarrow{*} d$ $\quad b \xrightarrow{*} e$
$c \rightarrow d$	$e \hookrightarrow a$	$a \xrightarrow{*} e$ $\quad b \xrightarrow{*} f$
		$a \xrightarrow{*} f$

Tabela 2.1: Caminhos e arestas da árvore da Figura 2.10

Um *desenho simples* de um grafo G é um desenho de G no plano tal que vértices são desenhados em lugares distintos, nenhuma aresta cruza a si mesma, arestas adjacentes não se cruzam, duas arestas caso se cruzem se cruzam no máximo uma única vez e no seu interior, nunca nos extremos, arestas não interceptam vértices, exceto quando incidentes nestes vértices e um cruzamento somente pode ser compartilhado por duas arestas. Dora-

vante neste trabalho quando mencionarmos desenho de um grafo subentendemos desenho simples.

Um grafo é dito *planar* se admite um desenho simples sem cruzamentos de arestas.

Teorema 2 (Kuratowski [47]). *Um grafo G é planar se e somente se não contém uma subdivisão do $K_{3,3}$ ou do K_5.*

Dizemos que um desenho (simples) é *ótimo* se apresenta o menor número possível de cruzamentos.

Chamamos de *número de cruzamentos* (*crossing number*) de G, denotado por $cr(G)$, o número mínimo de cruzamentos de arestas de um desenho ótimo de G. Este número é particularmente interessante, pois o valor é o mesmo independente do modo como G é desenhado.

2.2 Invariantes de Não-Planaridade

Chamamos o número de cruzamentos de arestas um *invariante de não-planaridade*, isto é, qualquer que seja o desenho D_G de um grafo G, o número de cruzamentos de arestas de D_G é maior ou igual a $cr(G)$, o que explica o nome "invariante". Denotamos ainda o número de cruzamentos de D_G por $cd(D_G)$. Desta forma, $cd(D_G) \geq cr(G)$, com igualdade se D_G é ótimo.

Uma outra maneira de definir um invariante de não-planaridade é associá-lo a uma operação de planarização. Uma *operação de planarização* é uma operação em um grafo não planar G que resulta em um novo grafo G' com o número de cruzamentos do desenho ótimo menor ($cr(G') < cr(G)$).

Um exemplo de operação de planarização é a inserção de vértices falsos. Seja G um grafo não planar, seja D_G um desenho ótimo de G, sejam $e_1 = \{u, v\}$ e $e_2 = \{x, y\}$ duas arestas que se cruzam em D_G. Uma operação de *inserção de vértices falsos* é a criação de um novo grafo G' com a remoção de e_1 e e_2 de G e a inserção de um novo vértice k e das

arestas $\{u, k\}$, $\{k, v\}$, $\{x, k\}$ e $\{k, y\}$, onde $D_{G'}$ pode ser formado colocando o vértice k nas coordenadas do cruzamento de e_1 e e_2 em D_G, colocando-se também as novas arestas no local correspondente às linhas entre o cruzamento de e_1 e e_2 e os extremos de e_1 e e_2, como mostra a Figura 2.11. Neste caso, o grafo resultante G' possui um desenho $D_{G'}$ com um cruzamento a menos que D_G e o vértice k é chamado de *vértice falso*. Obviamente, o número mínimo de vértices falsos a ser acrescentados que resultem em um grafo planar é igual ao número de cruzamentos do desenho ótimo do grafo. Portanto, o número de cruzamentos de arestas é o invariante associado à operação de inserção de vértices falsos.

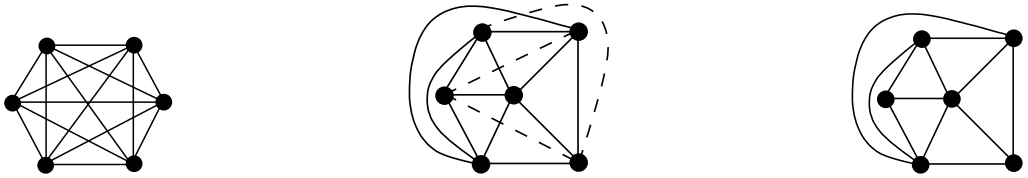
(a) Cruzamento entre e_1 e e_2 em D_G (b) Vértice k e seus adjacentes em $D_{G'}$

Figura 2.11: Exemplo da operação de inserção de vértice falso

O número de cruzamentos de arestas é, historicamente, o invariante de não-planaridade mais antigo, entretanto, não é o único.

Similarmente, podemos planarizar um grafo G através da remoção de algumas arestas (veja Figura 2.12). Este invariante é conhecido como *número de remoção de arestas* (*skewness*), denotado por $sk(G)$, e indica o número mínimo de arestas que devem ser removidas de G resultando em um grafo planar. Assim, dado um grafo $G = (V, E)$, seja $G' = (V, E')$ um subgrafo planar de G tal que não existe um outro subgrafo $G'' = (V, E'')$ de G com $|E''| > |E'|$. Neste caso, G' é chamado de subgrafo planar máximo de G e o número de arestas removidas $|E| - |E'| = sk(G)$ é chamado de número de remoção de arestas de G.

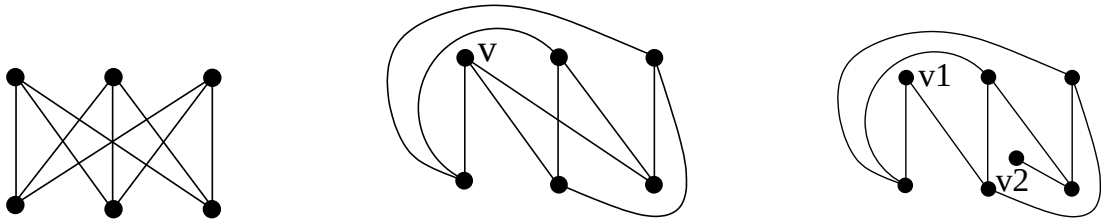
O invariante de não-planaridade *número de quebra de vértices* (*splitting number*), denotado por $sp(G)$, representa o número mínimo de operações de divisão de vértices que devem ser realizadas em um grafo G resultando num grafo G' planar. Dado um grafo



(a) $cd(D_{K_6}) = 14$ (b) $cr(K_6) = cd(D'_{K_6}) = 3$ e $sk(K_6) = 3$ (c) $cd(D''_{K_6}) = 0$

Figura 2.12: Exemplo da operação de remoção de arestas para o K_6

$G = (V, E)$, uma operação de quebra de um vértice $v \in V$ particiona $N(v)$ em dois conjuntos não vazios S_1 e S_2 e adiciona a $G \setminus v$ dois vértices novos não adjacentes v_1 e v_2, tal que $S_1 = N(v_1)$ e $S_2 = N(v_2)$ (veja Figura 2.13).

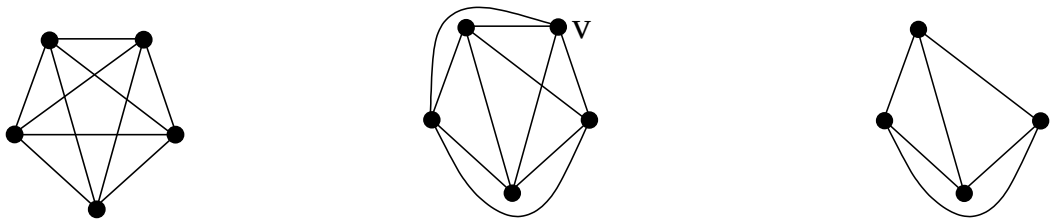


(a) $cd(D_{K_{3,3}}) = 8$ (b) $cr(K_{3,3}) = cd(D'_{K_{3,3}}) = 1$ e $sp(K_{3,3}) = 1$ (c) $cd(D''_{K_{3,3}}) = 0$

Figura 2.13: Exemplo da operação de quebra de vértices para o $K_{3,3}$

Outro invariante de não-planaridade é o *número de remoção de vértices* (*vertex deletion number*), denotado por $vd(G)$, que indica o número mínimo de vértices que precisam ser removidos para planarizar um grafo G (veja um exemplo na Figura 2.14). Neste caso, a remoção de um vértice implica também na remoção das arestas nele incidentes resultando num subgrafo induzido, ou seja, dado um grafo $G = (V, E)$, seja $G' = (V', E')$ um subgrafo planar induzido de G tal que não exista um subgrafo planar induzido $G'' = (V'', E'')$ de G com $|V''| > |V'|$, então G' é chamado de subgrafo planar induzido máximo de G, e a diferença $|V| - |V'| = vd(G)$ é chamado de número de remoção de arestas de G. Novamente, este número é um invariante pois qualquer remoção de vértices que torna um grafo G planar tem, no mínimo, $vd(G)$ vértices removidos.

O problema de decisão associado à cada um dos invariantes relacionados é NP-Completo [29, 33, 34, 77], permanecendo NP-Completo mesmo para grafos cúbicos [27, 29, 30, 40]. Recentemente, foi demonstrado que os invariantes número de quebra de



(a) $cd(D_{K_5}) = 5$ (b) $cr(K_5) = cd(D'_{K_5}) = 1$ e $vd(K_5) = 1$ (c) $D''_{K_5} = D'_{K_5} \setminus v$, $cd(D''_{K_5}) = 0$

Figura 2.14: Exemplo da operação de remoção de vértices para o K_5

vértices, número de remoção de vértices e número de remoção de arestas são também SNP-Difíceis [12, 13, 30], mesmo para grafos cúbicos [13, 30].

Neste trabalho nos concentramos no número de cruzamentos de arestas / inserção de vértices falsos.

Os valores exatos conhecidos dos invariantes de não-planaridade são restritos a poucas classes de grafos que, em geral, são grafos simétricos ou com características regulares, tais como os grafos K_n, $K_{n,m}$, Q_n e $C_n \times C_m$. Mesmo restritos a estas classes de grafos, existem poucos resultados exatos para o número de cruzamentos de arestas. Frequentemente, somente limites inferiores e superiores são conhecidos. Na Tabela 2.2 apresentamos os valores conhecidos de número de cruzamentos para as classes de grafos K_n, $K_{n,m}$, Q_n e $C_n \times C_m$.

Uma descrição da história do resultado do número de cruzamentos para os grafos K_n pode ser encontrada em [37, 38]. Um limite inferior foi também proposto por Leighton [48]: $cr(K_n) \geq \frac{1}{120}n(n-1)(n-2)(n-3)$, para $n \geq 5$.

Quanto aos grafos $K_{n,m}$, Zarankiewicz [78] afirmou que $cr(K_{n,m}) = \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor \lfloor \frac{m}{2} \rfloor \lfloor \frac{m-1}{2} \rfloor$. Porém, um erro foi encontrado nesta demonstração e, assim, a conjectura de Zarankiewicz tornou-se apenas um limite superior (veja em [36] um histórico deste problema). Mais tarde, Kleitman [45] provou a conjectura de Zarankiewicz para $min(n, m) \leq 6$ e Woodall [74] para $n \leq 8$, $m \leq 10$. Além disso, limites inferiores para os grafos $K_{n,m}$ têm sido estudados [14, 45, 54].

Em relação aos grafos Q_n, foi estabelecida uma conjectura por Eggleton e Guy [23]:

Número de cruzamentos	
Classe	**Valores**
K_n	$\leq \frac{1}{4}\lfloor\frac{n}{2}\rfloor\lfloor\frac{n-1}{2}\rfloor\lfloor\frac{n-2}{2}\rfloor\lfloor\frac{n-3}{2}\rfloor$, para $n \leq 10$ $\geq \frac{1}{120}n(n-1)(n-2)(n-3)$, para $n \geq 5$
$K_{n,m}$	$\leq \lfloor\frac{n}{2}\rfloor\lfloor\frac{n-1}{2}\rfloor\lfloor\frac{m}{2}\rfloor\lfloor\frac{m-1}{2}\rfloor$, para $min(n,m) \leq 6$ e $7 \leq m \leq 10$, $7 \leq n \leq 8$ $9\lfloor\frac{m-1}{2}\rfloor\lfloor\frac{m}{2}\rfloor$, para $n = 7$ $\begin{cases} m \text{ ímpar}, m \geq 7 \\ m \text{ par}, m \geq 8 \end{cases}$
Q_n	$\leq \frac{5}{32}4^n - \lfloor\frac{n^2+1}{2}\rfloor 2^{n-2}$, para $n \geq 3$
$C_n \times C_m$	$\leq (n-2)m$, para $3 \leq n \leq m$

Tabela 2.2: Número de cruzamentos do K_n, $K_{n,m}$, Q_n e $C_n \times C_m$

$cr(Q_n) \leq 4^n\frac{5}{32} - 2^{n-2}\lfloor\frac{n^2+1}{2}\rfloor$. Posteriormente, Guy [37] encontrou um intervalo nesta conjectura. Entretanto, Erdös e Guy [24] conjecturaram igualdade.

Mais tarde, Madej [51] propôs um limite superior para o número de cruzamentos do grafo n-cubo: $cr(Q_n) \leq 4^n\frac{1}{6} - 2^{n-3}n^2 - 2^{n-4}3 + (-2)^n\frac{1}{48}$. Além disso, Madej mostrou que $cr(Q_5) \leq 56$ confirmando, assim, a conjectura de Eggleton e Guy. Então, Sýkora e Vrťo [67] provaram que o limite de Madej é assintoticamente ótimo. Dean e Richter [17] mostraram que o número de cruzamentos do Q_4 é 8, confirmando a conjectura de Eggleton e Guy [23] para $n = 4$. Este resultado é não trivial e, até o momento, é o único resultado exato para número de cruzamentos de grafos Q_n.

A conjectura de Eggleton e Guy [23] também foi confirmada por Faria e Figueiredo [28] para $n = 6, 7$ e 8. Os desenhos dos grafos Q_6, Q_7 e Q_8 foram apresentados em [27]. Além disso, Faria e Figueiredo [28] estabeleceram um novo limite superior para grafos Q_n onde $n \geq 7$: $cr(Q_n) \leq \frac{165}{1024}4^n - \frac{2n^2-11n+34}{2}2^{n-2}$. Recentemente, um resultado de Faria et al. [31] propõe $cr(Q_n) \leq \frac{5}{32}4^n - \lfloor\frac{n^2+1}{2}\rfloor 2^{n-2}$, para $n \geq 3$, coincidindo com a conjectura de Erdös

e Guy [24].

Em número de cruzamentos de grafos $C_n \times C_m$, foi proposto por Harary et al. [39] a conjectura que $cr(C_n \times C_m) = (n-2)m$, para todo n, m onde $3 \leq n \leq m$. Esta conjectura têm sido provada para casos particulares [2, 3, 4, 5, 17, 46, 57, 58, 59, 75, 76]. O resultado de Salazar [61] confirma a conjectura, baseando-se no comportamento assintótico dos números mínimos de cruzamentos de grandes classes de desenhos para $C_n \times C_m$. Recentemente, Glebsky e Salazar [35] provaram que a conjectura é válida para valores de n suficientemente grandes quando comparados a m (aproximadamente $n \geq m^2$). Além disso, limites inferiores para os grafos $C_n \times Cm$ foram propostos em [43, 62, 65].

Mais informações sobre os invariantes de não-planaridade, recomendamos ao leitor o trabalho de Liebers [50] e de Shahrokhi et al. [63].

Na próxima seção, apresentamos os conceitos e trabalhos relacionados à um dos problemas de representação de grafos em livros. No Capítulo 3, um algoritmo é proposto para otimização do problema em questão.

2.3 Desenho Linear de Grafos

Uma vez que o problema de decisão associado ao número de cruzamentos de arestas é NP-Completo, há na literatura uma vasta discussão sobre a otimização do número de cruzamentos de arestas por meio de uma representação em livros que, segundo Shahrokhi et al. [63], tem resultado em métodos efetivos computacionalmente simples, fornecendo desenhos de grafos próximos do ótimo.

Nos chamados *problemas de desenho linear de grafos*, os vértices de um grafo são dispostos em uma linha horizontal no plano, chamada *"espiral"*. Esta linha divide o plano em duas metades chamadas *"páginas"*, correspondendo às duas páginas de um livro aberto. O número de cruzamentos de um grafo em um livro é definido como o número mínimo de cruzamentos de arestas quando os vértices são dispostos na espiral

de um livro de k-páginas e as arestas são desenhadas nas páginas, onde cada aresta está contida em alguma página. Dentre os problemas de desenho linear de grafos, o *problema do número de páginas* (*"pagenumber problem"*, em inglês), busca o número mínimo de páginas necessárias para representar um grafo em desenho linear sem cruzamentos de arestas. Este problema é equivalente ao *problema da espessura do livro* (*"book thickness"*, em inglês). Sabe-se que grafos não-planares necessitam de, no mínimo, três páginas [9].

Um problema de desenho linear de grafos é chamado *fixo* (resp., *não-fixo*) se a ordem dos vértices na espiral é específica (resp., livre). Dentre as aplicações destacamos: desenho de grafos, sistemas de visualização gráfica (onde o número de cruzamentos de arestas é um critério estético utilizado para medir a qualidade do desenho de um grafo) [18], projeto de placas de circuitos impressos e projeto de circuitos VLSI [9].

Cimikowski e Shope [10, 11] estudaram uma versão restrita do problema na qual procuravam otimizar o desenho linear de grafo em 2-páginas com um número mínimo de cruzamentos de arestas. A ordem dos vértices na espiral é fixa e cada aresta é desenhada em uma das páginas como um semicírculo. O problema é conhecido como *Problema do Número de Cruzamentos Linear Fixo* (FLCNP - *"Fixed Linear Crossing Number Problem"*, em inglês) cujo problema de decisão associado é NP-Difícil [52][1].

Dado um grafo G com n vértices e m arestas, o número de configurações possíveis (ou roteamentos[2]) das arestas do grafo G em desenho linear fixo é 2^m. Uma variante do FLCNP, na qual a ordem dos vértices é livre, é conhecida como *Problema do Número de Cruzamentos Linear Não-Fixo* (ULCNP - *"Unfixed Linear Crossing Number Problem"*, em inglês) ou simplesmente *Problema do Número de Cruzamentos Linear*, cujo problema de decisão associado é *NP*-Completo [9][3]. Neste caso, existem $(n-1)!/2$ ordens diferentes de vértices e 2^m roteamentos possíveis das arestas, resultando em $2^{m-1}(n-1)!$ configurações

[1]Masuda et al. [52] utilizaram o problema *Divisão de Conjunto* (*"Set Splitting"*, em inglês) para provar a NP-Dificuldade do Problema do Número de Cruzamentos Linear Fixo.

[2]O *posicionamento* dos vértices (*"placement"*, em inglês) refere-se à posição dos vértices de um grafo em um desenho, enquanto o *roteamento* (*"routing"*, em inglês) refere-se ao caminho percorrido pelos arcos que representam as arestas de um grafo em um desenho.

[3]Chung et al. [9] utilizaram o problema *Ciclo Hamiltoniano* (*"Hamiltonian Circuit"*, em inglês) para provar a NP-Completude do Problema do Número de Cruzamentos Linear.

possíveis, aumentando ainda mais a complexidade do problema.

A ordem dos vértices é fundamental na obtenção do desenho linear ótimo de um grafo [64]. Na Figura 2.15 apresentamos um desenho do $K_{6,3}$ com $cd(D_{K_{6,3}}) = 6$ (observe que $cr(K_{6,3}) = 6$). Outro desenho do $K_{6,3}$ é proposto na Figura 2.16. Este desenho $D'_{K_{6,3}}$, devido à ordem dos vértices, possui $cd(D'_{K_{6,3}}) = 7$, embora apresente o melhor roteamento das arestas em duas páginas.

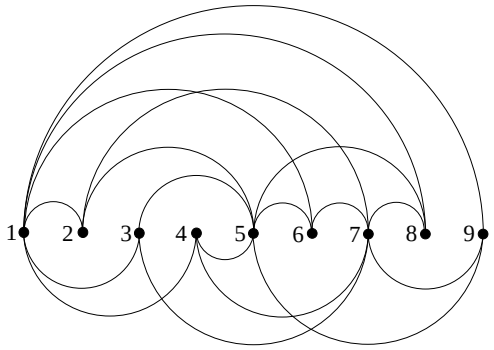


Figura 2.15: Desenho linear do $K_{6,3}$ com $cd(D_{K_{6,3}}) = 6$

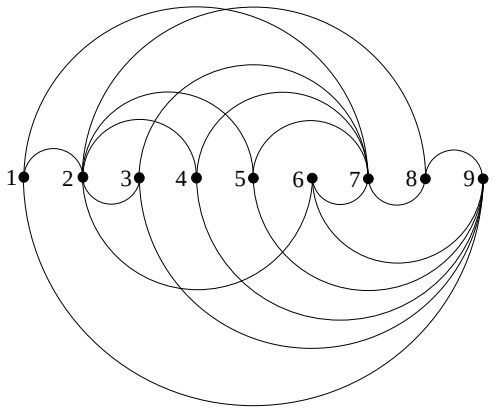


Figura 2.16: Desenho linear do $K_{6,3}$ com $cd(D'_{K_{6,3}}) = 7$

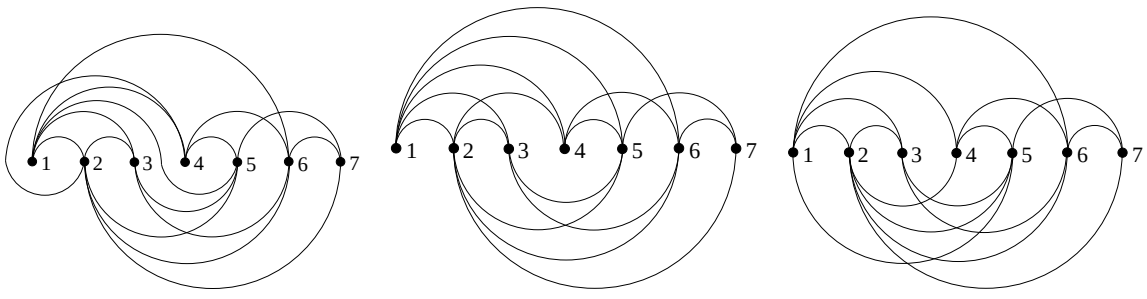
Cimikowski e Shope [11] estudaram a otimização do Problema do Número de Cruzamentos Linear Fixo utilizando uma rede neural. Posteriormente, Cimikowski [10] apresenta uma comparação entre oito heurísticas diferentes, entre as quais, a rede neural se destaca alcançando quase sempre a solução ótima. Uma descrição desta heurística é apresentada no Capítulo 3.

O Problema do Número de Cruzamentos Linear é estudado por Nicholson [55] que propõe uma heurística de complexidade $O(n^3)$. O autor propõe, basicamente, um método

guloso de duas fases. Na primeira fase, os vértices são colocados na espiral. O primeiro vértice posicionado é o vértice de grau máximo. Os demais vértices selecionados serão aqueles com maior quantidade de adjacências com os vértices já posicionados na espiral. Cada vértice é colocado na espiral numa posição onde o aumento no número de cruzamentos é menor e suas arestas são colocadas nas páginas de acordo com o roteamento que oferecer menor quantidade de cruzamentos. Na segunda fase, os vértices são movidos para diferentes posições na espiral, modificando o roteamento das arestas apropriadamente. O vértice escolhido é aquele que possui o maior número de cruzamentos associados às suas arestas. Este, por sua vez, é movido para a posição que oferece a maior redução no número de cruzamentos. Este processo continua até que nenhuma melhoria seja alcançada.

Representação de Arestas

Em um desenho linear de grafo, as arestas podem ser representadas por semicírculos ou por séries de semicírculos. No desenho apresentado na Figura 2.17 (a), todas as arestas são representadas por semicírculos, exceto as arestas $\{1, 5\}$ e $\{2, 4\}$. Assim, o desenho apresenta $cd(D_G) = 4$. Para esta ordem de vértices, se as arestas $\{1, 5\}$ e $\{2, 4\}$ são representadas por semicírculos na página superior ou inferior temos, respectivamente, $cd(D_G) = 5$ (Figura 2.17 (b)) e $cd(D_G) = 8$ (Figura 2.17 (c)). No trabalho de Nicholson [55] assume-se que as boas soluções podem ser alcançadas apenas com semicírculos e, devido à simplicidade, o autor não faz uso de séries de semicírculos na representação de arestas. Cimikowski e Shope [10, 11] também não utilizam esta representação.

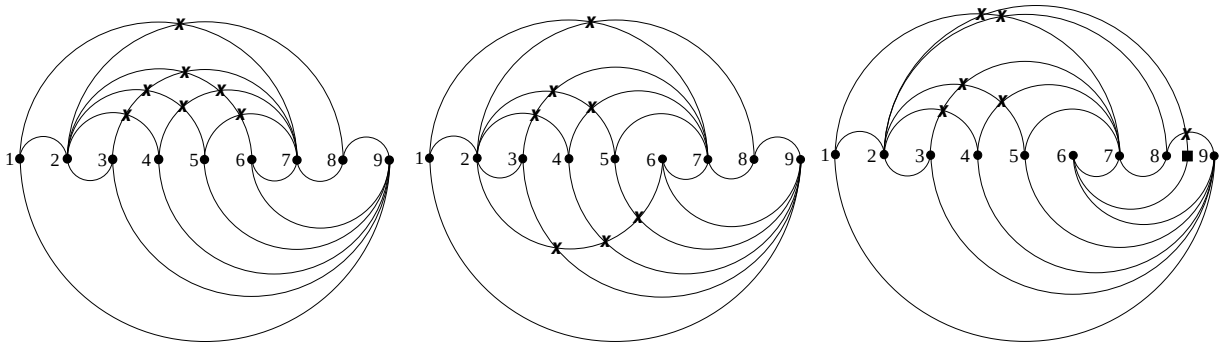


(a) $cd(D_G) = 4$ (b) $cd(D'_G) = 5$ (c) $cd(D''_G) = 8$

Figura 2.17: Desenho linear de grafo com semicírculos e séries de semicírculos

Considerando que o desenho linear ótimo de um grafo pode ser obtido pela otimização tanto da ordem dos vértices como do roteamento das arestas nas páginas, utilizaremos, na otimização do roteamento das arestas, séries de semicírculos através da inserção de vértices falsos na espiral. Limitaremos o conjunto de vértices falsos a um único vértice falso no intervalo central da espiral entre cada par de vértices verdadeiros consecutivos.

Assim, o desenho linear alcançado do $K_{6,3}$ pode ser ótimo sem a alteração da ordem dos vértices, conforme apresentado na Figura 2.18 ($cr(K_{6,3}) = 6$).



(a) $cd(D_{K_{6,3}}) = 7$ (b) $cd(D'_{K_{6,3}}) = 7$ (c) $cd(D''_{K_{6,3}}) = 6$

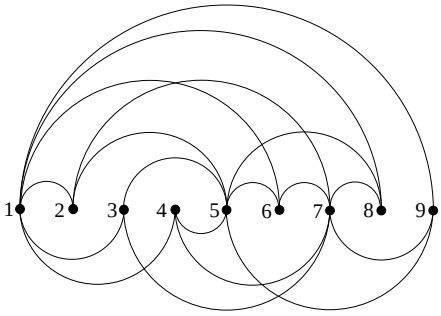
Figura 2.18: Desenho linear do $K_{6,3}$.
x Cruzamentos; ■ Vértices falsos; • Vértices Verdadeiros

Não temos conhecimento de outros trabalhos que utilizem vértices falsos em desenho linear para possibilitar a representação de séries de semicírculos. Assim, esta representação será verificada empiricamente em nosso trabalho.

Representação da Solução

Para representação da solução, Nicholson [55] e Cimikowski e Shope [11, 10] utilizaram duas matrizes de adjacências A e B de dimensões $N \times N$ para um grafo G com N vértices. Para cada aresta $\{i, j\}$, $A[i, j] = 1$ (resp., $B[i, j] = 1$) se a aresta está na página superior (resp., inferior), caso contrário, $A[i, j] = 0$ (resp., $B[i, j] = 0$). A Figura 2.19 (b) apresenta as matrizes de adjacências correspondentes ao grafo $K_{6,3}$ (Figura 2.19 (a)).

$$p_1 < q_1 < p_2 < q_2 \tag{2.3.1}$$



(a) Desenho linear

A (página superior)

	1	2	3	4	5	6	7	8	9
1	0	1				1		1	1
2	1	0			1		1		
3			0		1				
4				0					
5		1	1		0	1		1	
6	1				1	0	1		
7		1				1	0	1	
8	1				1		1	0	
9	1								0

B (página inferior)

	1	2	3	4	5	6	7	8	9
1	0		1	1					
2		0							
3	1		0				1		
4	1			0	1		1		
5				1	0				1
6						0			
7			1	1			0		1
8								0	
9					1		1		0

(b) Matrizes de adjacências

Figura 2.19: Representação de solução em duas matrizes de adjacências para desenho linear do $K_{6,3}$

$$q_1 < p_1 < q_2 < p_2 \tag{2.3.2}$$

No desenho linear de grafos, um cruzamento entre as arestas $e_1 = (p_1, p_2)$ e $e_2 = (q_1, q_2)$ ocorre somente quando e_1 e e_2 estão na mesma página (mesma matriz) e se satisfizerem a equação 2.3.1 ou 2.3.2 (veja Figura 2.20).

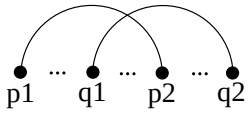


Figura 2.20: Condição para cruzamento de arestas em desenho linear

Dado um desenho D_G de um grafo G representado nesta estrutura matricial, o número de cruzamentos de arestas em D_G pode ser obtido através da equação 2.3.3.

$$cd(D) = \sum_{i=1}^{n-3} \sum_{j=i+2}^{n-1} \left\{ A[i,j] \sum_{k=i+1}^{j-1} \sum_{l=j+1}^{n} A[k,l] + B[i,j] \sum_{k=i+1}^{j-1} \sum_{l=j+1}^{n} B[k,l] \right\} \tag{2.3.3}$$

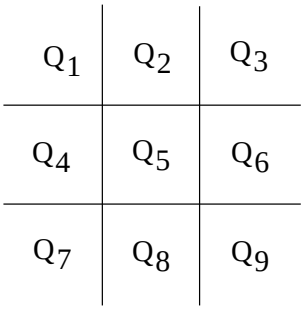


Figura 2.21: Quadrante definido na matriz de adjacências de uma solução

Para cada aresta $\{u, v\}$, a equação 2.3.3 define 9 quadrantes (veja Figura 2.21) na

matriz de adjacências, formado pelas linhas e colunas referentes aos vértices u e v. Considerando que a matriz contém somente as adjacências da página superior ou inferior, os cruzamentos associados à aresta $\{u, v\}$ podem ser encontrados nos quadrantes Q_2 e Q_6 (ou Q_4 e Q_8, uma vez que a matriz é simétrica).

Como exemplo, veja nas Figuras 2.22 e 2.23 o cálculo de cruzamentos para o K_5 em um desenho linear. O grafo K_5 (Figura 2.22 (a)) possui $cr(K_5) = 1$ (Figura 2.22 (b)) e um desenho linear do mesmo é apresentado na Figura 2.22 (c). A matriz de adjacências equivalente ao desenho linear é apresentada na Figura 2.23 (a), onde a matriz A contém as adjacências da página superior e a matriz B as adjacências da página inferior. Na Figura 2.23 (b), destacamos os quadrantes e cruzamentos referentes à aresta $\{1, 3\}$ na matriz A e à aresta $\{1, 4\}$ na matriz B.

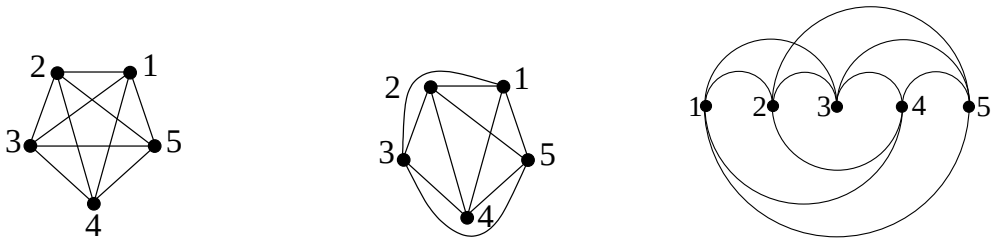


Figura 2.22: Grafo completo K_5

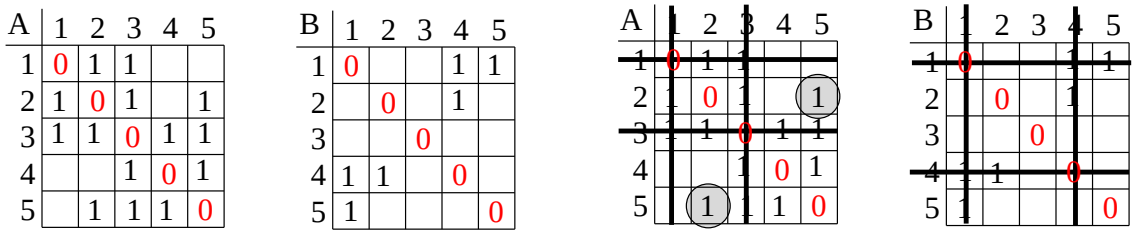


Figura 2.23: Cálculo de cruzamentos na matriz de adjacências para o K_5

Teorema de Nicholson

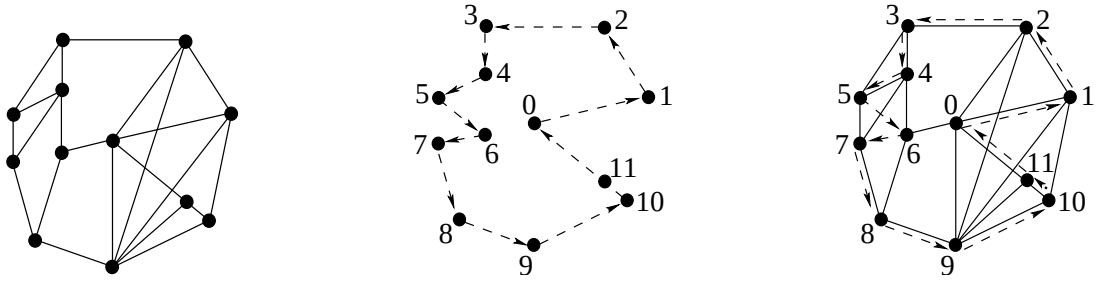
Em seu trabalho [55], Nicholson mostra que qualquer grafo G pode ser representado através de desenho linear com número mínimo de cruzamentos. Para demonstrar este resultado precisamos antes fazer algumas definições.

O *baricentro de um desenho simples* de um grafo G é o baricentro[4] das coordenadas dos vértices de D_G.

Seja D_G um desenho simples de um grafo G no plano π com número mínimo de cruzamentos (supondo que este desenho exista). Seja b o baricentro do grafo G. Para todo vértice $v \in V$ convertemos suas coordenadas cartesianas em coordenadas polares ρ_v e θ_v, onde b é a origem. Agora, numeramos cada vértice v com um valor $l_v \in \{1, 2, 3, \ldots, n\}$, onde $l_u < l_v$ (v e $u \in V$) se uma das seguintes condições é verdadeira:

$$\begin{aligned}
&(i) \quad \rho_u = 0;\\
&(ii) \quad \theta_u < \theta_v;\\
&(iii) \ \theta_u = \theta_v, \rho_u < \rho_v \text{ e } \exists\ w \text{ tal que } \theta_u = \theta_v < \theta_w;\\
&(iv) \ \ \theta_u = \theta_v, \rho_u > \rho_v \text{ e } \nexists\ w \text{ tal que } \theta_u = \theta_v < \theta_w;
\end{aligned} \tag{2.3.4}$$

Um *contorno fechado* C no desenho simples D_G é definido pelos segmentos de retas entre as coordenadas dos vértices u e $v \in V$ onde $l_v = (l_u + 1)\ mod\ n$. A Figura 2.24 ilustra a definição do contorno C para o grafo G da Figura 2.24 (a).



(a) Grafo G (b) Contorno C (c) Grafo G + contorno C

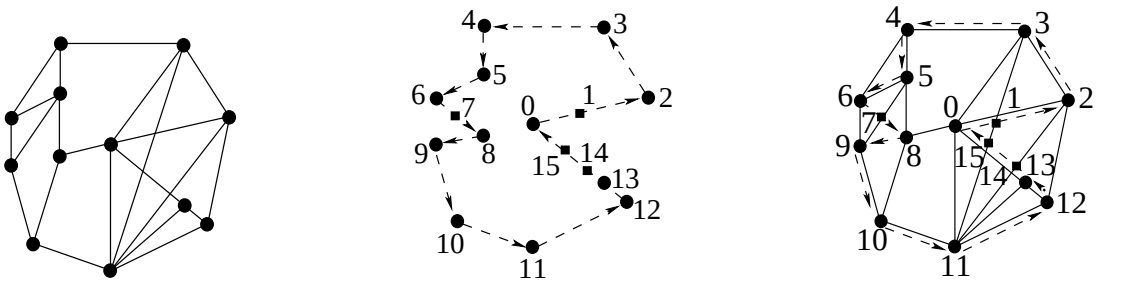
Figura 2.24: Definição do contorno C em um grafo G

Uma vez definido o contorno inicial C, encontramos os pontos em que arestas do desenho D_G interceptam C. Seja k_e o número de vezes que a aresta e intercepta algum segmento de C, exceto é claro quando a aresta e sobrepõe algum segmento do contorno. Seja $p_1, \ldots, p_{k_e}$ as coordenadas dos pontos onde e intercepta C no percurso do vértice u

[4] $\left(\sum_{i=1}^{n} x_i, \sum_{i=1}^{n} y_i\right)$

para o vértice v. Para toda aresta $e = \{u, v\}$ onde $k_e > 0$ subdividimos a aresta e, k_e vezes. Seja $x_1, \ldots, x_{k_e}$ os vértices da subdivisão de e na ordem do percurso do vértice u para o vértice v, atribuímos a x_i as coordenadas do ponto p_i.

Construímos, deste modo, um novo desenho simples provisório D_G^P para a subdivisão G' de G.



(a) Grafo G (b) Contorno C' (c) Grafo G' + contorno C'

Figura 2.25: Definição do contorno C' em um grafo G

Aplicamos novamente as regras de definição do contorno para o grafo G', obtendo praticamente o mesmo contorno, exceto pelo fato de que o novo contorno C' não intercepta nenhuma aresta, exceto as arestas que se sobrepõem ao contorno. A nova função l'_v está definida em V', onde

$$n' = |V'| = |V| + \sum_{e \in E} k_e \ ,$$

seguindo as mesmas regras definidas em 2.3.4, isto é, V' contém todos os vértices de V mais os vértices resultantes das subdivisões das arestas de G (aquelas que interceptaram o contorno C). A Figura 2.25 (a) mostra o grafo G' onde as arestas de G que interceptaram C foram subdivididas. A Figura 2.25 (b) mostra o contorno C' para o grafo G' representado pelo desenho D_G^P e (c) o desenho de grafo $G' + C'$.

Teorema 3 (Nicholson [55]). *Todo desenho simples D_G de um grafo G pode ser convertido em um desenho linear simples D'_G com o mesmo número de cruzamentos de arestas.*

Demonstração: Seja o grafo G' obtido do grafo G através da definição do contorno, conforme definido anteriormente. Obtemos um desenho linear simples D'_G a partir de G', posicionando os vértices do contorno C' na ordem $0, \ldots, n-1$ conforme os valores l'_v

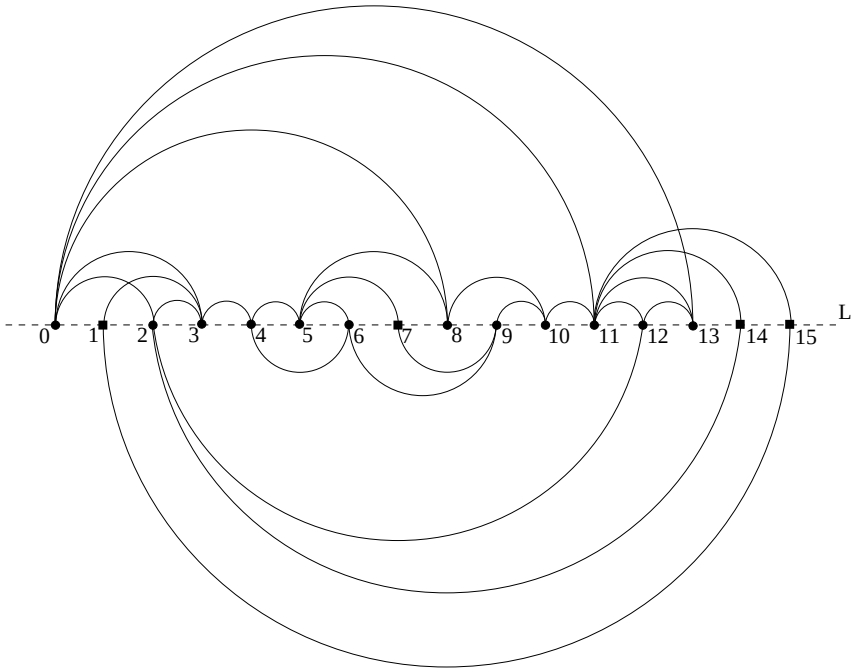


Figura 2.26: Desenho linear do desenho de grafo da Figura 2.24 (a)

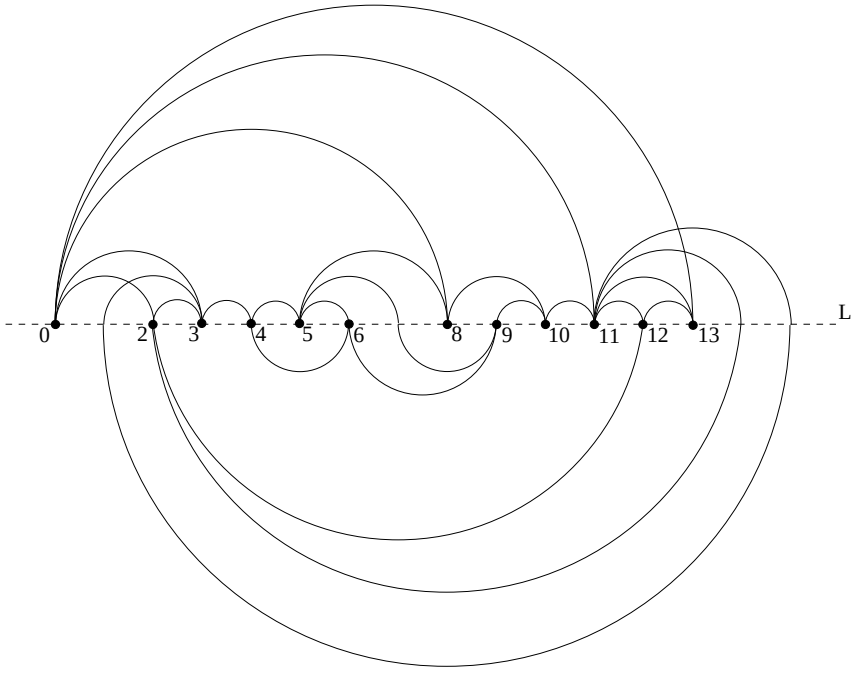


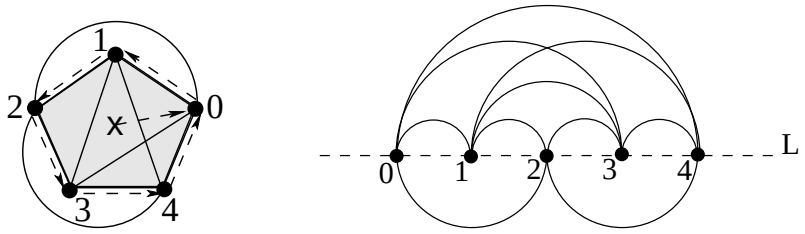
Figura 2.27: Desenho da Figura 2.26 sem os vértices falsos

$\forall v \in V'$ em uma linha horizontal L em D'_G. (veja Figuras 2.26 a 2.30).

Note que as arestas de G' (veja Figura 2.25 (c)) estão inteiramente dentro ou inteiramente fora do contorno C'. Colocamos em D'_G a aresta $e = \{u, v\} \in G'$ como o semicírculo com centro no baricentro das coordenadas de u de v desenhando e na página superior se

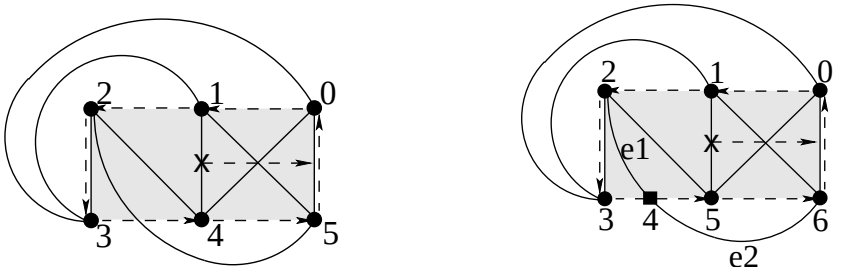
a aresta está inteiramente no interior do contorno (ou se e se sobrepõe a um segmento de C'), caso contrário desenhamos o semicírculo na página inferior.

Duas arestas quaisquer $e_i = \{u_i, v_i\}$ $(l_{u_i} < l_{v_i})$ e $e_j = \{u_j, v_j\}$ $(l_{u_j} < l_{v_j})$ se cruzam em D'_G se e somente se $u_i < u_j < v_i < v_j$ ou $u_j < u_i < v_j < v_i$. □



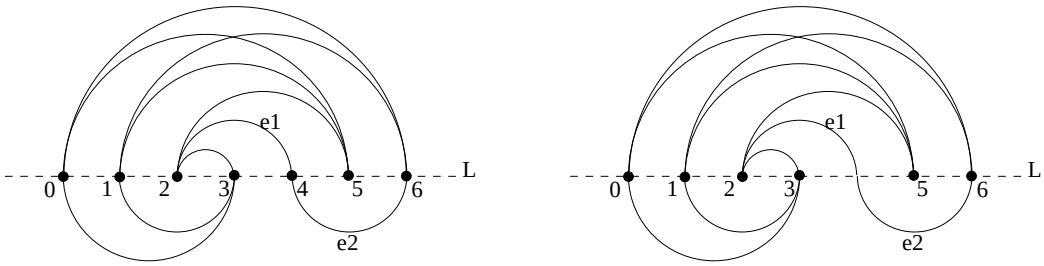
(a) (b)

Figura 2.28: Desenho linear do K_5 com 1 cruzamento



(a) (b)

Figura 2.29: Desenho do $K_{3,3}$ com 1 cruzamento



(a) (b)

Figura 2.30: Desenho linear do $K_{3,3}$ com 1 cruzamento

Assim, para um desenho D_G de um grafo G qualquer, mostramos que é possível obter um desenho linear D'_G do grafo G onde o $cd(D'_G) = cd(D_G)$. A seguir, apresentamos uma técnica de ordenação de vértices muito utilizada em teste de planaridade de grafos chamada *st*-numeração. Como o desenho linear ótimo de um grafo G está relacionado

à ordem dos vértices na espiral, utilizamos esta ordem no processo de otimização do problema.

Em [11], Cimikowski e Shope utilizaram uma heurística gulosa para obter uma ordem linear dos vértices de um grafo G sobre a qual aplicaram a heurística proposta. Em [10], Cimikowski fixou os vértices na espiral segundo a ordem de um ciclo hamiltoniano. Porém, o problema de determinar se um grafo G contém um ciclo hamiltoniano é NP-Completo [32]. Para grafos completos, qualquer ordem dos vértices é um ciclo hamiltoniano[5], porém em testes com grafos randômicos com n vértices, devido à dificuldade computacional do problema, o autor posicionou os vértices na espiral na seqüência $1, 2, \ldots, n$.

Por esta razão, preferimos utilizar a *st*-numeração para obter a ordem dos vértices de um grafo G, visto que esta pode ser encontrada em tempo $O(n + m)$. A *st*-numeração tem sido muito utilizada em planarização de grafos, é encontrada rapidamente e fornece inúmeras ordens dos vértices de um grafo. Além disso, todo ciclo hamiltoniano é uma *st*-numeração (o inverso nem sempre é verdade). Assim, decidimos pela utilização da *st*-numeração em um processo empírico de determinar a ordem dos vértices na espiral.

2.4 *st*-Numeração

A determinação da *st*-numeração de um grafo G é um passo fundamental de pré-processamento em muitas aplicações tais como teste de planaridade, planarização de grafos [6, 8, 19] e desenho de grafos. O primeiro algoritmo reportado na literatura para encontrar uma *st*-numeração para grafos foi proposto por Lempel, Even e Cederbaum [49] como parte de um algoritmo eficiente de teste de planaridade. Neste trabalho, os autores provam que:

Definição 1 (*st*-Numeração). Seja G um grafo biconexo com n vértices e uma aresta qualquer $\{s, t\}$. Os vértices de G podem ser numerados de 1 a n, onde o vértice s recebe o número 1, o vértice t recebe o número n e qualquer vértice v com numeração i (exceto s

[5] O autor não deixa claro o método utilizado para encontrar o ciclo hamiltoniano para outras classes de grafos.

e t) é adjacente a, pelo menos, um vértice u com numeração menor que i e a, pelo menos, um vértice w com numeração maior que i.

A *st*-numeração de um grafo não é única. Considere, por exemplo, o grafo G da Figura 2.31 e a aresta $\{u_1, u_n\}$ como a aresta $\{s, t\}$. Neste caso, qualquer permutação de $2, 3, 4 \ldots, n-1$ atribuída aos vértices $u_2, u_3, \ldots, u_{n-1}$ resulta em uma *st*-numeração, isto é, um total de $(n-2)!$ *st*-numerações possíveis. Dada uma *st*-numeração, a ordem inversa também é uma *st*-numeração de G. Logo, temos $2(n-2)!$ *st*-numerações distintas. Deixamos como exercício mostrar que este grafo possui $6(n-2)!$ *st*-numerações distintas (sugestão: toda aresta do grafo pode ser tomada como aresta $\{s, t\}$).

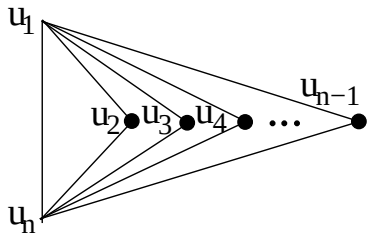


Figura 2.31: Desenho de um grafo G com $6(n-2)!$ *st*-numerações possíveis

O algoritmo de Lempel, Even e Cederbaum [49] tem tempo de execução $O(nm)$ onde n e m são, respectivamente, o número de vértices e o número de arestas do grafo. Esta complexidade foi otimizada por Even e Tarjan [25, 26] modificando o conhecido algoritmo de busca em profundidade (*DFS - Depth First Search*), obtendo um algoritmo de complexidade linear $O(n+m)$ para o mesmo fim. A seguir, apresentamos alguns algoritmos baseados em busca em profundidade para encontrar a *st*-numeração de um grafo.

st-Numeração por Even e Tarjan

Como já afirmamos, o primeiro algoritmo linear para encontrar a *st*-numeração foi desenvolvido por Even e Tarjan [25, 26]. Este algoritmo é dividido em três fases: busca em profundidade, decomposição em caminhos e *st*-numeração dos vértices.

Fase 1: Busca em profundidade

Dado um grafo biconexo G e uma aresta qualquer $\{s, t\}$, a primeira parte do algoritmo (pseudo-código apresentado na Figura 2.33) utiliza busca em profundidade para construir uma árvore de expansão (T, t) de G, onde a primeira aresta de busca da árvore é $\{t, s\}$. É possível mostrar que não existe outra aresta em T saindo de t, exceto a aresta $\{t, s\}$.

Lema 4. *Seja (T, t) uma árvore de expansão do grafo biconexo $G = (V, E)$ obtida pelo algoritmo DFS (busca em profundidade) a partir de $t \in V$ onde a primeira aresta a ser considerada pelo algoritmo é a aresta $\{t, s\}$. Então, toda aresta $\{t, x\}$ onde $x \neq s$ é uma aresta de retorno.*

Porém, antes de demonstrar este Lema, precisamos estabelecer a propriedade fundamental de árvore de expansão obtida pela busca em profundidade.

Lema 5 (Propriedade fundamental da árvore de expansão obtida pelo DFS). *Seja $G = (V, E)$ um grafo conexo, (T, r) uma árvore de expansão de G obtida pela busca em profundidade a partir de r onde $T = (V, F)$. Então, toda aresta $e \in E$ ou é uma aresta de T ($e \in F$) ou é uma aresta de retorno.*

Demonstração: Seja G um grafo conexo, (T, r) uma árvore encontrada pelo algoritmo DFS (apresentado na Figura 2.32) a partir do vértice r. Seja também $e = \{u, v\}$ uma aresta de G. Sem perda de generalidade, suponhamos que o vértice u é visitado pelo algoritmo DFS antes do vértice v. No momento em que o vértice u foi marcado (linha 3 do algoritmo), o algoritmo DFS é executado para todos os vizinhos de u ainda não visitados (linha 6), incluindo v. Se $\{u, v\} \notin T$ então v será visitado através de alguma aresta $\{x, v\}$, antes de considerar a aresta $\{u, v\}$, portanto, a aresta $\{x, v\}$ pertence a T, onde x é descendente de u. Logo, v é descendente de u e a aresta $\{u, v\}$ é uma aresta de retorno. Caso contrário, v será visitado através da aresta $\{u, v\}$ e a aresta $\{u, v\}$ irá pertencer a T. □

Uma vez estabelecida a propriedade fundamental da árvore de expansão obtida pela busca em profundidade, retornamos à demonstração do Lema 4.

```
Require: grafo conexo G e um vértice v qualquer
Ensure: depende da aplicação
    Initialize: pre(v) := 0 ∀v ∈ V e cont := 1;
 1: procedure DFS(G, v);
 2: begin
 3:       pre(v):=cont;
 4:       cont:=cont+1;
 5:       execute pré-processamento;        // depende da aplicação
 6:       for each w ∈ adj(v) do
 7:            if pre(w) = 0 then           // vértice ainda não visitado
 8:                 DFS(G, w);
 9:            execute pós-processamento em {v, w}; // depende da aplicação
10: end;
```

Figura 2.32: Algoritmo de busca em profundidade

Lema 4. *Seja (T, t) uma árvore de expansão do grafo biconexo $G = (V, E)$ obtida pelo algoritmo DFS (busca em profundidade) a partir de $t \in V$ onde a primeira aresta a ser considerada pelo algoritmo é a aresta $\{t, s\}$. Então, toda aresta $\{t, x\}$ onde $x \neq s$ é uma aresta de retorno.*

Demonstração: Seja G um grafo biconexo, (T, t) uma árvore de expansão obtida pelo algoritmo DFS e $\{t, s\}$ a primeira aresta de G a ser considerada pelo algoritmo DFS ($\{t, s\} \in T$). Suponhamos, por absurdo, que exista uma aresta $\{t, x\}$ em T, onde $x \neq s$. Logo pelo Lema 5 não existe um caminho de s para x, exceto o caminho $(s, \{s, t\}, t, \{t, x\}, x)$ passando pelo vértice t. Portanto, o vértice t é um ponto de articulação em G, o que é absurdo, uma vez que G é biconexo. □

Note que no algoritmo da Figura 2.32, linhas 5 e 9, não está definido o significado de pré-processamento e pós-processamento. Assim, a primeira fase do algoritmo de *st*-numeração de Even e Tarjan [25, 26] utiliza estes dois espaços (veja Figura 2.33) para definir os seguintes valores para cada vértice $v \in V$:

- $pre(v)$: é a ordem em que os vértices são visitados (pré-ordem);
- $pai(v)$: é o valor $pre(x)$, onde x é o vértice pai de v em T;

- $low(v)$: é calculado pela seguinte fórmula: Seja L_v o conjunto formado pelos vértices descendentes de v na árvore de expansão T (incluindo v) e P_v o conjunto formado pelos vértices ascendentes de v na árvore T (incluindo v). Pelo Lema 5 toda aresta incidente em um vértice contido em L_v que não pertence a T, é uma aresta de retorno e, portanto, tem seu outro extremo em P_v, então $low(v) = min\{pre(w)\}$ onde $x \in L_v$, $\{x, w\} \notin T$ e $w \in P_v$, isto é, $low(v)$ é o menor dos valores $pre(w)$ com w em P_v onde w possui, pelo menos, um vizinho em L_v. Tarjan [71] mostra que, se G é biconexo e $low(v) = v$, então v é o vértice raiz da árvore de expansão T.

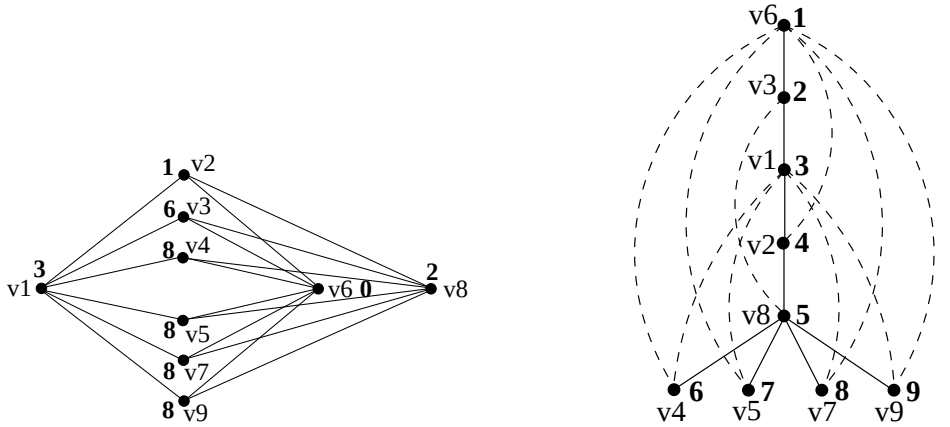
Require: grafo biconexo G e vértice qualquer v
Ensure: cálculo de valores low, pai e pre
Initialize: $pre(v) := 0\ \forall v \in V$ e $cont := 1$;

```
1:  procedure DFS(v);
2:  begin
3:      pre(v):=cont;
4:      cont:=cont+1;
5:      low(v) := pre(v);
6:      for each w ∈ adj(v) do
7:          if pre(w) = 0 then        // vértice ainda não visitado
8:              T := T + v → w;
9:              pai(w) := v;
10:             DFS(w);
11:             low(v) := min{low(v), low(w)};
12:         else if w ≠ pai(v) then
13:             low(v) := min{low(v), pre(w)};
14: end;
```

Figura 2.33: Primeira fase do algoritmo de *st*-numeração de Even e Tarjan [25, 26]

A Figura 2.34 apresenta o grafo $K_{6,3}$ e a árvore de expansão encontrada pelo procedimento DFS para $\{s, t\} = \{6, 3\}$. Em (a), a numeração em negrito corresponde ao vértice pai. Em (b), a numeração em negrito corresponde à propriedade $pre(v)$ pela qual a árvore de expansão foi determinada. Arestas pontilhadas representam as arestas de retorno e as demais representam arestas da árvore de expansão.

Fase 2: Decomposição em caminhos



(a) Grafo com $pai(v)$ em negrito (b) Árvore com $pre(v)$ em negrito

Figura 2.34: Grafo $K_{6,3}$ e árvore de expansão para $\{s,t\}=\{6,3\}$. Para cada vértice v do grafo, $low(v) = 6$.

Na segunda fase, a função $PATHFINDER$ retorna um caminho de arestas não visitadas a partir do vértice v (já visitado) até um vértice distinto w (também já visitado). A função, apresentada na Figura 2.35, é utilizada na última parte do método.

O Corrigendum publicado por Even e Tarjan[26] posteriormente, altera a função $PATHFINDER$ nas linhas 11 e 12. O código original nestas linhas constava como:

procure $(new(w \rightarrow x)$ **or** $new(w \hookrightarrow x))$ **and**
$(x = low(w)$ **or** $low(x) = low(w))$;

Fase 3: *st*-numeração dos vértices

Enfim, na última fase do algoritmo (veja pseudo-código na Figura 2.36), uma pilha S é utilizada para armazenar todos os vértices visitados. A princípio, a pilha contém apenas os vértices s e t, com s no topo da pilha. Então, o vértice v (no topo da pilha) é removido e a função $PATHFINDER$ é chamada. Se a função $PATHFINDER$ retornar um caminho, os vértices que compõem este caminho são adicionados à pilha S de forma que o vértice v esteja novamente no topo da pilha. Desta forma, todos os caminhos existentes a partir de v são encontrados e seus vértices adicionados à pilha S. Quando a função $PATHFINDER$ não retornar mais nenhum caminho a partir do vértice v, um número é associado ao vértice v que não retornará mais à pilha. Esta numeração associada ao vértice v, quando todos os caminhos a partir de v já foram percorridos, é a st-numeração.

```
Require: vértice v e valores low(v), pre(v) e pai(v) para v ∈ G
Ensure: caminho ainda não percorrido com início em v
 1: function PATHFINDER(v);
 2: begin
 3:       seja um vértice w ∈ adj(v);
 4:       if new(v ↪ w) and w →* w then        //aresta de retorno tal que w ∈ s →* v
 5:             marque v ↪ w como old;
 6:             path := {v, w};
 7:       else if new(v → w) then              // aresta de árvore não visitada
 8:             marque v → w como old;
 9:             path := {v, w};
10:             while new(w) do
11:                   procure (new(w ↪ x) and x = low(w)) or
12:                   procure (new(w → x) and low(x) = low(w));
13:                   marque w e w → x como old;
14:                   path := path ∪ (w, x);
15:                   w := x;
16:       else if new(v ↪ w) and v →* w then // aresta de retorno tal que v ∈ s →* w
17:             marque v ↪ w como old;
18:             path := {v, w};
19:             while new(w) do
20:                   procure (new(x → w));
21:                   marque w e x → w como old;
22:                   path := path ∪ (w, x);
23:                   w := x;
24:       else path := ∅;                      // todas adjacências de v estão marcadas
25:       return path;
26: end;
```

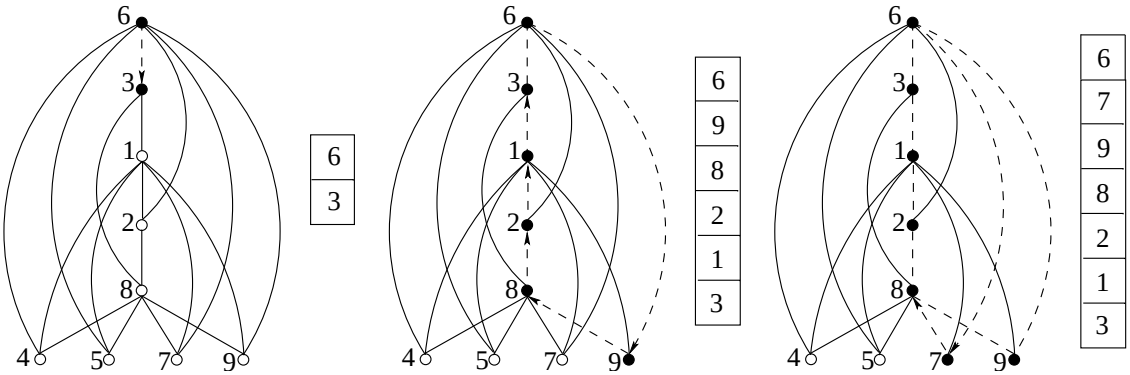
Figura 2.35: Segunda fase do algoritmo de *st*-numeração de Even e Tarjan [25, 26]

Nenhum vértice é permanentemente removido da pilha até que todas as suas arestas incidentes tenham sido visitadas. Assim, um vértice u é removido da pilha depois de s e antes de t (se $s \neq u \neq t$). Além disso, como todos os vértices são eventualmente colocados na pilha, todos recebem um número e, uma vez que s é o primeiro vértice permanentemente removido da pilha e t é o último, s recebe o número 1 e t recebe o número n. As Figuras 2.37 a 2.42 ilustram a ação do algoritmo aplicado ao $K_{6,3}$ da Figura 2.34.

Posteriormente, Ebert [22] também propôs um algoritmo para encontrar a *st*-numeração de um grafo. Baseado neste algoritmo, Tarjan [72] propôs uma simplificação a qual

Require: grafo biconexo G e vértices s e t
Ensure: grafo st-numerado
Initialize: marque s, t e $\{s,t\}$ como *old* e demais vértices e arestas como *new*;
1: **procedure STNUMBER(G,s,t);**
2: **begin**
3: P:=empilha(t);
4: P:=empilha(s);
5: $i := 0$;
6: **while** $P \neq \varnothing$ **do**
7: v:=desempilha(P);
8: $path$:=**PATHFINDER**(v)=$\{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{k-1}, v_k\}\}$;
9: **if** $path \neq \varnothing$ **then**
10: **for** j=k-1 **downto** 1 **do**
11: P:=empilha($path(v_j)$);
12: **else**
13: $i := i + 1$;
14: $stnumber(v) := i$;
15: **end**;

Figura 2.36: Terceira fase do algoritmo de st-numeração de Even e Tarjan [25, 26]



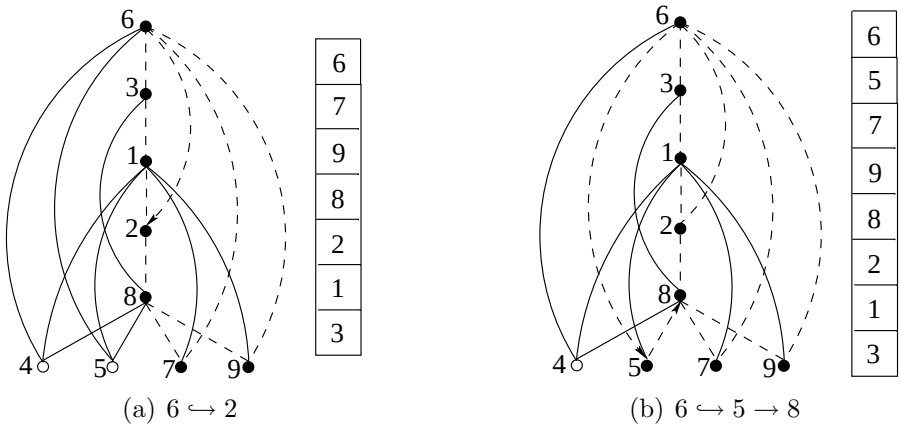
(a) $6 \rightarrow 3$ (b) $6 \hookrightarrow 9 \rightarrow 8 \rightarrow 2 \rightarrow 1 \rightarrow 3$ (c) $6 \hookrightarrow 7 \rightarrow 8$

Figura 2.37: Execução do algoritmo de Even e Tarjan [25, 26]. Arestas percorridas são tracejadas e vértices visitados são preenchidos

apresentamos a seguir.

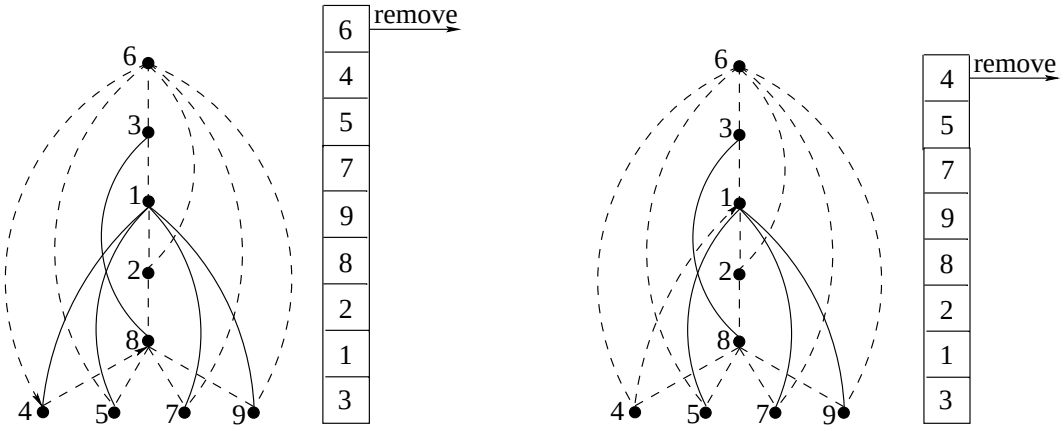
st-Numeração por Tarjan

O método proposto por Tarjan [72] para encontrar a st-numeração também executa em tempo $O(n+m)$, porém não necessita da fase de decomposição de caminhos utilizada no algoritmo de Even e Tarjan [25, 26]. Assim, o algoritmo é dividido em apenas duas fases.



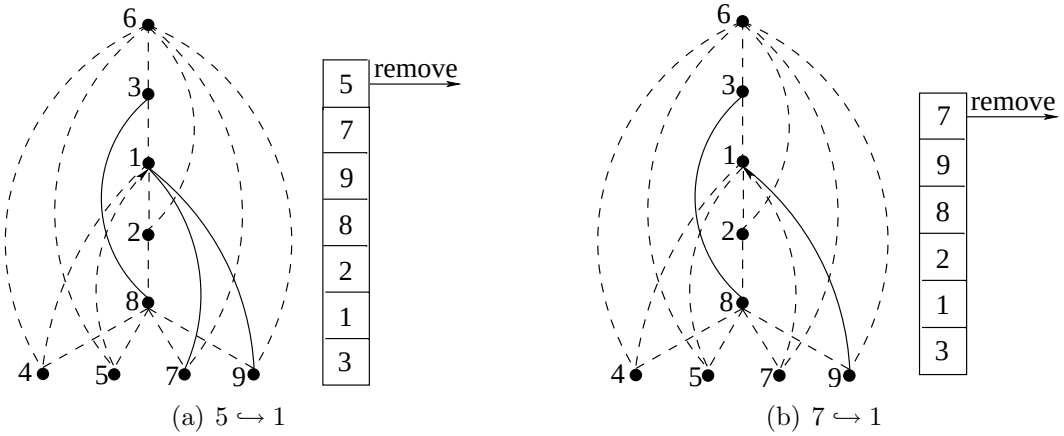
(a) $6 \hookrightarrow 2$ (b) $6 \hookrightarrow 5 \rightarrow 8$

Figura 2.38: Execução do algoritmo de Even e Tarjan [25, 26]



(a) $6 \hookrightarrow 4 \rightarrow 8$ (b) $4 \hookrightarrow 1$

Figura 2.39: Execução do algoritmo de Even e Tarjan [25, 26]

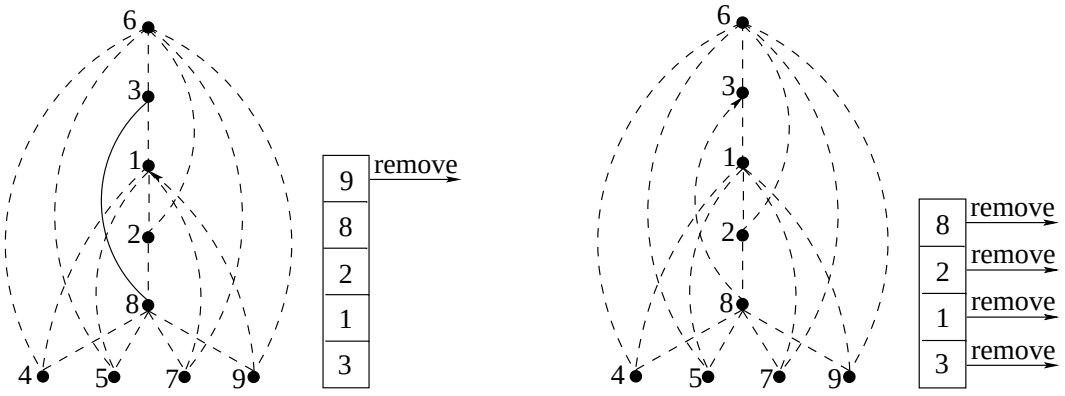


(a) $5 \hookrightarrow 1$ (b) $7 \hookrightarrow 1$

Figura 2.40: Execução do algoritmo de Even e Tarjan [25, 26]

A primeira fase constrói a árvore de expansão T de um grafo G (através da busca em profundidade) calculando os valores $pre(v)$, $low(v)$ e $pai(v)$ para cada $v \in G$.

Na segunda fase do algoritmo, uma lista L com os vértices de G é construída de



(a) $9 \hookrightarrow 1$ (b) $8 \hookrightarrow 3, 2 \rightarrow \varnothing, 1 \rightarrow \varnothing, 3 \rightarrow \varnothing$

Figura 2.41: Execução do algoritmo de Even e Tarjan [25, 26]

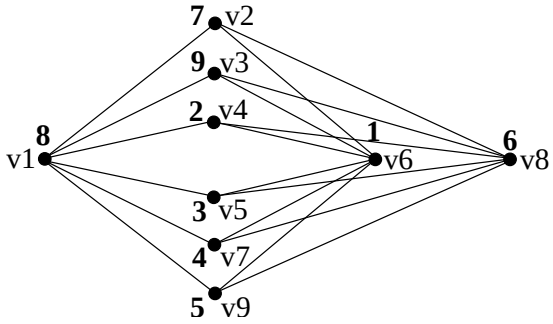


Figura 2.42: Grafo *st*-numerado do $K_{6,3}$ (*st*-numeração em negrito)

Vértice Adicionado	**Lista**
	6-,3
1	6-,1,3+
2	6-,2,1+,3+
8	6-,8,2+,1+,3+
4	6-,4,8+,2+,1+,3+
5	6-,4,5,8+,2+,1+,3+
7	6-,4,5,7,8+,2+,1+,3+
9	6-,4,5,7,9,8+,2+,1+,3+

Tabela 2.3: Lista L para o grafo da Figura 2.34 onde $\{s, t\} = \{6, 3\}$

modo que a ordem dos vértices em L consiste exatamente numa *st*-numeração de G. Esta fase consiste em percorrer a árvore de expansão em pré-ordem e, portanto, as arestas de retorno não são percorridas pelo algoritmo. Durante o percurso, cada vértice u ancestral do vértice v corrente tem um sinal negativo se u precede v em L e um sinal positivo se u sucede v em L. Inicialmente $L = (s, t)$ e s tem sinal negativo. Assim, para cada vértice v no percurso em pré-ordem: caso $sinal(low(v)) = +$, o vértice v será inserido na lista L

após o vértice $pai(v)$. Além disto, modificamos $sinal(pai(v)) = -$. Por outro lado, caso $sinal(low(v)) = -$, o vértice v é inserido na lista L antes do vértice $pai(v)$, modificando $sinal(pai(v)) = +$. A Tabela 2.3 apresenta a lista L para o grafo $K_{6,3}$ (Figura 2.34) onde $\{s, t\} = \{6, 3\}$. A *st*-numeração obtida é a mesma da Figura 2.42.

O pseudo-código do algoritmo de Tarjan [72] é apresentado na Figura 2.43, onde o procedimento recursivo DFS executa a busca em profundidade descrita na primeira fase, calculando os valores $pre(v)$, $low(v)$ e $pai(v)$ e construindo uma lista *preorder* dos vértices (ordenada em ordem crescente de acordo com o valor $pre(v)$). A função $STNUMBER$ retorna uma *st*-numeração para o conjunto de vértices V e a aresta $\{s, t\}$.

Recentemente, Brandes [7] propôs um novo algoritmo cuja vantagem principal sobre o algoritmo de Tarjan [72] é o fato de que o grafo não precisa necessariamente ser biconexo. O algoritmo retorna uma *st*-numeração para a componente biconexa que contém a aresta $\{s, t\}$ dada. Entretanto, este algoritmo não apresenta nenhum ganho na complexidade do algoritmo de Tarjan, este último muito mais conhecido e, por esta razão, o algoritmo de Tarjan [72] foi utilizado neste trabalho.

Mais sobre algoritmos e aplicações de *st*-numeração, recomendamos ao leitor o trabalho de Papamanthou [56].

Na próxima seção, apresentamos a meta-heurística Times Assíncronos utilizada na otimização do Problema do Número de Cruzamentos Linear.

2.5 Times Assíncronos

O conceito de Times Assíncronos ou *A-Teams* (*"Asynchronous Teams"*, em inglês) foi desenvolvido por Talukdar e De Souza [16, 70], sendo aplicado à diversos problemas de Otimização Combinatória de diferentes tipos de restrições e objetivos [20, 60].

Um *Time Assíncrono* é uma coleção de agentes autônomos, conectados através de memórias compartilhadas a fim de formar um fluxo de dados cíclico (veja Figura 2.44).

```
Require: conjunto de vértices V e aresta {s, t}
Ensure: grafo st-numerado
    Initialize: pre(v) := 0 ∀v ∈ V, cont := 1 e L := ∅;
 1: procedure STNUMBER(V,s,t)
 2: begin
 3:         pre(s) := count;
 4:         DFS(t);
 5:         L := s;
 6:         L := L ∪ t;
 7:         plus := true; minus := false;
 8:         sinal(s) := minus;
 9:         for each v ∈ preorder do
10:             if sinal(low(v)) = minus then
11:                 adicione v antes de pai(v) em L;
12:                 sinal(pai(v)) := plus;
13:             else if sinal(low(v)) = plus then
14:                 adicione v depois de pai(v) em L;
15:                 sinal(pai(v)) := minus;
16:         count := 0;
17:         for each v ∈ L do
18:             count := count + 1;
19:             stnumber(v) := count;
20: end;

Require: vértice v ∈ V
Ensure: valores low(v), pre(v) e pai(v)
    Initialize: preorder := ∅;
21: procedure DFS(v)
22: begin
23:         count := count + 1;
24:         pre(v) := count;
25:         low(v) := v;
26:         for each w ∈ adj(v) do
27:             if pre(w) = 0 then
28:                 DFS(w);
29:                 pai(w) := v;
30:                 preorder := preorder ∪ w;
31:                 if pre(low(w)) < pre(low(v)) then
32:                     low(v) := low(w);
33:             else if pre(w) ≠ 0 and pre(w) < pre(low(v)) then
34:                 low(v) := w;
35: end;
```

Figura 2.43: Pseudo-código do algoritmo de *st*-numeração de Tarjan [72]

Os agentes são autônomos, ou seja, não há um agente supervisor que os controla. Cada agente trabalha independentemente dos demais e a comunicação entre eles é feita assincronamente através das memórias. Basicamente, o ciclo de trabalho de um agente consiste em ler soluções da memória de entrada, modificar estas soluções e escrever os resultados na memória de saída, formando assim um fluxo cíclico de dados e permitindo a auto-realimentação do sistema.

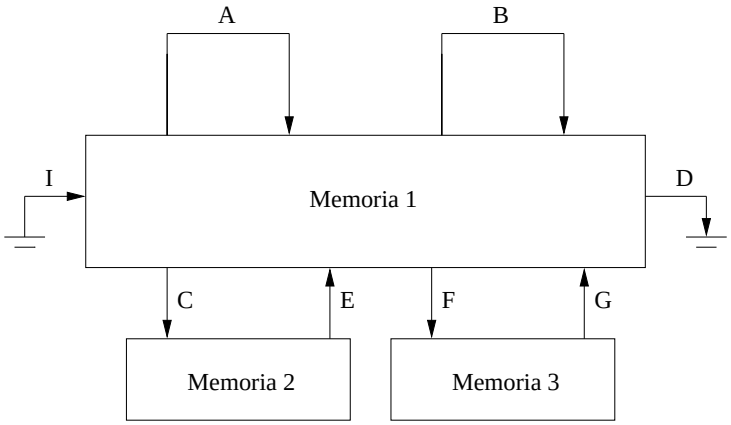


Figura 2.44: Representação gráfica de um Time Assíncrono

A Figura 2.44 apresenta o conceito geral de um Time Assíncrono composto por três memórias (retângulos) e oito agentes (arcos): A, B, C, D, E, F, G, I. O agente I é um agente iniciador, D é o agente destruidor e A, B, C, E, F e G são agentes construtores de soluções.

As memórias de um Time Assíncrono geralmente armazenam soluções parciais ou completas de um problema de otimização. Quando as memórias estão cheias, um agente destruidor (agente D na Figura 2.44) libera espaço removendo soluções "pouco promissoras" conforme uma política de destruição pré-determinada.

Não há restrições na complexidade dos agentes que são adicionados a um Time Assíncrono. Os agentes podem ser simples ou complexos, algoritmos exatos ou heurísticas, etc. Um agente é composto basicamente de:

- uma memória de entrada;
- um escalonador que determina quando o agente trabalha;

- um seletor que escolhe uma ou mais soluções da memória de entrada conforme uma política de seleção pré-determinada;
- um operador que modifica as soluções selecionadas e escreve o resultado na memória de saída;

Um agente é *destruidor* se ele apaga soluções da sua memória de entrada. Um *agente construtor* produz novas soluções pela modificação de soluções existentes na memória.

A eficácia de um agente destrutivo está em seu sistema de controle. Seu operador tem somente que executar a tarefa trivial de apagar objetos. O propósito de um agente destruidor é eliminar os erros ou erros potenciais dos construtores pela remoção das soluções. Uma análise mais profunda sobre os agentes destruidores pode ser encontrada no trabalho de Talukdar et al. [69].

Segundo Talukdar [68], quando agentes são conectados em redes cíclicas, tanto a construção como a destruição são processos duais, ou seja, bons agentes destruidores compensam agentes construtores fracos e vice-versa.

Além disso, uma política de seleção é definida para determinar a seleção de soluções da memória pelos agentes de construção. A experiência com Times Assíncronos [68] sugere que a seleção de Boltzmann (uma adaptação simétrica da estratégia de aceitação da solução usada no *Simulated Annealing*) é suficiente para servir como estratégia de seleção padrão quando nenhuma outra pareça melhor. A seleção randômica usando uma distribuição de Boltzmann pode ser utilizada para selecionar soluções de alta qualidade para os agentes construtores e para selecionar soluções de baixa qualidade para os agentes destruidores.

O poder de um Time Assíncrono encontra-se em seu projeto modular. A abordagem de Times Assíncronos é bastante parecida aos Algoritmos Genéticos em alguns aspectos. Um Algoritmo Genético (GA) trabalha com uma população de soluções, onde perturbações (chamadas "mutações") e combinações (chamados "cruzamentos" ou *"crossovers"*) de

membros existentes são usados para gerar novos membros da população. O tamanho da população e sua qualidade é controlada pela remoção das piores soluções a cada iteração (ou "geração").

Assim, tanto Times Assíncronos quanto Algoritmos Genéticos podem manipular mais de uma solução ao mesmo tempo, criam um fluxo cíclico de dados e combinam tipos diferentes de operadores para produção de bons resultados. A despeito destas semelhanças, um Time Assíncrono não tem o ciclo fixo de execução do Algoritmo Genético, que envolve os passos: de avaliação, de seleção e de reprodução. Em um Time Assíncrono, as atividades são feitas de maneira independente e assíncrona pelos agentes. Além disso, a abordagem de Times Assíncronos define um modelo organizacional para combinar os agentes, que é muito mais geral que os Algoritmos Genéticos, onde um agente pode ser um Algoritmo Genético ou até outro Time Assíncrono. Esta característica permite-nos especificar um sistema bastante modular e flexível.

Outra diferença importante reside na forma estrutural do tempo de vida das soluções. Em um Time Assíncrono uma solução "promissora" pode ter um tempo de vida "quase eterno" enquanto que em um Algoritmo Genético, uma solução tem um ciclo de vida igual a uma única geração. Neste sentido, Algoritmos Genéticos são síncronos. O conceito de geração é praticamente inexistente em Times Assíncronos.

Entre as principais características de Times Assíncronos, destacam-se sua iteratividade, modularidade e possivelmente alto grau de paralelismo. A autonomia e a comunicação assíncrona possibilitam que as execuções dos agentes sejam feitas em paralelo ou cooperativamente, sendo também adequadas ao processamento distribuído, como demonstrado por De Souza [16]. Experiências com paralelismo mostraram um avanço na qualidade das soluções quase proporcional ao número de CPUs utilizadas. Em outras palavras, Times Assíncronos representam comumente um *speed up* próximo do linear em função do número de processadores.

Uma outra característica dos Times Assíncronos é que eles são eficientes em escala.

Segundo Talukdar e De Souza [70] uma organização é eficiente em escala se ela é aberta (cresce facilmente) e se é efetiva no sentido de que seus membros cooperam conseguindo resultados melhores à medida em que novos agentes são acrescentados.

De um modo geral, os Times Assíncronos são adequados para a resolução de uma classe de Problemas Multi-Algorítmicos (De Souza [15]). Essa classe envolve problemas para os quais: (1) não se conhece algoritmos de solução exata com complexidade de tempo polinomial; (2) existem algoritmos heurísticos, aproximativos ou de relaxação capazes de ajustar soluções; e (3) os algoritmos existentes podem ser utilizados como agentes que, por sua vez, podem ser combinados a funcionar de modo iterativo a fim de melhorar continuamente as soluções obtidas.

Em problemas de função multi-objetivo, Algoritmos Genéticos e *Simulated Annealing* otimizam uma função simples, expressa como uma soma ponderada de todos os objetivos e/ou restrições. Assim, os usuários devem decidir a importância relativa dos vários objetivos. Em um Time Assíncrono, os objetivos e restrições individuais de um problema são capturados nos agentes que interagem entre si de maneira autônoma e assíncrona através de memórias compartilhadas contendo soluções candidatas. O problema é particionado por restrições e objetivos e, assim, os agentes são desenvolvidos para melhorar soluções com respeito a uma restrição e/ou objetivo específico (ou a um pequeno número deles). Desta forma, os agentes são chamados agentes especializados pois são especialistas em suas próprias restrições e objetivos, ignorando as demais restrições e/ou objetivos.

Uma discussão mais profunda das características dos Times Assíncronos pode ser encontrada no trabalho de Talukdar [68].

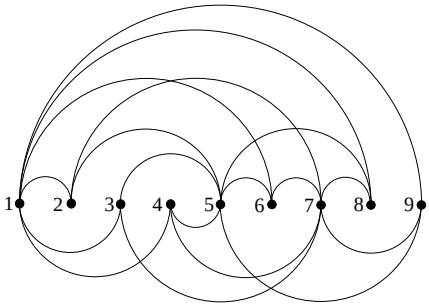
No próximo Capítulo, apresentamos um Time Assíncrono para otimização do problema do número de cruzamentos em desenho linear. Dividimos o problema em dois objetivos principais: otimização da ordem dos vértices na espiral e a otimização do roteamento das arestas nas páginas, obtendo a cooperação entre agentes especialistas em cada um destes objetivos.

3 *Time Assíncrono Proposto*

Neste capítulo, descrevemos um Time Assíncrono utilizado na otimização do Problema do Número de Cruzamentos Linear (ULCNP).

3.1 Representação da Solução

A representação da solução utilizada por Nicholson [55], Cimikowski e Shope [10, 11] consiste do uso de duas matrizes de adjacências A e B as quais correspondem às adjacências contidas nas páginas superior e inferior, respectivamente. Como A e B são simétricas, neste trabalho utilizamos uma única matriz de adjacências, diminuindo assim a estrutura de dados necessária para representação de uma solução. Convencionamos que as adjacências acima e abaixo da diagonal principal da matriz correspondem às arestas da página superior e da página inferior, respectivamente. A Figura 3.1 apresenta um exemplo desta representação.



(a) Grafo $K_{6,3}$

	1	2	3	4	5	6	7	8	9
1	0	1				1		1	1
2		0			1		1		
3	1		0		1				
4	1			0					
5				1	0	1		1	
6						0	1		
7			1	1			0	1	
8								0	
9					1		1		0

(b) Matriz de adjacências

Figura 3.1: Representação de solução em uma matriz de adjacências para desenho linear do $K_{6,3}$

Desta forma, dada uma matriz de adjacências A, o pseudo-código apresentado na

Figura 3.2 encontra o número de cruzamentos de um grafo G nesta representação.

```
Require: matriz de adjacências A e comprimento da matriz N
Ensure: número de cruzamentos de arestas na matriz A
 1: function CROSSGRAPH(A, N);
 2: begin
 3:         sum:=0;
 4:         for i := 1 to N − 3 do
 5:                 for j := i + 2 to N − 1 do
 6:                         for k := i + 1 to j − 1 do
 7:                                 for l := j + 1 to N do
 8:                                         sum := sum + (A[i, j] ∗ A[k, l]);
 9:         for i := N downto 4 do
10:                 for j := i − 2 downto 2 do
11:                         for k := i − 1 downto j + 1 do
12:                                 for l := j − 1 downto 1 do
13:                                         sum := sum + (A[i, j] ∗ A[k, l]);
14:         return sum;
15: end;
```

Figura 3.2: Pseudo-código para cálculo do número de cruzamentos de arestas

3.2 Agentes Utilizados

O Time Assíncrono desenvolvido para o ULCNP possui oito agentes e uma única memória, conforme apresentado na Figura 3.3.

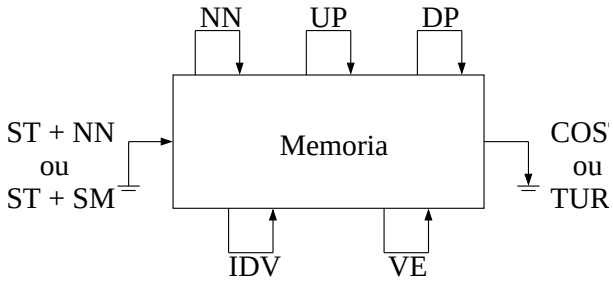


Figura 3.3: Time Assíncrono para o ULCNP

Dois agentes construtores são utilizados para preencher inicialmente a memória: *Start Memory* (SM) e *Neural Network* (NN). A entrada fornecida para estes agentes é uma *st*-numeração do grafo de entrada gerada pelo agente *ST-Numbering* (ST). Como a escolha da aresta $\{s, t\}$ é aleatória, a memória do Time Assíncrono é iniciada com diversas *st*-

numerações (possivelmente diferentes) do grafo de entrada e os agentes SM e NN são responsáveis apenas pelo roteamento das arestas nas páginas.

Considere que neste problema o custo de cada solução é o número de cruzamentos de arestas que buscamos minimizar. Os dois agentes destruidores ($COST$ e $TOUR$) liberam espaço na memória do Time Assíncrono, utilizando como critério de remoção o custo das soluções.

Considerando que os dois aspectos principais do ULCNP são: a otimização da ordem dos vértices na espiral e a otimização do roteamento das arestas nas páginas, cada agente construtor é especializado somente em um destes objetivos. Os agentes desenvolvidos são apresentados a seguir:

1. ***St*-Numbering (ST)**: Gera uma st-numeração do grafo de entrada utilizando o método de Tarjan [72]. A escolha da aresta $\{s, t\}$ é aleatória;

2. **Start Memory (SM)**: agente construtor utilizado para iniciar a memória. Este agente faz uso da st-numeração gerada pelo agente ST para ordenar os vértices na espiral. O agente coloca arestas numa dada página de maneira aleatória;

3. **Neural Network (NN)**: agente construtor utilizado para iniciar a memória e no melhoramento de soluções da memória. Este agente define o roteamento das arestas através da Rede Neural desenvolvida por Cimikowski e Shope [11];

4. **Insert a Dummy Vertex (IDV)**: agente construtor. Insere um vértice falso em uma dada aresta com a finalidade de diminuir o número de cruzamentos pela utilização de uma série de semicírculos;

5. **Vertex Exchange (VE)**: agente construtor. Este agente efetua uma mutação em uma solução obtida da memória invertendo a posição de dois vértices na espiral e modificando apropriadamente suas arestas nas páginas;

6. **Upper Page (UP)**: agente construtor. Este agente é uma modificação do método descrito por Shahrokhi et al. [66] e utilizado por Cimikowski em [10]. O agente

UP é um método guloso que, para cada aresta na página superior, altera a aresta para a página inferior. Se o número de cruzamentos associado à aresta diminuir, a alteração é mantida, caso contrário, a aresta é devolvida à página superior;

7. **Down Page (DP)**: agente construtor. De idéia similar à do agente UP, para cada aresta na página inferior, este agente altera a aresta para a página superior. Se o número de cruzamentos associado à aresta diminuir, a alteração é mantida, caso contrário, a aresta é devolvida à página inferior. Ambos agentes UP e DP utilizam o pseudo-código apresentado na Figura 3.4 para o cálculo de número de cruzamentos associado a uma aresta;

8. **COST**: agente destruidor. Este agente remove uma solução da memória com probabilidade proporcional ao seu custo. Desta forma, soluções com maior número de cruzamentos tem maior probabilidade de serem eliminadas que soluções com menor número de cruzamentos;

9. **Tournament (TOUR)**: agente destruidor. Este agente seleciona duas soluções S_1 e S_2 da memória utilizando a mesma seleção do agente $COST$ (probabilidade proporcional ao custo). Então, compara o custo das soluções S_1 e S_2 e remove a solução de maior custo;

A política de seleção utilizada por todos os agentes construtores é a seleção aleatória. O Time Assíncrono inicia pelo preenchimento da memória através dos agentes ST+SM e ST+NN. Os agentes SM e NN podem executar conjuntamente ou separadamente na tarefa de iniciar a memória. Se ambos forem utilizados conjuntamente, a escolha por um deles é aleatória.

Então, após o preenchimento da memória, agentes construtores geram novas soluções com base em soluções obtidas da memória. À cada solução inserida na memória, um agente destruidor então deve liberar espaço na memória. Os agentes $COST$ e $TOUR$ também podem ser utilizados conjuntamente ou separadamente. Se utilizados conjuntamente, a escolha por um deles é aleatória.

```
Require: matriz de adjacências A e aresta {i, j}
Ensure: número de cruzamentos associado à aresta {i, j}
 1: function CROSSEDGE(A, i, j);
 2: begin
 3:         sum:=0;
 4:         if i < j then                    // aresta na página superior
 5:               for k := i + 1 to j − 1 do
 6:                     for l := 1 to i − 1 do
 7:                           sum := sum + A[l, k]);
 8:                     for l := j + 1 to N do
 9:                           sum := sum + A[k, l]);
10:         else if i > j then               // aresta na página inferior
11:               for k := j + 1 to i − 1 do
12:                     for l := 1 to j − 1 do
13:                           sum := sum + A[k, l]);
14:                     for l := i + 1 to N do
15:                           sum := sum + A[l, k]);
16:         return sum;
17: end;
```

Figura 3.4: Pseudo-código para cálculo do número de cruzamentos de uma aresta

O Time executa até que o critério de parada seja alcançado. Para isto, alguns parâmetros são configurados:

1. Temperatura inicial (T_i) [1];
2. Temperatura final (T_f);
3. Fator de redução da temperatura (T_r);
4. Número de iterações a cada temperatura (N_{it});
5. Tamanho da memória (N);

A partir de T_i o Time Assíncrono executa N_{it} iterações a cada temperatura. A cada iteração um agente é escolhido aleatoriamente. Após N_{it} iterações, a temperatura é decrementada em T_r até alcançar T_f.

[1] A variável temperatura foi inserida originalmente para implementar um agente destruidor baseado na distribuição de Boltzmann. Entretanto, nenhuma contribuição significativa foi observada, uma vez que o agente $TOUR$ demonstrou ser experimentalmente muito mais expressivo.

A quantidade de soluções na memória é definida por N. Através deste parâmetro podemos avaliar a influência que o tamanho da memória tem na qualidade das soluções. Supõe-se que uma memória relativamente pequena não oferece uma diversidade suficiente, portanto, com maior possibilidade de "cair" em mínimos locais.

O pseudo-código do Time Assíncrono é apresentado na Figura 3.5.

```
Require: grafo biconexo G
Ensure: desenho linear de G
 1: ATEAM(G);
 2: begin
 3:         for i := 1 to N do
 4:                 inicia memória através dos agentes ST + SM e/ou ST + NN;
 5:         it := 1;
 6:         while (T > T_f) do
 7:         begin
 8:                 escolhe aleatoriamente um agente construtor;
 9:                 escolhe aleatoriamente uma solução na memória;
10:                 aplica política de destruição através dos agentes COST e/ou TOUR;
11:                 it := it + 1;
12:                 if (it > N_it) then
13:                 begin
14:                         it := 1;
15:                         T := T * T_r;
16:                 end;
17:         end;
18: end;
```

Figura 3.5: Pseudo-código do Time Assíncrono proposto

Com o objetivo de delinear padrões na obtenção de boas soluções, para cada solução o algoritmo armazena um histórico. Este histórico contém informações tais como: nome do agente, custo da solução após a aplicação do agente e tempo de execução do agente sobre a solução. Pela análise destes históricos, acreditamos ser possível definir "receitas" de como gerar boas soluções bem como "receitas" que geram soluções ruins e que devam ser evitadas. Assim, se uma solução ótima foi gerada por uma determinada ordem de execução de determinados agentes, poderíamos criar novos agentes que utilizam diretamente estas "receitas". Esta idéia é utilizada por Do Nascimento et al. [21] na aplicação de um Time Assíncrono para desenho de grafos. Uma discussão sobre esta abordagem está além

do assunto proposto neste trabalho de dissertação e, embora interessante, foi deixado intencionalmente para estudos futuros[2].

3.3 Rede Neural

As *Redes Neurais Artificiais* (*"Artificial Neural Networks"*, em inglês) são sistemas computacionais inspirados no funcionamento do cérebro humano. Existem diversos problemas que os seres humanos resolvem de maneira inata, utilizando o cérebro para: processamento de imagens, reconhecimento de fala, recuperação de informações de maneira associativa, aprendizado de novos fatos e idéias, seleção de informações, entre outras atividades. Entre as áreas de aplicação de Redes Neurais Artificiais encontram-se computação paralela, otimização, controle e reconhecimento de padrões.

Uma Rede Neural Artificial (RNA) consiste em um conjunto de elementos de processamento chamados *neurônios*. Estes elementos são interconectados entre si e são nomeados com inteiros $1, 2, \ldots, N$. As conexões entre os neurônios possuem pesos chamados de *pesos sinápticos*. Cada neurônio i tem M pesos sinápticos, onde cada peso está associado a um dos neurônios de entrada. Além disso, cada neurônio i tem uma entrada Θ_i que representa o limiar do neurônio. A Figura 3.6 apresenta um exemplo de uma Rede Neural.

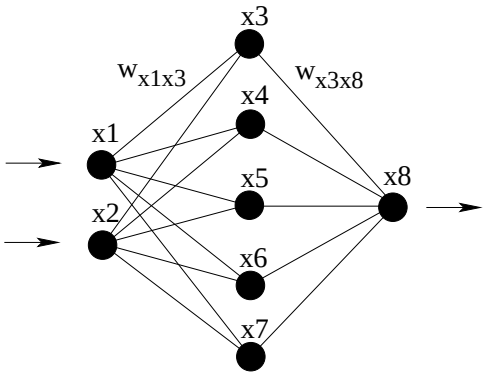


Figura 3.6: Exemplo de uma Rede Neural

As Redes Neurais Artificiais têm origem no trabalho de McCulloch e Pitts [53] que propuseram um modelo matemático simplificado para o neurônio humano. Neste modelo

[2]A implementação do algoritmo proposto já armazena os históricos das soluções, porém o processo de análise destes históricos foi deixado intencionamente para estudos futuros.

cada neurônio tem um estado binário, ou seja, sua saída pode ser 0 ou 1. As entradas têm peso sináptico e a saída do neurônio é calculada pela soma ponderada das entradas com os respectivos pesos como fatores de ponderação. Se o resultado for maior ou igual a um certo limiar, a saída do neurônio é 1, caso contrário, a saída é 0.

O neurônio artificial de McCulloch e Pitts pode ser modelado como um discriminador linear de entradas binárias conforme apresentado na Figura 3.7, onde $w_1, w_2, \ldots, w_p$ são os pesos sinápticos associados às entradas $x_1, x_2, \ldots, x_p$ e podem assumir valores positivos (atuando como força excitatória) ou negativos (atuando como força inibidora). O limiar é representado por Θ, a é o valor da soma ponderada das entradas com os respectivos pesos e $f(a)$ é a função degrau unitário (veja a equação 3.3.1) que define o valor da saída y.

$$y = \begin{cases} 1 \text{ se } a > 0 \\ 0 \text{ se } a \leq 0 \end{cases} \tag{3.3.1}$$

Hopfield [41] mostrou que alguns esquemas de relaxação implementados em Redes Neurais tem uma função de custo e que os estados para os quais a Rede converge são os mínimos locais desta função. Isto significa que a Rede está otimizando uma função bem definida. Porém, não há garantias de que a Rede encontrará o melhor mínimo local.

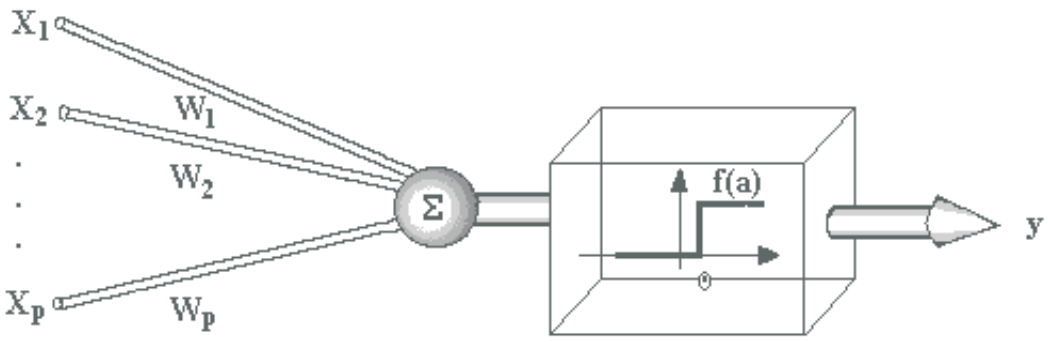


Figura 3.7: Modelo do neurônio de McCulloch-Pitts [53]

Além disso, Hopfield [41] introduziu um modelo de Rede Neural no qual os neurônios são simetricamente conectados, ou seja, para dois neurônios i e j, o peso sináptico w_{ij} é igual a w_{ji}. Cada atualização nos pesos sinápticos reduz (ou, no pior caso, não aumenta)

o valor da função de custo que o autor chamou de "energia" em analogia a sistemas físicos. Repetidas iterações são utilizadas para encontrar a energia mínima do sistema. A energia total do sistema é dada por:

$$E = -\sum_{i<j} w_{ij} x_i x_j + \sum_i \Theta_i x_j \tag{3.3.2}$$

onde w_{ij} é o peso sináptico entre os neurônios i e j, x_i é o estado do neurônio i (0 ou 1) e Θ_i é o limiar do neurônio i.

Uma regra de atualização altera o estado de cada neurônio para o estado que resulta na energia total mais baixa. Como as conexões são simétricas, a diferença entre a energia total do sistema com o estado 0 de um neurônio k e sua energia com o estado 1 de um neurônio k, pode ser determinada pelo neurônio k como:

$$\Delta E_k = \sum_i w_{ki} x_i - \Theta_k \tag{3.3.3}$$

Enquanto Hopfield apresentava este modelo de Rede Neural, Kirkpatrick et al. [44] introduziam um método interessante para resolução de problemas de otimização.

Uma técnica padrão para resolver problemas de otimização é modificar os valores das variáveis na direção em que a função de custo (energia) é reduzida. Esta foi a idéia utilizada por Hopfield em seu modelo de Rede Neural. Pórem, frequentemente, esta técnica obtém os mínimos locais não necessariamente ótimos. Esta é uma conseqüência inevitável ao se permitir somente movimentos "para baixo" (*"downhill"*, em inglês). Se alguns "saltos" para estados de energia mais altos ocorrem ocasionalmente, é possível escapar destes mínimos locais. Entretanto, falta definir como o sistema se comportará com relação a estes movimentos "para cima" (*"uphill"*, em inglês).

Kirkpatrick et al. [44] utilizaram uma analogia física para guiar o uso de movimentos "para cima". No processo físico de resfriamento de metais, para encontrar um estado de menor energia de um metal, a melhor estratégia é aquecê-lo e então reduzir vagarosamente sua temperatura.

Em analogia a este processo, Kirkpatrick et al. [44] propuseram um método de otimização iterativo chamado *Simulated Annealing* (SA) (conhecido também como *Máquinas de Boltzmann* [1]). Neste método, as soluções estão sujeitas a perturbações aleatórias. As perturbações que resultam em soluções melhores são sempre aceitas e as perturbações que resultam em soluções piores são aceitas conforme um critério baseado na temperatura. Se uma perturbação é aceita, a solução corrente é substituída pela nova solução. A proporção de aceitação de soluções ruins diminui conforme a temperatura diminui.

Durante o processo, o sistema obedece a distribuição de Boltzmann:

$$P(E) \simeq e^{\frac{-E}{kT}}$$

que define uma distribuição de probabilidades para estados de energia E em função da temperatura T e da constante de Boltzmann k.

Assim, dada uma solução corrente com custo E e uma nova solução com custo E' onde $E' > E$, a nova solução é aceita se um número aleatório no intervalo $[0, 1]$ é menor que $P(E')$.

Algumas vezes, as Máquinas de Boltzmann são referidas como uma generalização da Rede Neural de Hopfield [41]. Há uma simples modificação na regra de atualização da Rede de Hopfield que a torna uma Máquina de Boltzmann. Esta modificação contorna o problema de convergência da Rede Neural de Hopfield para um mínimo local que não é necessariamente ótimo. A alteração é feita no comportamento dos neurônios que assumem o estado 1 pela probabilidade dada por:

$$P(E_k) = \frac{1}{1 + e^{-\frac{\Delta E_k}{T}}} \tag{3.3.4}$$

onde T é a temperatura corrente e ΔE_k é o valor da atualização de um neurônio k. Então, se um número aleatório no intervalo $[0, 1]$ é menor que $P(E_k)$, o estado do neurônio é 1, caso contrário, é 0.

Desta forma, uma Rede Neural de Hopfield pode escapar de mínimos locais através

de uma regra de atualização não-determinística.

O Agente NN

Hopfield e Tank [42] estão entre os pioneiros a propor uma Rede Neural para problemas de otimização. Atualmente, há uma variedade de Redes Neurais para otimização combinatória baseadas no modelo de Hopfield.

A Rede Neural desenvolvida por Cimikowski e Shope [11] para o FLCNP baseia-se na Rede Neural de Hopfield. A complexidade para implementação paralela é de $O(1)$, entretanto, numa implementação seqüencial é de $O(n^3)$. A ordem dos vértices na espiral foi determinada por uma heurística gulosa.

Para um grafo G com m arestas, a heurística de Cimikowski e Shope utiliza $2m$ neurônios, pois cada aresta é associada a um neurônio *"up"* e a um neurônio *"down"* que representam as páginas superior e inferior, respectivamente. Desta forma, os neurônios $V_{up_{ij}}$ e $V_{down_{ij}}$ representam a aresta $\{i, j\}$ nas páginas superior e inferior, respectivamente. O estado de $V_{up_{ij}}$ e $V_{down_{ij}}$ é binário e está associado à presença ou não da aresta na página relacionada ao neurônio. Assim, para uma aresta $\{i, j\}$, os neurônios $V_{up_{ij}}$ e $V_{down_{ij}}$ podem assumir as seguintes configurações:

- $V_{up_{ij}} = 1$ e $V_{down_{ij}} = 0$: aresta $\{i, j\}$ na página superior;
- $V_{up_{ij}} = 0$ e $V_{down_{ij}} = 1$: aresta $\{i, j\}$ na página inferior;
- $V_{up_{ij}} = 1$ e $V_{down_{ij}} = 1$: duplicidade de arestas pois a aresta $\{i, j\}$ está na página superior e na página inferior;
- $V_{up_{ij}} = 0$ e $V_{down_{ij}} = 0$: adjacência inexistente entre os vértices i e j.

A presença de uma aresta $\{i, j\}$ no grafo G encoraja os neurônios $V_{up_{ij}}$ e $V_{down_{ij}}$ que atuam como força excitatória. Neurônios de arestas com cruzamentos são desencorajados atuando assim como força inibidora.

Require: grafo $G(V, E)$, ordem dos vértices, $n = |V|$ e parâmetros A, B, C, T e Δt
Ensure: desenho linear de G
1: **procedure NEURAL;**
2: **begin**
3: $t := 0$;
4: atribua valores aleatórios no intervalo $[-w, 0]$ (w um número real) a $Up_{ij}(t)$ e $Udown_{ij}(t)$ onde $i = 1, \ldots, n$ e $j = 1, \ldots, n$;
5: **repeat**
6: defina os estados de $Vup_{ij}(t)$ e $Vdown_{ij}(t)$ (equações 3.3.8 e 3.3.9);
7: calcule as alterações de $Up_{ij}(t)$ e $Udown_{ij}(t)$ (equações 3.3.5 e 3.3.6);
8: calcule $Up_{ij}(t+1)$ e $Udown_{ij}(t+1)$ (equação 3.3.7);
9: t:=$t+1$;
10: **until** $t = T$ ou estado estável;
11: **end**;

Figura 3.8: Pseudo-código da Rede Neural de Cimikowski e Shope [11]

A Figura 3.8 apresenta o pseudo-código da Rede Neural de Cimikowski e Shope [11]. A variável t indica a iteração corrente e T é o parâmetro de configuração para o número máximo de iterações (no agente NN, utilizamos $T = 60$). Os pesos Up_{ij} e $Udown_{ij}$ correspondem aos neurônios Vup_{ij} e $Vdown_{ij}$, respectivamente.

Require: valores x e y tal que $V_{up_{ij}} = x$ e $V_{down_{ij}} = y$
1: **function HILL(x, y);**
2: **begin**
3: **if** $x = y = 0$ **then**
4: **return** 1
5: **else**
6: **return** 0;
7: **end**;

Figura 3.9: Pseudo-código da função $hill(x, y)$ utilizada na Rede Neural

A princípio, o desenho não contém nenhuma aresta. Durante a execução do algoritmo, as arestas são adicionadas e movidas entre as páginas superior e inferior conforme os valores das equações 3.3.8 e 3.3.9 a cada iteração. O algoritmo executa até a iteração máxima ou até o estado estável da Rede. A Rede é dita estável se todas as arestas de G

estão presentes no desenho e não existem arestas múltiplas entre quaisquer par de vértices.

$$\begin{aligned} Up_{ij}(t) = &-A(Vup_{ij} + Vdown_{ij} - 1) - B\sum_{l}\sum_{\substack{m \\ l<m}} f(l,i,m)f(i,m,j)Vup_{lm} - \\ &-B\sum_{l}\sum_{\substack{m \\ l<m}} f(i,l,j)f(l,j,m)Vup_{lm} + B\sum_{l}\sum_{\substack{m \\ l<m}} f(l,i,m)f(i,m,j)Vdown_{lm} + \\ &+B\sum_{l}\sum_{\substack{m \\ l<m}} f(i,l,j)f(l,j,m)Vdown_{lm} + C.hill(Vup_{ij}, Vdown_{ij}) \end{aligned} \tag{3.3.5}$$

$$\begin{aligned} Udown_{ij}(t) = &-A(Vup_{ij} + Vdown_{ij} - 1) - B\sum_{l}\sum_{\substack{m \\ l<m}} f(l,i,m)f(i,m,j)Vup_{lm} - \\ &-B\sum_{l}\sum_{\substack{m \\ l<m}} f(i,l,j)f(l,j,m)Vdown_{lm} + B\sum_{l}\sum_{\substack{m \\ l<m}} f(l,i,m)f(l,i,m)Vup_{lm} + \\ &+B\sum_{l}\sum_{\substack{m \\ l<m}} f(i,l,j)f(l,j,m)Vup_{lm} + C.hill(Vup_{ij}, Vdown_{ij}) \end{aligned} \tag{3.3.6}$$

As equações 3.3.8 e 3.3.9 determinam a página de cada aresta, de acordo com os valores calculados nas equações 3.3.5 e 3.3.6. Observe que, na primeira iteração, os pesos são calculados aleatoriamente no intervalo $[-w, 0]$ (utilizamos $w = 1$). A seguir, as equações 3.3.5 e 3.3.6 calculam as alterações nos pesos de cada neurônio. Então, a regra de atualização (veja a equação 3.3.7) calcula o peso de cada neurônio para a próxima iteração utilizando a energia total do sistema ($\Delta Up_{ij}(t)$ e $\Delta Udown_{ij}(t)$).

$$\begin{aligned} Up_{ij}(t+1) &= Up_{ij}(t) + \Delta Up_{ij}(t)\Delta t \\ Udown_{ij}(t+1) &= Udown_{ij}(t) + \Delta Udown_{ij}(t)\Delta t \end{aligned} \tag{3.3.7}$$

As constantes A, B e C, utilizadas nas equações 3.3.5 e 3.3.6, atuam na convergência da Rede Neural e na qualidade da solução final. Segundo Cimikowski e Shope [11], valores baixos para as constantes A e B retardam a convergência da Rede Neural e valores muito altos prejudicam a qualidade da solução final. A constante C encoraja os neurônios de arestas (presentes no grafo original) ausentes na solução. Assim, como estes parâmetros são definidos por tentativa e erro, adotamos os mesmos valores utilizados por Cimikowski

e Shope [11]: $A = 1$, $B = 2$, $C = 2$ e $\Delta t = 10^{-5}$.

$$Vup_{ij}(t) = \begin{cases} 1 \text{ se } Uup_{ij}(t) > 0 \\ 0 \text{ se } Uup_{ij}(t) \leq 0 \end{cases} \tag{3.3.8}$$

$$Vdown_{ij}(t) = \begin{cases} 1 \text{ se } Udown_{ij}(t) > 0 \\ 0 \text{ se } Udown_{ij}(t) \leq 0 \end{cases} \tag{3.3.9}$$

Na função $HILL$ (pseudo-código na Figura 3.9), os valores x e y indicam a presença da aresta $\{i, j\}$ nas páginas superior e inferior, assim $V_{up_{ij}} = x$ e $V_{down_{ij}} = y$. A função $HILL$ age como força excitatória quando uma aresta existente no grafo de entrada está ausente na solução. Além disso, a função permite escapar de mínimos locais e convergir para uma solução estável.

Recentemente, Wang e Okazaki [73] desenvolveram uma Rede Neural (também baseada no modelo de Hopfield) para o FLCNP. Neste trabalho, os autores afirmam que a Rede converge mais rapidamente para a solução ótima que a Rede Neural de Cimikowski e Shope [11]. A Rede Neural de Wang e Okazaki [73] foi aplicada a grafos completos (K_1 a K_{13}) alcançando o resultado ótimo em todos os casos.

3.4 Aspectos dos agentes VE e IDV

No Time Assíncrono proposto, a *st*-numeração é utilizada para gerar diversas ordens de vértices na espiral, trabalhando assim na otimização deste aspecto do problema. Porém, o agente VE não respeita as propriedades da *st*-numeração para inverter a posição de dois vértices v_1 e v_2 na espiral. A razão é ilustrada na Figura 3.10 que apresenta o grafo completo $K_{3,3}$ sem uma aresta. O grafo apresenta um cruzamento de arestas que pode ser resolvido se as posições dos vértices 2 e 3 forem invertidas (b). No entanto, a inversão das posições dos vértices 2 e 3 infringe as propriedades da *st*-numeração pois, ao atribuir a nova numeração aos vértices (em negrito) o vértice 2 deixa de ser adjacente a um vértice de numeração menor.

Uma *st*-numeração possível para o grafo em questão, na qual a ordem dos vértices permite um desenho linear ótimo é apresentado na Figura 3.10 (c). Porém ao inverter as posições de vértices na espiral sem considerar as propriedades da *st*-numeração, possivelmente podemos encontrar uma ordem de vértices que permita um desenho linear ótimo.

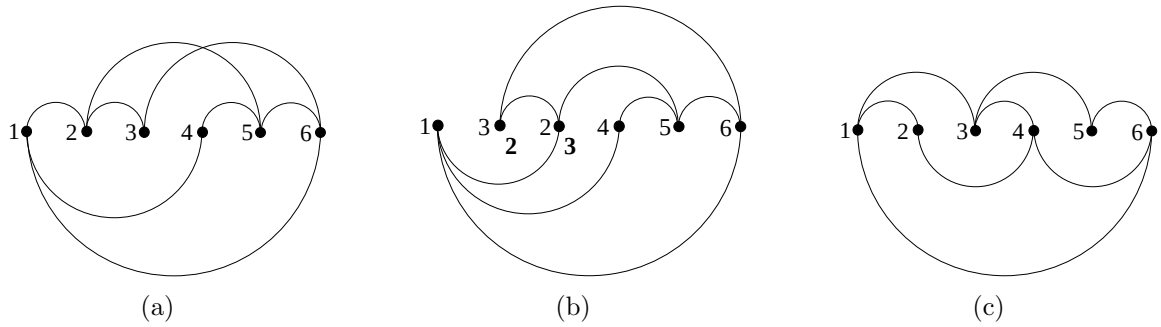


Figura 3.10: *st*-Numeração para desenho linear do $K_{3,3}$ sem uma aresta

No mesmo objetivo, no agente IDV as arestas podem ser representadas simplesmente como semicírculos ou ainda como série de semicírculos. Assim, o objetivo do agente IDV é definir uma aresta que esteja associada a um grande número de cruzamentos e representá-la como uma série de semicírculos.

Na operação de planarização por inserção de vértices falsos, um vértice falso é inserido no cruzamento entre duas arestas. Desta forma, para um grafo G com n vértices, o grafo G' resultante possui $n + cr(G)$ vértices, sendo $cr(G)$ vértices falsos.

Entretanto, a inserção de vértices falsos em um desenho linear pelo agente IDV não altera a estrutura original do grafo pois os vértices falsos são então removidos do grafo resultante, uma vez que utilizamos vértices falsos nesta estrutura apenas para, possivelmente, diminuir o número de cruzamentos da instância.

Para uma aresta e com k cruzamentos, procuramos minimizar k através da divisão da aresta e em e_1 e e_2, de modo que e_1 é desenhada na página superior e e_2 na página inferior ou vice-versa. Assim, a aresta e é representada por uma série de semicírculos formada por e_1 e e_2, conforme ilustra a Figura 3.11 onde $e = \{1, 5\}$.

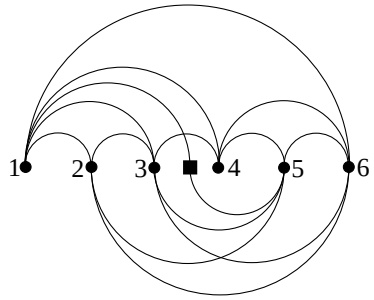


Figura 3.11: Vértices falsos na representação de séries de semicírculos

O agente IDV não insere vértices falsos nos extremos da espiral e a inserção de um vértice falso em uma aresta $e = \{i, j\}$ é feita somente se a distância entre os vértices i e j é maior que 2. Além disso, é necessário pelo menos um vértice verdadeiro qualquer entre o vértice falso inserido e seu vértice verdadeiro adjacente. Na Figura 3.11 a aresta $\{4, 5\}$ tem comprimento 1 e a aresta $\{4, 6\}$ tem comprimento 2. Nestes casos, a inserção de um vértice falso não diminuiria o número de cruzamentos da aresta e, por esta razão, é evitada.

A seguir, apresentamos alguns resultados obtidos pelo Time Assíncrono proposto neste capítulo.

4 Resultados

Neste capítulo, apresentamos alguns resultados obtidos da execução do Time Assíncrono proposto. Este Time Assíncrono dedica-se à otimização do Problema do Número de Cruzamentos Linear procurando minimizar o número de cruzamentos de arestas em desenho linear de grafos. A implementação foi feita em Borland Delphi 7.0 e os testes foram realizados em um PC AMD Athlon XP 1.46 GHz com 256 Mb de RAM.

Na Tabela 4.1 descrevemos as configurações experimentadas. O agente NN (baseado na Rede Neural de Cimikowski e Shope [11]) é utilizado somente na configuração R_2.

Conf.	**ST**	**SM**	**NN**	**IDV**	**VE**	**UP**	**DP**	**COST**	**TOUR**	$\boldsymbol{N}$	$\boldsymbol{N_{it}}$
R_1	X	X		X	X	X	X	X	X	20	20
R_2	X		X					X	X	20	20
R_3	X	X		X	X	X	X	X	X	40	20
R_4	X	X		X	X	X	X	X	X	50	20
R_5	X	X		X	X	X	X	X	X	70	20
R_6	X	X		X	X	X	X	X	X	10	10
R_7	X	X		X	X	X	X	X	X	80	10
R_8	X	X		X	X	X	X	X	X	20	10
R_9	X	X				X	X	X	X	50	20

$T_i = 0.5$, $T_f = 0.1$ e $T_r = 0.9$

Tabela 4.1: Configurações Utilizadas

O algoritmo foi aplicado aos grafos K_n, $K_{n,m}$, Q_n e $C_n \times C_m$. As Tabelas 4.2, 4.3, 4.4 e 4.5 apresentam os resultados obtidos para estas classes de grafos, respectivamente. Nestas tabelas destacamos (em negrito) os resultados que não alcançaram o valor ótimo para facilitar a comparação entre as configurações utilizadas. Os resultados do algoritmo de Nicholson [55] (abreviado para NI) e da Rede Neural implementada por Cimikowski e Shope [11] (abreviado para CS) também são apresentados quando disponíveis.

Para os grafos completos, todas as configurações alcançaram o resultado ótimo (veja a Tabela 4.2), exceto a configuração R_7 (veja o gráfico da Figura 4.1) para os grafos K_{12}, K_{13} e K_{14}. Porém, a configuração R_2 tem tempo de execução muito maior que as demais configurações (veja a Figura 4.2).

Grafo	**Ótimo**	$\boldsymbol{R_1}$	$\boldsymbol{R_2}$	$\boldsymbol{R_3}$	$\boldsymbol{R_4}$	$\boldsymbol{R_5}$	$\boldsymbol{R_6}$	$\boldsymbol{R_7}$	$\boldsymbol{R_8}$	$\boldsymbol{R_9}$	**CS**	**NI**
K_5	1	1	1	1	1	1	1	1	1	1	1	1
K_6	3	3	3	3	3	3	3	3	3	3	3	3
K_7	9	9	9	9	9	9	9	9	9	9	9	9
K_8	18	18	18	18	18	18	18	18	18	18	18	18
K_9	36	36	36	36	36	36	36	36	36	36	36	36
K_{10}	60	60	60	60	60	60	60	60	60	60	60	60
K_{11}	100	100	100	100	100	100	100	100	100	100	100	100
K_{12}	150	150	150	150	150	150	150	**152**	150	150	150	150
K_{13}	225	225	225	225	225	225	225	**231**	225	225	225	225
K_{14}	315*	315	315	315	315	315	315	**327**	315	315	**	315

* Conjectura

** Resultado não disponibilizado pelos autores

Tabela 4.2: Resultados para grafos K_n

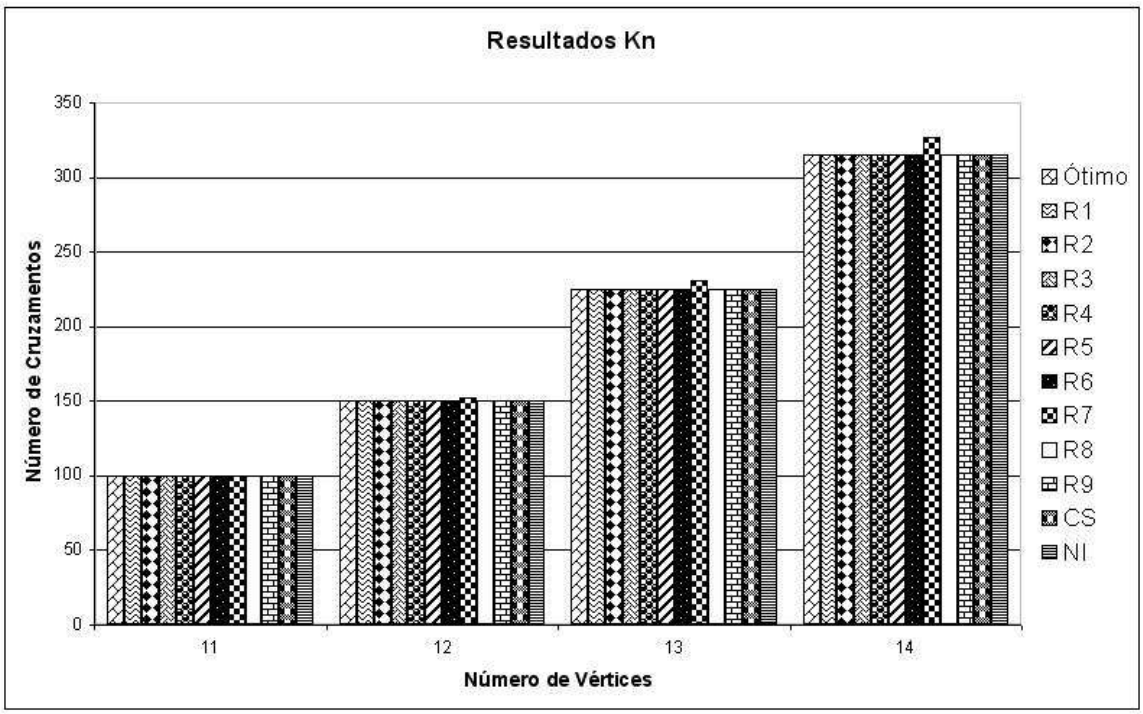


Figura 4.1: Número de cruzamentos para grafos K_n

Na Tabela 4.3 apresentamos os resultados para alguns grafos completos bipartidos. Observamos que novamente a configuração R_7 tem um desempenho inferior em relação às

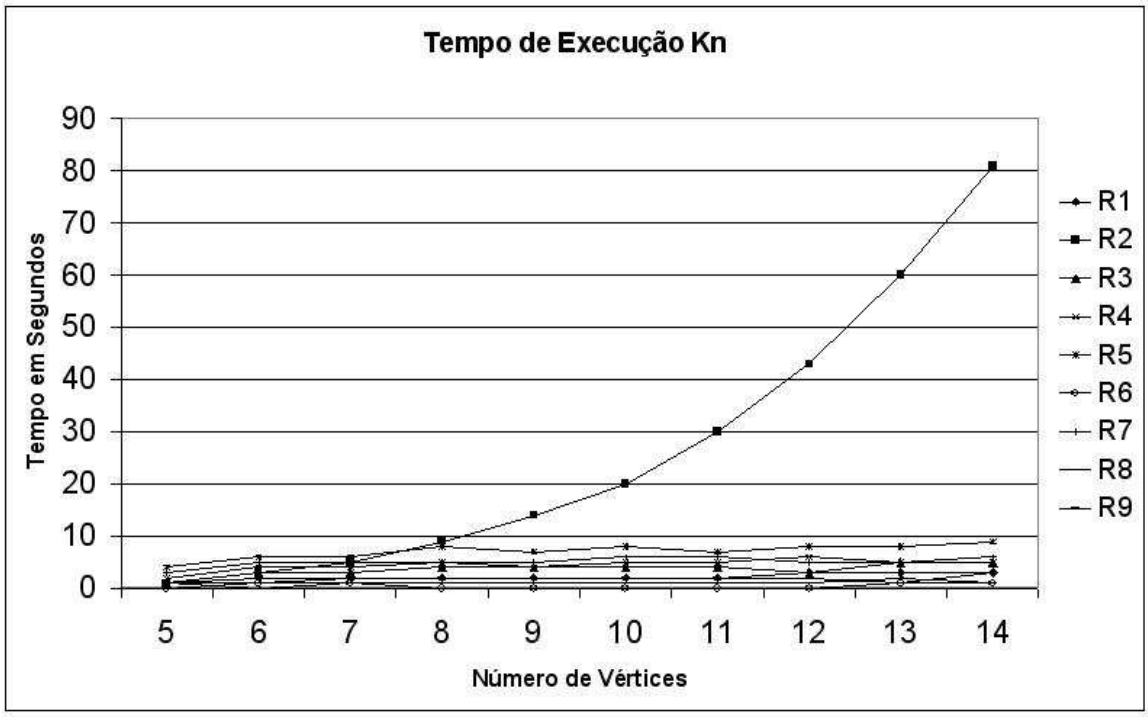


Figura 4.2: Tempo de execução para grafos K_n

demais configurações (veja gráfico da Figura 4.3) até para os grafos $K_{9,9}$ e $K_{11,11}$, onde o resultado ótimo é alcançado em todas as outras configurações. As configurações R_1 e R_6 alcançam os melhores resultados para todos os grafos submetidos. O tempo de execução de cada configuração é apresentado na Figura 4.4.

Grafo	**Ótimo**	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**
$K_{3,3}$	1	1	1	1	1	1	1	1	1	1
$K_{4,4}$	4	4	4	4	4	4	4	4	4	4
$K_{5,5}$	16	16	16	16	16	16	16	16	16	16
$K_{6,6}$	36	36	36	**38**	36	**38**	36	**38**	36	36
$K_{7,7}$	81	81	81	81	81	81	81	81	81	81
$K_{8,8}$	144	144	144	**146**	144	**148**	144	**156**	**146**	144
$K_{9,9}$	256*	256	256	256	256	256	256	**260**	256	256
$K_{10,10}$	400*	400	**402**	400	400	**402**	400	**420**	400	**402**
$K_{11,11}$	625*	625	625	625	625	625	625	**633**	625	625
$K_{12,12}$	900*	**902**	**908**	**904**	**902**	**924**	**902**	**928**	**902**	**902**
$K_{13,14}$	1296*	1296	1296	1296	1296	1296	1296	1296	1296	1296
$K_{14,14}$	1764*	**1766**	**1778**	**1766**	**1770**	**1796**	**1766**	**1858**	**1768**	**1768**

* Conjectura

Tabela 4.3: Resultados para grafos $K_{n,m}$

Na Tabela 4.4 apresentamos os resultados para alguns n-cubos. Observe que apesar

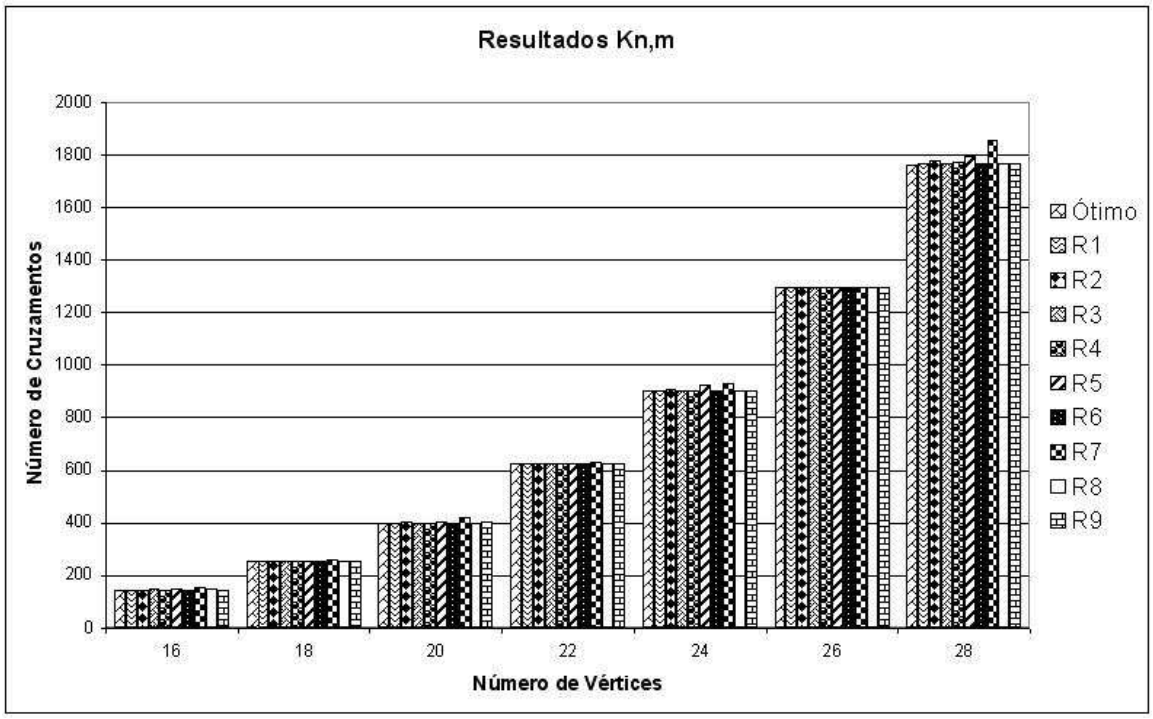


Figura 4.3: Número de cruzamentos para grafos $K_{n,m}$

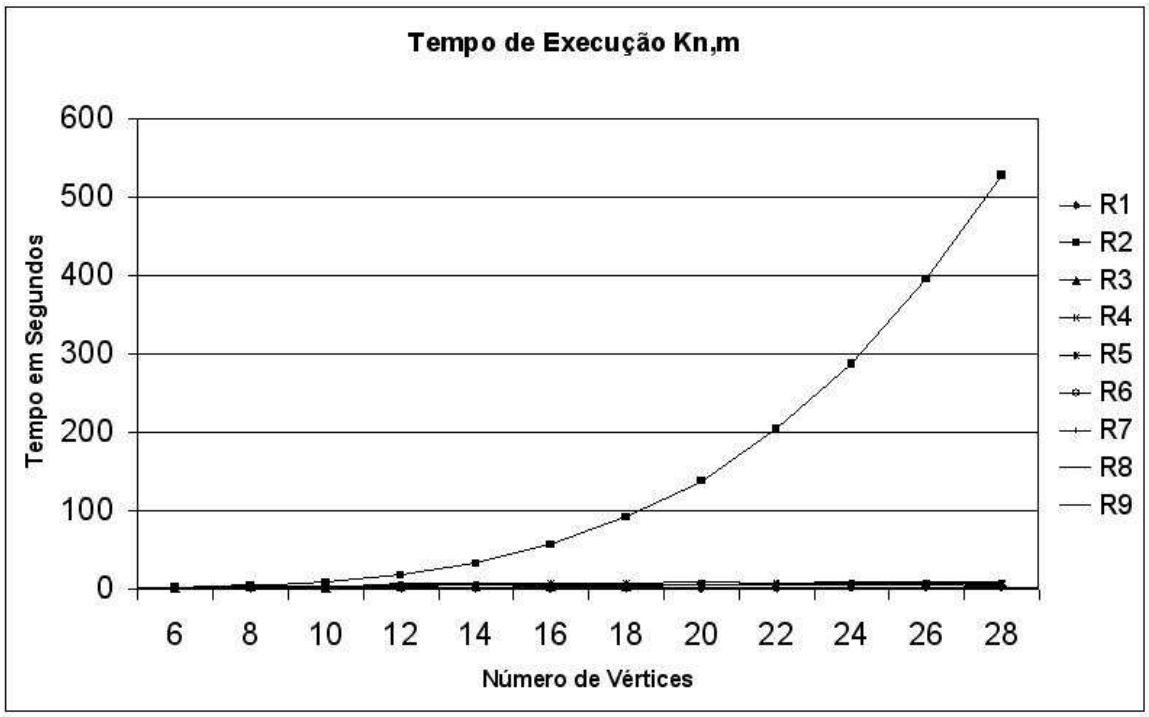


Figura 4.4: Tempo de execução para grafos $K_{n,m}$

do resultado ótimo não ser alcançado pelas configurações experimentadas, exceto para o grafo Q_4, os melhores resultados foram obtidos pela configuração R_9. O tempo de execução de cada configuração é apresentado na Figura 4.6.

Grafo	Ótimo	R1	R2	R3	R4	R5	R6	R7	R8	R9	CS
Q_4	8	8	8	8	8	8	**10**	8	8	8	8
Q_5	56	**92**	**103**	**94**	**90**	**102**	**90**	**113**	**110**	**86**	**62**
Q_6	352*	**672**	**715[a]; 659[b]**	**692**	**690**	**599**	**590**	**753**	**610**	**558**	**376**

* Limite Superior $cr(Q_n) = \frac{5}{32}4^n - \lfloor \frac{n^2+1}{2} \rfloor 2^{n-2}$

[a] Resultado para $N = 3$, $N_{it} = 1$ e $T_i = 0.3$

[b] Resultado para $N = 3$, $N_{it} = 1$ e $T_i = 0.3$, apenas agente SM para iniciar a memória

Tabela 4.4: Resultados para grafos Q_n

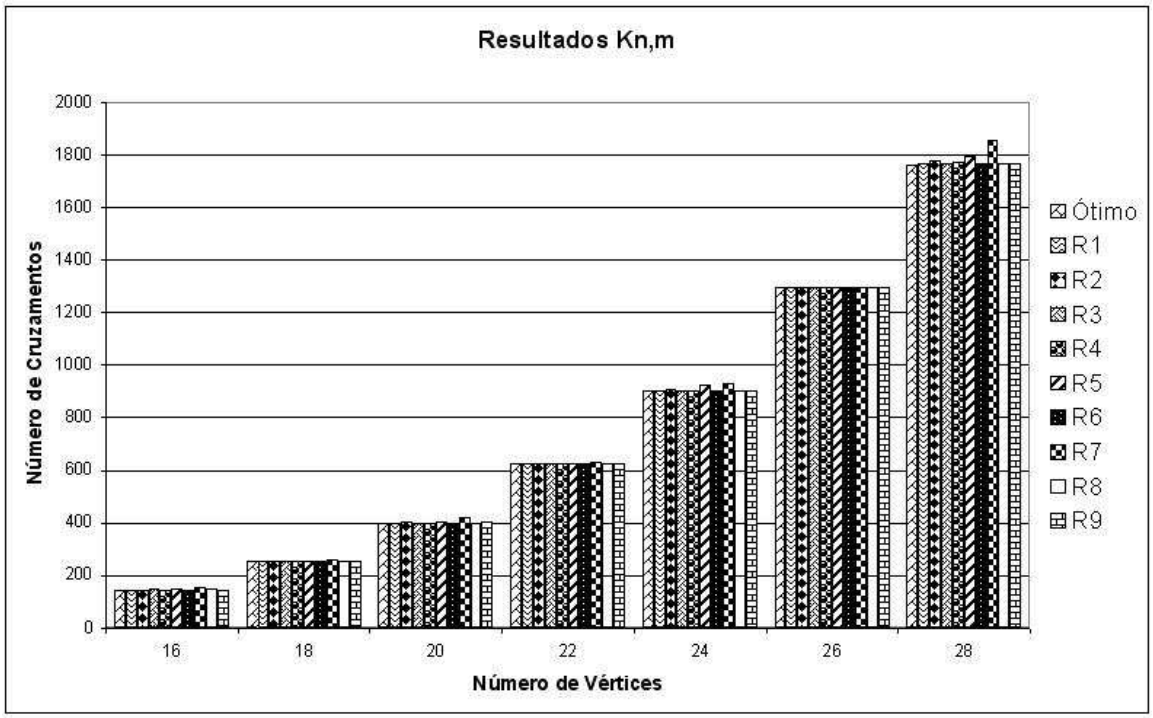


Figura 4.5: Número de cruzamentos para grafos Q_n

Os resultados de grafos $C_n \times C_m$ submetidos ao algoritmo são apresentados na Tabela 4.5. As configurações R_1 e R_9 obtiveram os melhores resultados. A configuração R_2 teve um desempenho inferior em relação às demais pois não alcançou o valor ótimo em quatro casos (as demais configurações não alcançaram o valor ótimo para três casos). Além disso, o tempo de execução da configuração R_2 é muito maior que o tempo das outras configurações (veja Figura 4.8).

Para todos os grafos submetidos ao algoritmo, o tempo de execução da configuração R_2 é sempre muito maior que as demais e nem sempre alcança o melhor resultado. Observamos que a configuração R_7 tem um desempenho visivelmente inferior em relação às demais configurações, em especial para grafos completos e completos bipartidos. Uma possível

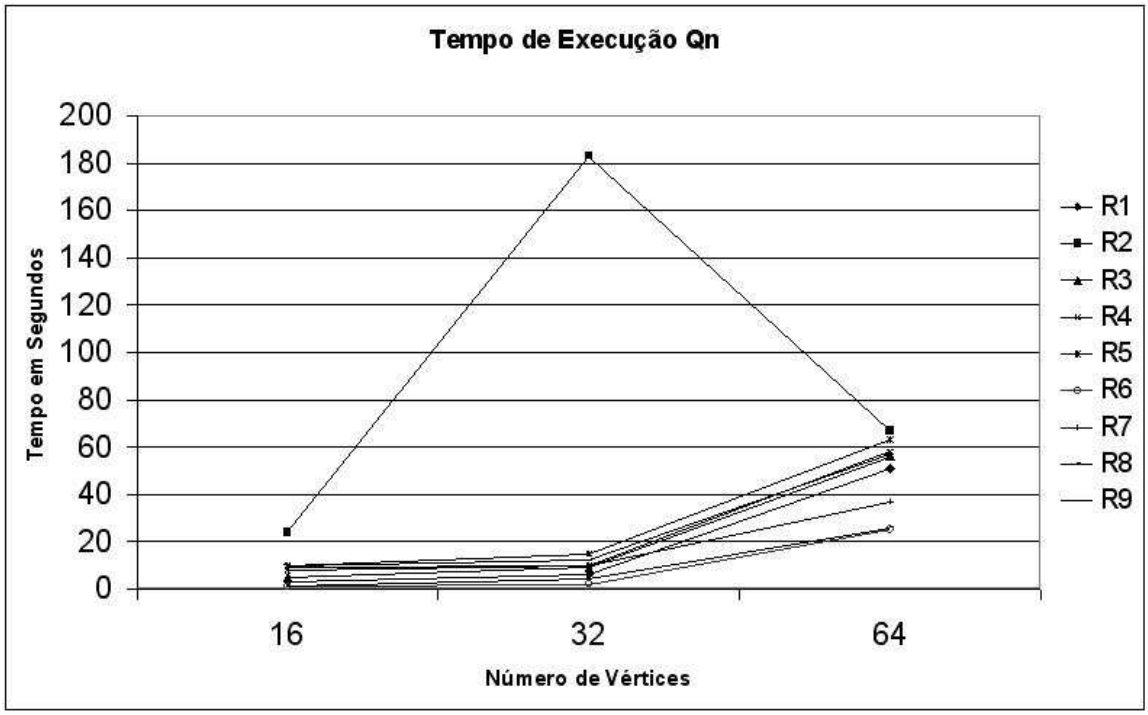


Figura 4.6: Tempo de execução para grafos Q_n

Grafo	**Ótimo**	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**CS**
$C_3 \times C_3$	3	3	3	3	3	3	3	3	3	3	3
$C_4 \times C_4$	8	8	8	8	8	8	8	8	8	8	8
$C_5 \times C_5$	15	**19**	**20**	**18**	**19**	**19**	**18**	**20**	**20**	**18**	**20**
$C_6 \times C_6$	24	24	**32**	24	24	24	24	24	24	24	24
$C_7 \times C_7$	35	**48**	**54**	**55**	**50**	**51**	**52**	**52**	**54**	**52**	**48**
$C_8 \times C_8$	48	48	48	48	48	48	48	48	48	48	48
$C_9 \times C_9$	63^a	**88**	**108**	**112**	**107**	**91**	**112**	**102**	**100**	**102**	**225**

[a] Conjectura

Tabela 4.5: Resultados para grafos $C_n \times C_m$

explicação é que como o tamanho da memória para a configuração R_7 é a maior dentre todas as configurações ($N = 80$), o tamanho da memória tem relação com a qualidade das soluções. Esperamos investigar mais a fundo este fato.

A configuração R_9 teve resultados muito interessantes. Esta configuração tem apenas o agente SM para iniciar a memória e os dois agentes UP e DP para melhorar as soluções na memória. No entanto, alcançou os melhores resultados em todos os grafos experimentados. Nos resultados com grafos Q_n, a configuração R_9 obteve os melhores resultados.

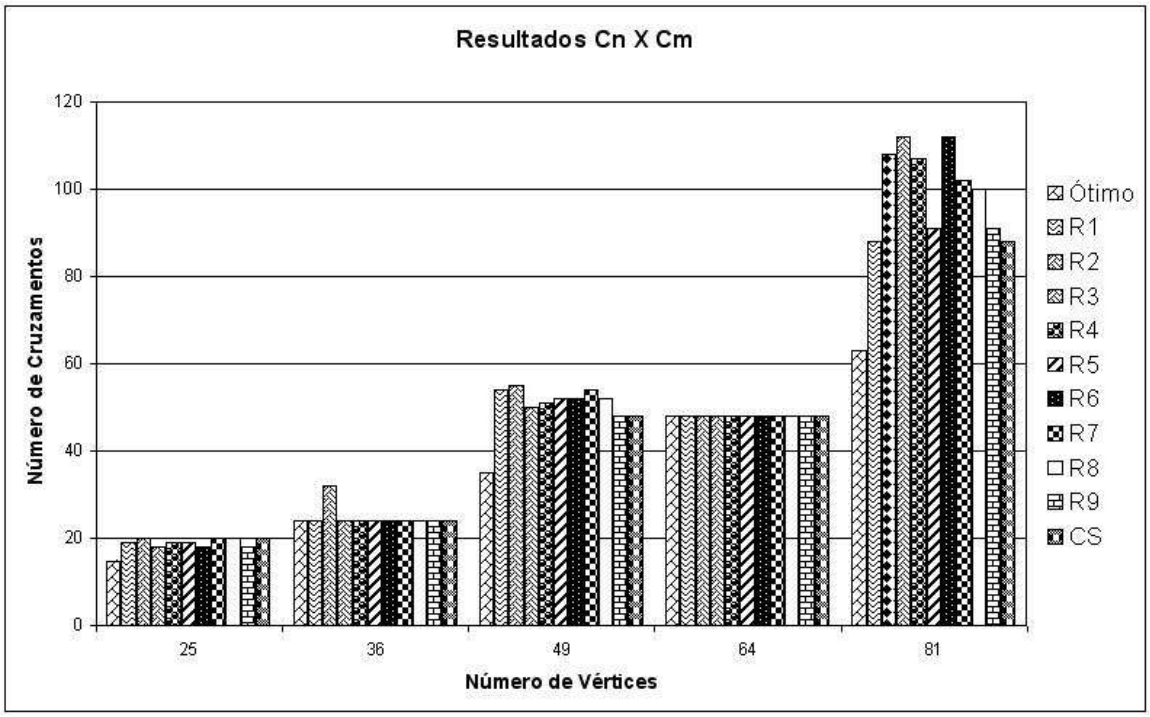


Figura 4.7: Número de cruzamentos para grafos $C_n \times C_m$

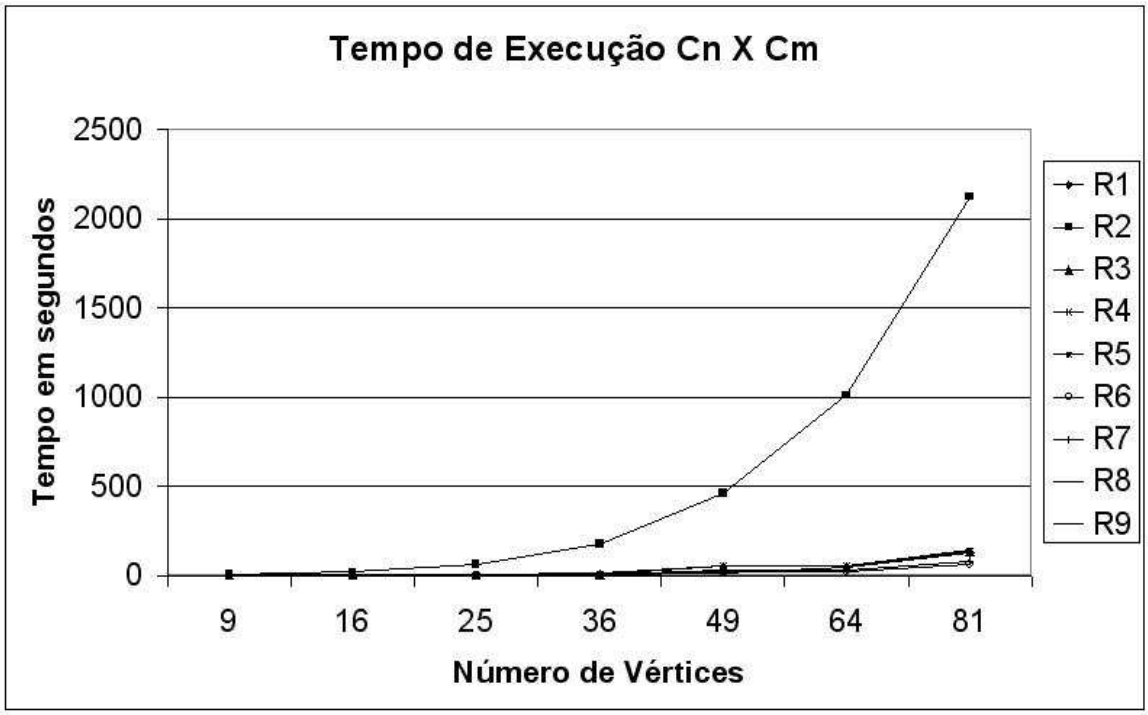


Figura 4.8: Tempo de execução para grafos $C_n \times C_m$

5 Conclusão

Apresentamos um Time Assíncrono para otimização do Problema do Número de Cruzamentos Linear procurando minimizar o número de cruzamentos de arestas em desenho linear de grafos. O Problema requer a otimização da ordem dos vértices na espiral e a otimização do roteamento das arestas nas páginas. Na literatura, com exceção do trabalho de Nicholson [55], não temos conhecimento de outros trabalhos para otimização deste problema.

Utilizamos a *st*-numeração para determinar a posição dos vértices na espiral. Esta ordem pode ser encontrada em tempo $O(n + m)$ qualquer que seja o grafo. Isto é uma vantagem sobre a abordagem de Cimikowski, que em seu trabalho [10], utiliza ciclos hamiltonianos. Note que esta ordem específica é submetida a algoritmos que determinam o roteamento das arestas nas páginas. Porém, sabe-se que nem todo ciclo hamiltoniano é uma ordem ótima dos vértices na espiral [9]. Além disso, o problema de determinar se um grafo G contém um ciclo hamiltoniano é NP-Completo [32]. Embora seja trivial para o grafo completo e completo bipartido.

Utilizamos também séries de semicírculos para representação das arestas no desenho linear de grafos. Esta implementação utiliza o conceito de vértices falsos.

Além disso, nosso algoritmo não somente determina o número de cruzamentos de um grafo, como também fornece um desenho do grafo com o número de cruzamentos obtido.

Baseado na meta-heurística Times Assíncronos, o algoritmo permite a sinergia de diferentes agentes que podem otimizar características particulares do problema. A escala-

bilidade dos Times Assíncronos permite uma fácil inserção/remoção de agentes, proporcionando maior flexibilidade ao método, de forma que novas alterações podem ser facilmente realizadas de acordo com características ou necessidades particulares.

Finalmente, outra contribuição importante deste trabalho consiste na apresentação do Teorema 3 de Nicholson [55].

Alguns aspectos do Time Assíncrono proposto podem ainda ser investigados, como a execução de outras configurações através da adição/remoção de agentes e/ou da configuração de diversos valores nos parâmetros de entrada, bem como o tamanho da memória. Desejamos também investigar a utilização dos históricos das soluções que permite a obtenção de "receitas" de como gerar as soluções ótimas (e também como evitar as soluções ruins) na possível diminuição do tempo de execução do algoritmo.

Além disso, devido ao alto grau de paralelismo dos Times Assíncronos, a implementação paralela do algoritmo provavelmente resultará em melhores soluções dentro de um tempo mais aceitável.

ANEXO A – Definições

A

Adjacente 17

Agente destruidor 52

Agente de construção 52

Ancestral 22

Aresta 17

Arestas paralelas ou múltiplas 17

Arestas paralelas ou múltiplas 17

Árvore 21

Aresta de árvore 22

Árvore de expansão 22

Aresta de retorno 22

Árvore enraizada 22

B

Baricentro de um desenho simples 35

C

Caminho 19

Caminhos disjuntos 19

Caminho hamiltoniano 19

Ciclo 19

Ciclo hamiltoniano 19

Componente biconexa ou bloco 21

Referências

[1] Aarts, E., and Korst, J. *Simulated Annealing and Boltzmann Machines.* John Wiley & Sons Ltd., 1989.

[2] Adamsson, J., and Richter, R. B. Arrangements, circular arrangements and the crossing number of $C_7 \times C_n$. *Journal of Combinatorial Theory Series B 90* (2004), 21–39.

[3] Anderson, M. S., Richter, R. B., and Rodney, P. The crossing number of $C_6 \times C_6$. *Congressus Numerantium 118* (1996), 97–107.

[4] Anderson, M. S., Richter, R. B., and Rodney, P. The crossing number of $C_7 \times C_7$. In *Proc.* 28^{th} *Southeastern Conference on Combinatorics, Graph Theory and Computing* (Florida, USA, 1997), Boca Raton.

[5] Beineke, L. W., and Ringeisen, R. D. On the crossing numbers of products of cycles and graphs of order four. *Journal of Graph Theory 4* (1980), 145–155.

[6] Biedl, T., and Kant, G. A better heuristic for orthogonal graph drawings. *Computational Geometry 9* (1998), 159–180.

[7] Brandes, U. Eager *st*-ordering. In *ESA'02: Proceedings of the 10th Annual European Symposium on Algorithms* (London, UK, 2002), vol. 2461 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 247–256.

[8] Calamoneri, T., and Petreschi, R. An efficient orthogonal grid drawing algorithm for cubic graphs. In *Proc. First Annual International Conference on Computing and Combinatorics (COCOON'95)* (1995), vol. 959 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 31–40.

[9] Chung, F. R. K., Leighton, F. T., and Rosenberg, A. L. Embedding graphs in books: a layout problem with applications to VLSI design. *SIAM Journal of Algebra Discrete Methods 8* (1987), 33–58.

[10] Cimikowski, R. Algorithms for the fixed linear crossing number problem. *Discrete Applied Mathematics 122* (2002), 93–115.

[11] Cimikowski, R., and Shope, P. A neural network algorithm for a graph layout problem. *IEEE Transactions on Neural Networks 7*, 2 (1996), 341–349.

[12] Călinescu, G., Fernandes, C. G., Finkler, U., and Karloff, J. A better approximation algorithm for finding planar subgraphs. *Journal of Algorithms 27* (1998), 269–302.

[13] de Figueiredo, C. M. H., Faria, L., and de Mendonça, C. F. X. On the complexity of the approximation of nonplanarity parameters for cubic graphs. In *Proceedings of the 2001 Brazilian Symposium on Graphs, Algorithms and Combinatorics (GRACO'2001)* (2001), Electronic Notes in Discrete Mathematics. Full paper in Discrete Applied Mathematics 141 (1-3), 119-134, 2004.

[14] de Klerk, E., Maharry, J., Pasechnik, D. V., Richter, R. B., and Salazar, G. Improved bounds for the crossing numbers of $K_{m,n}$ and K_n. In *MIGHTY (Midwest Graph Theory) Conferences* (2005).

[15] de Souza, P. S. *Asynchronous Organizations for Multi-algorithm Problems.* PhD thesis, University of Carnegie Mellon University, Department of Electrical and Computer Engineering, 1993.

[16] de Souza, P. S., and Talukdar, S. N. Asynchronous organizations for multi-algorithm problems. *ACM Symposium on Applied Computing* (1993).

[17] Dean, A. M., and Richter, R. B. The crossing number of $C_4 \times C_4$. *Journal of Graph Theory 19* (1995), 125–129.

[18] di Batista, G., Eades, P., Tamassia, T., and Tollis, I. G. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry 4* (1994), 235–282.

[19] di Batista, G., Garg, A., Liotta, G., Tamassia, R., Tassinari, E., and Vargiu, F. An experimental comparison of four graph drawing algorithms. *Computational Geometry 7*, 5-6 (1997), 303–325.

[20] do Nascimento, H. A. D., Eades, P., and de Mendonça Neto, C. F. X. A multiagent approach using A-teams for graph. *Proceedings of the ISCA 9th International Conference on Intelligent Systems* (2000), 39–42.

[21] do Nascimento, H. A. D., Stolfi, J., and de Mendonça Neto, C. F. X. Heuristics and pedigrees for drawing directed graphs. *Journal of Brazilian Computer Society 6*, 1 (1999), 52–63.

[22] Ebert, J. *st*-ordering the vertices of biconnected graphs. *Computing 30* (1983), 19–33.

[23] Eggleton, R. B., and Guy, R. P. The crossing number of the n-cube. *Amer. Math. Soc. Notices 17* (1970), 757.

[24] Erdös, P., and Guy, R. P. Crossing number problems. *Amer. Math. Monthly 80* (1973), 52–58.

[25] Even, S., and Tarjan, R. E. Computing and *st*-numbering. *Theoretical Computer Science 2* (1976), 339–344.

[26] Even, S., and Tarjan, R. E. Corrigendum: Computing and *st*-numbering. *Theoretical Computer Science 4* (1977), 123.

[27] Faria, L. *Alguns Invariantes de Não-Planaridade: Uma Abordagem Estrutural de Complexidade*. PhD thesis, COPPE / Sistemas e Computação, Universidade Federal do Rio de Janeiro, Brazil, August 1998.

[28] Faria, L., and de Figueiredo, C. M. H. On Eggleton and Guy conjectured upper bound for the crossing number of the n-cube. *Mathematica Slovaca 50*, 3 (2000), 271–287.

[29] Faria, L., de Figueiredo, C. M. H., and de Mendonça Neto, C. F. X. Splitting number is NP-Complete. *Discrete Applied Mathematics 108* (2001), 65–83.

[30] Faria, L., de Figueiredo, C. M. H., Gravier, G., de Mendonça, C. F. X., and Stolfi, J. Nonplanar vertex deletions: maximum degree thresholds for NP/max SNP-Hardness and a $\frac{3}{4}$-approximation for finding maximum planar induced subgraphs. In *Proceedings of LACGA 2004* (2004), vol. 18 of *Electronic Notes in Discrete Mathematics*, pp. 121–126.

[31] Faria, L., de Figueiredo, C. M. H., Sýkora, O., and Vrťo, I. An improved upper bound on the crossing number of the hypercube. *Lecture Notes in Computer Science 2880* (2003), 230–236.

[32] Garey, M. R., and Johnson, D. S. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, CA, 1979.

[33] Garey, M. R., and Johnson, D. S. Crossing number is NP-Complete. *SIAM Journal of Alg. Discr. Meth. 4*, 3 (1983), 312–316.

[34] Geldmacher, R. C., and Liu, P. C. On the deletion of nonplanar edges of a graph. *Congr. Numer. 24* (1979), 727–738.

[35] Glebsky, L., and Salazar, G. The conjecture $cr(C_m \times C_n) = (m-2)n$ holds for all but finitely many values of n, for each m. *Journal of Graph Theory*. Submetido.

[36] Guy, R. K. The decline and fall of zarankiewicz's theorem. In *Proof Techniques in Graph Theory* (1968), Proceedings of Second Ann Arbor Graph Theory Conf., Ann Arbor, Mich., 1968, Academic Press, pp. 63–69.

[37] Guy, R. K. Latest results on crossing numbers. In *Recent Trends in Graph Theory* (1971), M. Capobianco, J. B. Frechen, and M. Krolik, Eds., vol. 186 of *Lecture Notes in Mathematics*, Proceedings of the First New York City Graph Theory Conference, June 11–13, 1970, Springer-Verlag, pp. 143–156.

[38] Guy, R. K. Crossing numbers of graphs. In *Graph Theory and Applications* (1972), Y. Alavi, D. R. Lick, and A. T. White, Eds., vol. 303 of *Lecture Notes in Mathematics*, Proceedings of the Conference at Western Michigan University, May 10–13, 1972, Springer-Verlag, pp. 111–124.

[39] Harary, F., Kainen, P. C., and Schwenk, A. J. Toroidal graphs with arbitrarily high crossing number. *Nanta Mathematics 6* (1973), 58–67.

[40] Hliněný, P. Crossing number is hard for cubic graphs. In *Math Foundations of Computer Science MFCS 2004* (2004), vol. 3153 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 772–782.

[41] Hopfield, J. J. Neural networks and physical systems with emergent collective computational habilities. *Proc. Natl. Acad. Science - Byophysics 79* (1982), 2554–2558.

[42] Hopfield, J. J., and Tank, D. Neural computation of decisions in optimization problems. *Biological Cybernetics 52* (1985), 141–152.

[43] Juarez, H. A., and Salazar, G. Drawings of $C_m \times C_n$ with one disjoint family ii. *Journal of Graph Theory Series B 82* (2001), 161–165.

[44] Kirkpatrick, S., Gelatt, C. D., and Cecchi, M. P. Optimization by simulated annealing. *Science 220*, 4598 (May 1983), 671–680.

[45] Kleitman, D. The crossing number of $K_{5,n}$. *Journal of Combinatorial Theory 9* (1970), 315–323.

[46] Klesc, M., Richter, R. B., and Stobert, I. The crossing number of $C_5 \times C_n$. *Journal of Graph Theory 22* (1996), 239–243.

[47] Kuratowski, K. *Sur le probleme dès courbes gauches en topologie*, vol. 15. Fundamenta Mathematicae, 1930.

[48] Leighton, F. T. New lower bound techniques for VLSI. *Math. Systems Theory 17* (1984), 47–70.

[49] Lempel, A., Even, S., and Cederbaum, I. An algorithm for planarity testing of graphs. In *Theory of Graphs: Internat. Symposium (Rome 1966)* (New York, 1967), Gordon and Breach, pp. 215–232.

[50] Liebers, A. Planarizing graphs - a survey and annotated bibliography. *Journal of Graph Algorithms and Applications 5*, 1 (2001), 1–74.

[51] Madej, T. Bounds for the crossing number of the n-cube. *Journal of Graph Theory 15* (1991), 81–97.

[52] Masuda, S., Nakajima, K., Kashiwabara, T., and Fujisawa, T. Crossing minimization in linear embeddings of graphs. *IEEE Transac. Comput. 39*, 1 (1990), 124–127.

[53] McCulloch, W. S., and Pitts, W. H. A logical calculus of ideas imminent in nervous activity. *Bull. Math. Biophys 5* (1943), 115–133.

[54] Nahas, N. On the crossing number of $K_{m,n}$. *Eletronic Journal of Combinatorics 10* (2003). Note 8.

[55] Nicholson, T. A. J. Permutation procedure for minimising the number of crossings in a network. *Proc. IEE 115*, 1 (1968), 21–26.

[56] Papamanthou, C. Computing longest path parameterized *st*-orientations of graphs: algorithms and applications. Master's thesis, University of Crete, Heraklion, Greece, July 2005.

[57] Richter, R. B., and Salazar, G. The crossing number of $C_6 \times C_n$. *Australasian Journal of Combinatorics 23* (2001), 135–143.

[58] Richter, R. B., and Thomassen, C. Intersections of curve systems and the crossing number of $C_5 \times C_5$. *Discrete & Computational Geometry 13* (1995), 149–159.

[59] Ringeisen, R. D., and Beineke, L. W. The crossing number of $C_3 \times C_n$. *Journal of Combinatorial Theory Ser. B 24* (1978), 134–136.

[60] Sachdev, S., Paredis, C. J. J., Gupta, S. K., and Talukdar, S. N. 3D spatial layouts using A-teams. *Proceedings of ASME 24th Design Automation Conference* (1998).

[61] Salazar, G. On the crossing number of $C_m \times C_n$. *Journal of Graph Theory 28* (1998), 163–170.

[62] Salazar, G., and Ugalde, E. An improved bound for the crossing number of $C_m \times C_n$: a self-contained proof using mostly combinatorial arguments. *Graphs and Combinatorics 20* (2004), 247–253.

[63] Shahrokhi, F., Sýkora, O., Székely, L., and Vrťo, I. Crossing numbers: Bounds and applications. In *Intuitive Geometry* (1997), K. B. I. Barany, Ed., vol. 6, Bolyai Soc. Math. Studies, Akademia Kiado, Budapest, pp. 179–206.

[64] Shahrokhi, F., Sýkora, O., Székely, L. A., and Vrťo, I. Book embeddings and crossing numbers. In *WG'94: Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science* (Berlin, 1995), vol. 903 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 256–268.

[65] Shahrokhi, F., Sýkora, O., Székely, L. A., and Vrťo, I. Intersections of curve and the crossing number of $C_m \times C_n$ in surfaces. *Discrete & Computational Geometry 19* (1998), 237–247.

[66] Shahrokhi, F., Székely, L. A., Sýkora, O., and Vrťo, I. The book crossing number of a graph. *Journal of Graph Theory 21*, 4 (1996), 413–424.

[67] Sýkora, O., and Vrťo, I. On the crossing number of hypercubes and cube connected cycles. *BIT 33* (1993), 232–237.

[68] Talukdar, S. N. Autonomous cyber agents: rules for collaboration. *Proceedings of the thirty-first Hawai International Conference on System Sciences (HICSS-31)* (January 1998).

[69] Talukdar, S. N., Baerentzen, L., Gove, A., and de Souza, P. S. Asynchronous teams: cooperation schemes for autonomous agents. *Journal of Heuristics 4*, 4 (1998), 295–321.

[70] Talukdar, S. N., and de Souza, P. S. Scale efficient organizations. *IEEE International Conference on Systems, Man and Cybernetics* (1992).

[71] Tarjan, R. Depth-first search and linear graph. *SIAM Journal of Comput. 1* (1972), 146–160.

[72] Tarjan, R. E. Two streamlined depth-first search algorithms. *Fundamenta Informaticae 9* (1986), 85–94.

[73] Wang, R. L., and Okazaki, K. An efficient parallel algorithm for the minimum crossing number problem. *Neurocomputing 64* (Agosto 2005), 411–416.

[74] Woodall, D. R. Cyclic-order graphs and Zarankiewics's crossing number conjecture. *Journal of Graph Theory 17* (1993), 657–671.

[75] Yang, Y., Lin, X., Lü, J., and Hao, X. The crossing number of $C(n; \{1, 3\})$. *Discrete Mathematics 289* (2004), 107–118.

[76] Yang, Y., and Zhao, C. The crossing number of $C(n; \{1, k\})$. *National symposium on software technology held by Chinese Computer Institute 2001* (2001), 134–136.

[77] Yannakakis, M. Node and edge-deletion NP-Complete problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing* (1978), pp. 253–264.

[78] Zarankiewicz, K. On a problem of p. turán concerning graphs. *Fundamenta Informaticae 41* (1954), 137–145.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo